Product Status

The information in this document is Final, that is for a developed product.

Web Address

http://www.arm.com
**Contents**

**ARM®v8-M Exception Handling**

---

**Preface**

About this book .............................................................................................................. 8
Feedback ..................................................................................................................... 10

---

**Chapter 1**  
**The ARM®v8-M exception model**

1.1 Exception handling .................................................................................................. 1-12
1.2 Exceptions in Thread and Handler modes .............................................................. 1-13
1.3 Types of exception .................................................................................................. 1-15
1.4 Exception properties ................................................................................................ 1-17
1.5 Exception handlers .................................................................................................. 1-20
1.6 Preemption .............................................................................................................. 1-21
1.7 Exception entry ....................................................................................................... 1-22
1.8 Stack frame ............................................................................................................. 1-23
1.9 Exception return ...................................................................................................... 1-26
1.10 EXC_RETURN register .......................................................................................... 1-27
List of Figures

ARM®v8-M Exception Handling

Figure 1  Key to timing diagram conventions ............................................................................................. 9
List of Tables

ARM®v8-M Exception Handling

Table 1-1  Exception number .................................................................................................................. 1-17
Table 1-2  Exception priorities ............................................................................................................... 1-19
Table 1-3  EXC_RETURN bit assignments ............................................................................................ 1-27
Preface

This preface introduces the *ARM® v8-M Exception Handling*. It contains the following:

About this book

Describes the fundamentals of Exception handling on ARMv8-M processors.

Product revision status

The rmpn identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rm Identifies the major revision of the product, for example, r1.

pn Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

Using this book

This book is organized into the following chapters:

Chapter 1 The ARM®v8-M exception model

Exceptions are conditions or system events that usually require remedial action or an update of system status by privileged software to ensure smooth functioning of the system. The ARM®v8-M exception model describes how the processor responds to an exception, the properties that are associated with each exception, such as its priority level, and the exception return behavior.

Glossary

The ARM Glossary is a list of terms used in ARM documentation, together with definitions for those terms. The ARM Glossary does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See the ARM Glossary for more information.

Typographic conventions

italic Introduces special terminology, denotes cross-references, and citations.

bold Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

monospace Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace italic Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

monospace bold Denotes arguments to monospace text where the argument is to be replaced by a specific value.

<and> Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>

SMALL CAPITALS Used in body text for a few terms that have specific technical meanings, that are defined in the ARM glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

![Timing Diagrams](image)

Figure 1  Key to timing diagram conventions

Signals

The signal conventions are:

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name denotes an active-LOW signal.
Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:
• The product name.
• The product revision or version.
• An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to errata@arm.com. Give:
• The title ARM®v8-M Exception Handling.
• The number ARM 100701_0100_00_en.
• If applicable, the page number(s) to which your comments refer.
• A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.
Chapter 1
The ARM®v8-M exception model

Exceptions are conditions or system events that usually require remedial action or an update of system status by privileged software to ensure smooth functioning of the system. The ARM®v8-M exception model describes how the processor responds to an exception, the properties that are associated with each exception, such as its priority level, and the exception return behavior.

It contains the following sections:
• 1.1 Exception handling on page 1-12.
• 1.2 Exceptions in Thread and Handler modes on page 1-13.
• 1.3 Types of exception on page 1-15.
• 1.4 Exception properties on page 1-17.
• 1.5 Exception handlers on page 1-20.
• 1.6 Preemption on page 1-21.
• 1.7 Exception entry on page 1-22.
• 1.8 Stack frame on page 1-23.
• 1.9 Exception return on page 1-26.
• 1.10 EXC_RETURN register on page 1-27.
1.1 Exception handling

Exceptions are conditions or system events that usually require remedial action or an update of system status by privileged software to ensure smooth functioning of the system.

This is called handling an exception. There is typically an exception handler associated with each exception type.

Hardware external to the processor core usually signals an interrupt, but there are several other events that can cause the core to take an exception, for example, internal events such as OS calls using the SVC instruction.
1.2 Exceptions in Thread and Handler modes

The processor mode changes when an exception occurs in Thread mode, but does not change when an exception occurs in Handler mode.

The processor mode, states, and privilege level provide multiple possible combinations, as the following figure shows:

![Diagram showing possible combinations of processor mode, states, and privilege level]

When in Thread mode, execution can be privileged or unprivileged. The stack pointer used can be the Main Stack Pointer (MSP) or Process Stack Pointer (PSP). When in Handler mode, the processor is Privileged. The stack pointer is always MSP.

The following sequence and figure show the possible stages in exception handling.

1. Reset: Enters Privileged Thread mode using the MSP.
2. Internal exception taken:
   - Switch to Handler mode.
   - Run exception handler.
   - Return to Privileged Thread Mode +Thread+MSP).
3. Change from PT(Privileged +Thread) to UT(Unprivileged + Thread).
4. External exception.
   - Switch to Handler mode.
   - Run exception handler.
5. Exception handler is interrupted.
   - Re-enter Handler mode
   - Service higher priority interrupt
   - Return to Handler mode.
   - Return to UT.
1.3 Types of exception

The exception model for the ARMv8-M architecture provides various types of exception. Each exception is handled in turn before returning to the original application. When multiple exceptions occur simultaneously, they are handled in a fixed order of priority.

The following types of exception exist:

Reset

Reset is invoked on power-up or a Warm reset. The ARMv8-M exception model treats reset as a special form of exception. It is permanently enabled. When reset occurs, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address that is provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.

NMI

A Non-maskable Interrupt (NMI) is signaled by a peripheral or triggered by software. This is the highest priority exception other than reset. It is permanently enabled. NMIs cannot be:

• Masked or prevented from activation by any other exception.
• Preempted by any exception other than Reset.

HardFault

A HardFault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism.

MemManage

A MemManage fault is an exception that occurs because of a memory protection-related fault. The MPU or the fixed memory protection constraints determines this fault, for both instruction and data memory transactions. This fault is always used to abort instruction accesses to Execute Never (XN) memory regions.

BusFault

A BusFault is an exception that occurs because of a memory-related fault for an instruction or data memory transaction. This might be from an error that is detected on a bus in the memory system.

UsageFault

A UsageFault is an exception that occurs because of a fault that is related to instruction execution. This includes:

• An UNDEFINED instruction.
• An illegal unaligned access.
• Invalid state on instruction execution.
• An error on exception return.

The following can cause a UsageFault when the core is configured to report them:

• An unaligned address on word and halfword memory access.
• Division by zero.

SVCall

A Supervisor Call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.
PendSV

PendSV is an interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active.
The SVC and PendSV exceptions are banked between Secure and Non-secure states. When the SVC instruction is executed:

- If the processor is in Secure state, the SVC exception handling sequence fetches the exception vector from the Secure vector table and executes the SVCall handler in Secure state.
- If the processor is in Non-secure state, the SVC exception handling sequence fetches the exception vector from the Non-secure vector table and executes the SVCall handler in Non-secure state.

Similarly, the PendSVSet and PendSVClr bit in the Interrupt Control and State Register (ICSR) is banked. Secure software can also trigger a Non-secure PendSV using the Non-secure alias of ICSR (0xE002ED04).

The priority level registers for SVC and PendSV are also banked between Secure and Non-secure states.

SysTick

A SysTick exception is an exception the system timer generates when it reaches zero. Software can also generate a SysTick exception. In an OS environment, the processor can use this exception as system tick.

SysTick can exist in neither, either or both (banked) Security states.

Interrupt (IRQ)

An interrupt request, or IRQ, is an exception signaled by a peripheral, or generated by a software request. All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor.

Interrupts default to Secure state but can be retargeted to non-secure state by Secure software.

SecureFault

For ARMv8-M architecture with Main Extension the architecture defines an extra SecureFault exception. This exception is triggered by the various security checks that are performed. It is triggered, for example, when jumping from Non-secure code to an address in Secure code that is not marked as a valid entry point. Most systems choose to treat a SecureFault as a terminal condition that either halts or restarts the system. Any other handling of the SecureFault must be checked carefully to make sure that it does not inadvertently introduce a security vulnerability.

SecureFaults always target the Secure state.
1.4 Exception properties

Each exception has an associated identification number, a vector address that is the exception entry point in memory, and a priority level which determines the order in which multiple pending exceptions are handled (the lower the priority number, the higher the priority level).

This section contains the following subsections:

• 1.4.1 Exception number on page 1-17.
• 1.4.2 Vector address on page 1-17.
• 1.4.3 Exception priorities on page 1-19.

1.4.1 Exception number

Each exception has an associated exception number.

<table>
<thead>
<tr>
<th>Exception number</th>
<th>Name</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reset</td>
<td>Secure</td>
</tr>
<tr>
<td>2</td>
<td>Non Maskable Interrupt</td>
<td>Configurable</td>
</tr>
<tr>
<td>3</td>
<td>Secure HardFault</td>
<td>Secure</td>
</tr>
<tr>
<td>3</td>
<td>Non-secure Hard Fault</td>
<td>Non-secure</td>
</tr>
<tr>
<td>4</td>
<td>MemManage</td>
<td>Banked</td>
</tr>
<tr>
<td>5</td>
<td>BusFault</td>
<td>Configurable</td>
</tr>
<tr>
<td>6</td>
<td>UsageFault</td>
<td>Banked</td>
</tr>
<tr>
<td>11</td>
<td>SVC</td>
<td>Banked</td>
</tr>
<tr>
<td>12</td>
<td>DebugMonitor</td>
<td>Configurable</td>
</tr>
<tr>
<td>14</td>
<td>PendSV</td>
<td>Banked</td>
</tr>
<tr>
<td>15</td>
<td>SysTick</td>
<td>Banked</td>
</tr>
<tr>
<td>16+N</td>
<td>Interrupts #0 -N</td>
<td>Configurable</td>
</tr>
</tbody>
</table>

1.4.2 Vector address

The vector table contains the reset value of the Stack Pointer and the start addresses, also called exception vectors, for all exception handlers.

The following figure shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is code.
The first entry contains the initial Stack Pointer. All other entries are the addresses for the exception handlers.

On system reset, the vector table is at an implementation defined address. Privileged software can write to the Vector Table Offset Register (VTOR) to relocate the vector table start address to a different memory location. It is implementation defined which bits are writable.

The silicon vendor must configure the top range value, which depends on the number of interrupts implemented. The minimum alignment is 32 words, enough for up to 16 interrupts. For more interrupts, adjust the alignment by rounding up to the next power of two. For example, if you require 21 interrupts, the alignment must be on a 64-word boundary because the required table size is 37 words, and the next power of two is 64.

ARM recommends that you locate the vector table in the CODE, SRAM, External RAM, or External Device areas of the system memory map.

Using the Peripheral, Private Peripheral Bus, or Vendor-specific memory areas can lead to unpredictable behavior in some systems. This is because the processor might use different interfaces for load/store instructions and vector fetch in these memory areas. If the vector table is located in a region of memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Exception number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40+4*N</td>
<td>External interrupt N: 16+N</td>
</tr>
<tr>
<td>0x40</td>
<td>External interrupt 0: 16</td>
</tr>
<tr>
<td>0x3C</td>
<td>SysTick: 15</td>
</tr>
<tr>
<td>0x38</td>
<td>PendSV: 14</td>
</tr>
<tr>
<td>0x34</td>
<td>Reserved: 13</td>
</tr>
<tr>
<td>0x30</td>
<td>Debug monitor: 12</td>
</tr>
<tr>
<td>0x2C</td>
<td>SVC: 11</td>
</tr>
<tr>
<td>0x28</td>
<td>Reserved: 8-10</td>
</tr>
<tr>
<td>0x20</td>
<td>SecureFault: 7</td>
</tr>
<tr>
<td>0x1C</td>
<td>UsageFault: 6</td>
</tr>
<tr>
<td>0x18</td>
<td>BusFault: 5</td>
</tr>
<tr>
<td>0x14</td>
<td>MemManage: 4</td>
</tr>
<tr>
<td>0x10</td>
<td>HardFault: 3</td>
</tr>
<tr>
<td>0x0C</td>
<td>NMI: 2</td>
</tr>
<tr>
<td>0x08</td>
<td>Reset: 1</td>
</tr>
<tr>
<td>0x04</td>
<td>SP main: N/A</td>
</tr>
</tbody>
</table>
that is cacheable, you must treat any store to the vector as self-modifying code and use cache maintenance instructions to synchronize the update.

### 1.4.3 Exception priorities

All exceptions have an associated priority, with a lower priority value indicating a higher priority to the exception, and configurable priorities for all exceptions except Reset, HardFault, and NMI. If software does not configure any priorities, all exceptions with a configurable priority have a priority of 0. Lower priority numbers take precedence.

#### Table 1-2 Exception priorities

<table>
<thead>
<tr>
<th>Name</th>
<th>Exception Priority #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupts #0 -N</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>SysTick</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>PendSV</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>DebugMonitor</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>SVCAll</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>UsageFault</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>BusFault</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>MemManage</td>
<td>0-255 (programmable)</td>
</tr>
<tr>
<td>Non-secure Hard Fault</td>
<td>-1</td>
</tr>
<tr>
<td>Secure HardFault</td>
<td>-3 or -1 (programmable)</td>
</tr>
<tr>
<td>Non Maskable Interrupt</td>
<td>-2</td>
</tr>
<tr>
<td>Reset</td>
<td>-4</td>
</tr>
</tbody>
</table>

**Note**

Configurable priority values are in the range 0-255. This means that the Reset, HardFault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

**Note**

For the ARMv8-M architecture, the available programmable priorities are limited to pre-empting priorities of 0, 64, 128 and 192. For the ARMv8-M architecture with Main Extension, the number of priority bits is between 3 and 8 inclusive, left-justified in a 8-bit field. However, the available range of pre-empting priorities is programmable and at most the top 7 bits, giving 0-254 in multiples of 2.

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].
1.5 Exception handlers

The processor handles exceptions using Interrupt Service Routines (ISRs), fault handlers, and system handlers.

**Interrupt Service Routines**
Handle interrupt exceptions for IRQ0-IRQ239.

**Fault handlers**
Handle fault exceptions for HardFault, MemManage fault, UsageFault, and BusFault.

**System handlers**
Handle system exceptions for NMI, PendSV, SVCall, SysTick, and system faults.

1.5.1 Writing exception handlers

Programmers should note the following points when writing exception handlers for the ARMv8-M processor in C code or Assembly language.

The following general points should be noted when writing exception handlers:

- The ARMv8-M NVIC supports level-sensitive and pulse-sensitive IRQs. For interrupt sources that generate level-sensitive requests, clear the interrupt source. Some system handlers, such as SysTick, do not need to be cleared.
- Be careful of Main stack overflows. Take care with the additional stack requirements on the Secure Main stack in systems that support both Secure and Non-secure interrupts.
- There is no requirement to set the STKALIGN bit as in ARMv7-M. Bit[9] in the Configuration and Control Register is a RES1 bit.

The following points should be noted if writing exception handlers in C code:

- The ISR function must be type `void` and cannot accept arguments.
- The `__irq` keyword is optional but is recommended for clarity. For example,
  ```c
  __irq void SysTickHandler (void) { }
  ```
- No special assembly language entry or exit code is required.

The following points should be noted if writing exception handlers in Assembler language:

- Handlers must manually save and restore any non-scratch registers which are used, such as:
  - R4
  - R5
  - R6
  - R7
  - R8
  - R9
  - R10
  - R11
  - LR.
- Handlers must maintain 8-byte stack alignment if making function calls.
- Handlers must return from interrupt using EXC_RETURN with the proper instruction.
  - LDR PC
  - LDM and POP, which includes loading the PC
  - BX LR
1.6 Preemption

When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled.

If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

When one exception preempts another the exceptions are called nested exceptions.

For example, consider the following exceptions where all have software configurable priority numbers:

- A has the highest priority.
- B has medium priority.
- C has lowest priority.

Example sequence of events:
1. Exception B occurs. The processor takes the exception and starts executing the handler for this exception. The execution priority is priority B.
2. Exception A occurs. Because its priority is higher than the execution priority, it preempts exception B and the processor starts executing the Exception A handler. Execution priority is now priority A.
3. Exception C occurs. Its priority is less than the execution priority so its status is pending
4. The handler for exception A reduces the priority of exception A, to a priority lower than priority C. The execution priority falls to the highest priority of all active exceptions. This is priority B. Exception C remains pending because its priority is lower than the current execution priority.

Only a pending exception with higher priority than priority B can preempt the current exception handler. Therefore, a new exception with lower priority than exception B cannot take precedence over the preempted exception B

1.6.1 Late-arriving

The late-arriving mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. It is implementation defined whether a processor does this.

There are several combinations of security states where the state saving might be affected by a late-arriving interrupt.

If a higher priority late-arriving Secure exception occurs during entry to a Non-secure exception when the Background Security state is Secure, it is implementation defined whether the stacking of the additional state context is rolled back.

The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.
1.7 Exception entry

Exception entry occurs when there is a pending exception with higher priority than the currently-executing context, in which case the new exception preempts this context.

Sufficient priority means that the exception has more priority than any limits set by the mask registers. An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is called stacking and the block of data written to the stack is referred as the stack frame. The basic stack frame contains eight words of data.

When using floating-point routines, or where an exception causes a change of security state, the processor might automatically stack additional registers to form an extended stack frame.

Note
Where stack space for floating-point state or security state is not allocated, the stack frame is the same as that of ARMv8-M implementations without an FPU.

Immediately after stacking, the Stack Pointer indicates the lowest address in the stack frame. ARMv8-M requires that the stack frame is doubleword aligned, and will automatically decrement the stack pointer if necessary to enforce this.

The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.

In parallel to the stacking operation the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the LR. This indicates which Stack Pointer corresponds to the stack frame and what operational mode the processor was in before the entry occurred.

If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.

When an exception occurs the processor automatically transitions to the security state associated with that exception. This transition avoids the need for an SG instruction at the start of the Secure exception handlers, which would result in them being directly callable from the Non-secure state. If the exception and the exception vector target address are associated with different security states, a SecureFault is raised unless the exception is associated with the Non-secure state and targets an SG instruction. The Baseline profile has no SecureFault, so uses a HardFault exception instead.
1.8 Stack frame

For processors that do not implement the Security Extension, the stack frame format is divided into two sections, integer caller saved, and floating-point caller saved registers. To accommodate the additional state that must be preserved when an exception causes a transition from the Secure to the Non-secure state the format is extended.

The only mandatory section of the stack frame is the integer caller saved registers, with all other sections being optionally created depending on the state transitions being performed. ARMv8-M-profile cores contain hardware to automatically stack and unstack the caller saved register state around exceptions, as defined in the ARM procedural calling standard. This reduces interrupt latency and enables extra optimizations like tail chaining. The approach taken by the ARMv8-M architecture with Security Extension is to extend this mechanism to stack extra register contents if an exception causes a transition from the Secure to the Non-secure state. Some events also cause the processor to clear the registers. This prevents Secure data being visible to a Non-secure exception handler.

The following figure shows the structure of the stack frame:
This section contains the following subsections:

- **1.8.1 Stack limit** on page 1-25.
1.8.1 Stack limit

Multiple nested interrupts or other techniques might be used to overflow the Secure stack, and therefore potentially overwrite other Secure data in memory.

To protect against this possibility, there are two stack limit registers, MSPLIM_S and PSPLIM_S. These registers limit the extent to which the Main and Process Secure Stack Pointers respectively can descend. Violation of the stack limit raises a synchronous UsageFault.

The ARMv8-M architecture with Main Extension also provides Non-secure stack limit registers.
1.9 Exception return

An exception return occurs when the processor is in Handler mode and executes a suitable instruction that loads the EXC_RETURN value into the PC.

An exception return also occurs when:

- There is no pending exception with sufficient priority to be serviced.
  - The completed exception handler was not handling a late-arriving exception.

The processor pops the stack and restores the processor state to the state it had before the interrupt occurred.

Any of the following instructions cause an exception return when the processor is in Handler mode:
- An LDM or POP instruction that loads the PC.
- An LDR instruction with PC as the destination.
- A BX instruction using any register.

1.9.1 Tail-chaining

The ARMv8-M architecture tail-chaining mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.
1.10 **EXC\_RETURN register**

The EXC\_RETURN register is used to communicate additional information about which state to return to after handling an exception, and which registers need to be unstacked.

EXC\_RETURN is the value that is loaded into the LR on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest 5 bits of this value provide information on the return stack and processor mode. The following table shows the EXC\_RETURN values with a description of the exception return behavior.

The EXC\_RETURN register has the following bit assignments:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:24]</td>
<td>0xFF</td>
<td>Prefix. Indicates that this is an EXC_RETURN value. This field reads as <code>0b11111111</code>.</td>
</tr>
<tr>
<td>[7:23]</td>
<td></td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
| [6]    | S         | Indicates whether the stack frame to restore from is stored on a Secure or Non-secure stack. It is used together with the Mode bit and CONTROL\_SPSEL to determine which of the four Stack Pointers is used to unstack the register state. The possible values of this bit are:  
  | 0       | Non-secure stack used.                                                     |
  | 1       | Secure stack entry.                                                        |
| [5]    | DCRS      | Default callee register stacking. Indicate whether the default stacking rules should apply, or whether the callee registers have already been pushed onto the stack and therefore do not need to be stacked again. The possible values of this bit are:  
  | 0       | Stacking of the callee saved registers skipped.                            |
  | 1       | Default rules for stacking the callee registers followed.                  |
Table 1-3  EXC_RETURN bit assignments (continued)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4]</td>
<td>Ftype</td>
<td>Stack frame type. Indicates whether the stack frame is a standard integer-only stack frame, or an extended floating-point stack frame. The possible values of this bit are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>[3]</td>
<td>Mode</td>
<td>The Mode to return to (as in the existing ARMv7-M architecture). The possible values of this bit are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>[2]</td>
<td>SPSEL</td>
<td>Stack pointer selection. Indicates which stack pointer the exception frame resides on. The possible values of this bit are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>[0]</td>
<td>ES</td>
<td>Indicates the security domain the exception is taken to:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>