ARM® Compiler

Version 6.02

Errors and Warnings Reference Guide

ARM

ARM DUI0807C

ARM® Compiler

Errors and Warnings Reference Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
A	14 March 2014	Non-Confidential	ARM Compiler v6.00 Release
B	15 December 2014	Non-Confidential	ARM Compiler v6.01 Release
C	30 June 2015	Non-Confidential	ARM Compiler v6.02 Release

Non-Confidential Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

http://www.arm.com

Contents

ARM® Compiler Errors and Warnings Reference Guide

Preface

About this book 6

Chapter 1 Assembler Errors and Warnings

1.1 List of the armasm error and warning messages 1-9

Chapter 2 Linker Errors and Warnings

2.1 Suppressing armlink error and warning messages 2-38

2.2 List of the armlink error and warning messages 2-39

Chapter 3 ELF Image Converter Errors and Warnings

3.1 List of the fromelf error and warning messages 3-79

Chapter 4 Librarian Errors and Warnings

4.1 List of the armar error and warning messages 4-82

Chapter 5 Other Errors and Warnings

5.1 Internal faults and other unexpected failures 5-84

5.2 List of other error and warning messages 5-85

Preface

This preface introduces the *ARM® Compiler Errors and Warnings Reference Guide*.

It contains the following:

- *About this book* on page 6.

About this book

The ARM® Compiler Errors and Warnings Reference Guide provides lists of the errors and warnings that each of the compilation tools can generate. It does not include errors and warnings produced by `armclang`.

Using this book

This book is organized into the following chapters:

Chapter 1 Assembler Errors and Warnings
Describes the error and warning messages for the assembler, `armasm`.

Chapter 2 Linker Errors and Warnings
Describes the error and warning messages for the linker, `armlink`.

Chapter 3 ELF Image Converter Errors and Warnings
Describes the error and warning messages for the ELF image converter, `fromelf`.

Chapter 4 Librarian Errors and Warnings
Describes the error and warning messages for the ARM librarian, `armar`.

Chapter 5 Other Errors and Warnings
Describes error and warning messages that might be displayed by any of the tools.

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.

`mono`space
Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

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

`monospace bold`
Denotes language keywords when used outside example code.

`<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.

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.
- The number ARM DUI0807C.
- 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.

Other information

- *ARM Information Center*.
- *ARM Technical Support Knowledge Articles*.
- *Support and Maintenance*.
- *ARM Glossary*.

Chapter 1
Assembler Errors and Warnings

Describes the error and warning messages for the assembler, `armasm`.

It contains the following sections:

- *1.1 List of the armasm error and warning messages* on page 1-9.

1.1 List of the armasm error and warning messages

A list of the error and warning messages that `armasm` produces.

`A1017E: :INDEX: cannot be used on a pc-relative expression`
: The `:INDEX:` expression operator has been applied to a PC-relative expression, most likely a program label. `:INDEX:` returns the offset from the base register in a register-relative expression.

  If you require the offset of a label called `<label>` within an area called `<areaname>`, use `<label> - <areaname>`.

  See the following in the *armasm User Guide*:

  Unary operators.

`A1020E: Bad predefine: <directive>`
: The operand to the `--predefine` or `--pd` command-line option was not recognized. The directive must be enclosed in quotes if it contains spaces, for example on Windows:

  ```
  --predefine "versionnum SETA 5"
  ```

  If the `SETS` directive is used, the argument to the directive must also be enclosed in quotes, which might require escaping, depending on the operating system and shell. For example:

  ```
  --predefine "versionstr SETS \"5A\""
  ```

`A1021U: No input file`
: No input file was specified on the command line. This might be because there was no terminating quote on a quoted argument.

`A1023E: File "<filename>" could not be opened: <reason>`

`A1024E: File "<filename>" could not all be loaded: <reason>`

`A1042E: Unrecognized APCS qualifier '<qualifier>'`
: There is an error in the argument given to the `--apcs` command line option. Check the spelling of `<qualifier>`.

`A1051E: Cannot open --depend file '<filename>': <reason>`

`A1055E: Cannot open --errors file '<filename>': <reason>`

`A1056E: Target cpu '<cpu>' not recognized`
: The name given in the `--cpu` command line option is not a recognized processor name. Check the spelling of the argument.

  Use `--cpu=list` to list the supported processors and architectures.

`A1067E: Output file specified as '<filename1>', but it has already been specified as '<filename2>'`
: More than one output file, `-o filename`, has been specified on the command line. Misspelling a command line option can cause this.

`A1071E: Cannot open listing file '<filename>': <reason>`
: The file given in the `--list <filename>` command-line option could not be opened. This could be for any of the following reasons:
  - The given name is not valid.
  - There is no space.
  - A read-only file with the same name already exists.
  - The file is in use by another process.

  Check you have specified the correct path for the file.

`A1072E: The specified listing file '<filename>' must not be a .s or .o file`
: The filename argument to the `--list` command line option has an extension that indicates it is a source or object file. This might be because the filename argument was accidentally omitted from the command line. Check that the correct argument is given to the `--list` command line option.

`A1073E: The specified output file '<filename>' must not be a source file`
The object file specified on the command line has a filename extension that indicates it is a source file. This might be because the object filename was accidentally omitted from the command line.

`A1074E: The specified depend file '<filename>' must not be a source file`
The filename argument to the `--depend` command line option has an extension that indicates it is a source (`.s`) file. This might be because the filename argument was accidentally omitted from the command line. Check that the correct arguments are given.

`A1075E: The specified errors file '<filename>' must not be a source file`
The filename argument to the `--errors` command line option has an extension that indicates it is a source (`.s`) file. This might be because the filename argument was accidentally omitted from the command line. Check that the correct arguments are given.

`A1085W: Forced user-mode LDM/STM must not be followed by use of banked R8-R14`
The ARM architecture does not permit you to access banked registers in the instruction immediately following a User registers `LDM` or `STM`. Adding a `NOP` immediately after the `LDM` or `STM` is one way to avoid this.

For example:

```
stmib sp, {r0-r14}^ ; Return a pointer to the frame in a1.
mov r0, sp
```

change to:

```
stmib sp, {r0-r14}^ ; Return a pointer to the frame in a1.
nop
mov r0, sp
```

`A1088W: Faking declaration of area AREA |$$$$$$$|`
This is reported when no `AREA` directive is specified. See also message number A1105E.

`A1099E: Structure stack overflow max stack size <max>`

`A1100E: Structure stack underflow`

`A1105E: Area directive missing`
This is reported when no `AREA` directive is specified. See also message number A1088W.

`A1106E: Missing comma`

`A1107E: Bad symbol type, expect label`

`A1108E: Multiply defined symbol '<name>'`

`A1109E: Bad expression type`

`A1110E: Expected constant expression`
A constant expression was expected after, for example, `SETA`.

See the following in the *armasm User Guide*:

Numeric expressions.

`A1111E: Expected constant or address expression`

`A1112E: Expected address expression`

`A1113E: Expected string expression`
A string expression was expected after, for example, `SETS`.

See the following in the *armasm User Guide*:

String expressions.

`A1114E: Expected register relative expression`

`A1116E: String operands can only be specified for DCB`

`A1117E: Register symbol '<name>' already defined`

`A1118E: No current macro expansion`

A1119E: MEND not allowed within conditionals

MEND means END of Macro (not the English word mend).

See the following in the *armasm User Guide*:

About macros.

A1120E: Bad global name
A1121E: Global name '<name>' already exists
A1122E: Locals not allowed outside macros
A1123E: Bad local name
A1125E: Unknown or wrong type of global/local symbol '<name>'
A1126E: Bad alignment boundary, must be a multiple of 2
A1127E: Bad IMPORT/EXTERN name
A1128E: Common name '<sym>' already exists
A1129E: Imported name '<sym>' already exists
A1130E: Bad exported name
A1131E: Bad symbol type for exported symbol '<sym>'
A1132E: REQUIRE directive not supported for <entity> format output
A1133E: Bad required symbol name
A1134E: Bad required symbol type, expect (symbol is either external or label) and (symbol is relocatable and absolute)
A1135E: Area name missing

AREA names starting with any non-alphabetic character must be enclosed in bars, for example change:

```
AREA 1_DataArea, CODE, READONLY
```

to:

```
AREA |1_DataArea|, CODE, READONLY
```

A1136E: Entry address already set
A1137E: Unexpected characters at end of line

This is given when extra characters that are not part of an instruction are found on an instruction line.

For example:

```
ADD r0, r0, r1 comment
```

You could change this to:

```
ADD r0, r0, r1 ; comment
```

A1138E: String "<string>" too short for operation, length must be > <oplength>
A1139E: String overflow, string exceeds <max> characters
A1140E: Bad operand type
A1141E: Relocated expressions may only be added or subtracted
A1142E: Subtractive relocations not supported for <entity> format output

This can occur when subtracting symbols that are in different areas, for example:

```
IMPORT sym1
IMPORT sym2
DCD (sym2 - sym1)
```

A1143E: COMMON directive not supported for %s format output
A1144E: DCDO directive not supported for %s format output
A1145E: Undefined exported symbol '<sym>'
A1146E: Unable to open output file <filename>: <reason>
A1147E: Bad shift name
A1148E: Unknown shift name <name>, expected one of LSL, LSR, ASR, ROR, RRX

`A1150E: Bad symbol, not defined or external`

This typically occurs in the following cases:

- When the current file requires an `INCLUDE` of another file to define some symbols, for example:

```
"init.s", line 2: Error: A1150E: Bad symbol
2 00000000 DCD EBI_CSR_0
```

  In this case, the solution is to include the required definitions file. For example:

```
INCLUDE targets/eb40.inc
```

- When the current file requires `IMPORT` for some symbols, for example:

```
"init.s", line 4: Error: A1150E: Bad symbol
4 00000000 LDR r0, =||Image$$RAM$$ZI$$Limit||
```

  In this case, the solution is to import the required symbol, for example:

```
IMPORT ||Image$$RAM$$ZI$$Limit||
```

`A1151E: Bad register name symbol`

Example:

```
MCR p14, 3, R0, Cr1, Cr2
```

The coprocessor registers `CR` must be labeled as a lowercase `c` for the code to build. The ARM register can be `r` or `R`:

```
MCR p14, 3, r0, c1, c2
```

or

```
MCR p14, 3, R0, c1, c2
```

`A1152E: Unexpected operator`

`A1153E: Undefined symbol`

`A1154E: Unexpected operand, operator expected`

`A1155E: Unexpected unary operator equal to or equivalent to <operator>`

`A1156E: Missing open bracket`

`A1157E: Syntax error following directive`

`A1158E: Illegal line start, should be blank`

Some directives, for example, `ENTRY`, `IMPORT`, `EXPORT`, and `GET` must be on a line without a label at the start of the line. This error is given if a label is present.

`A1159E: Label missing from line start`

Some directives, for example, `FUNCTION` or `SETS`, require a label at the start of the line, for example:

```
my_func FUNCTION
```

or

```
label SETS
```

This error is given if the label is missing.

`A1160E: Bad local label number`

A numeric local label is a number in the range 0-99, optionally followed by a name.

See the following in the *armasm User Guide*:

Numeric local labels.

`A1161E: Syntax error following local label definition`

`A1162E: Incorrect routine name '<name>'`

`A1163E: Unknown opcode <name> , expecting opcode or Macro`

The most common reasons for this are:

- Forgetting to put white space in the left hand margin, before the instruction. For example, change the following:

  ```
  MOV PC,LR
  ```

  to:

  ```
  MOV PC,LR
  ```

- Use of a hardware floating point instruction without using the `--fpu` switch. For example:

  ```
  FMXR FPEXC, r1 ;
  ```

  must be assembled with `armasm --fpu=vfp`

- Mis-typing the opcode:

  ```
  ADDD
  ```

  instead of:

  ```
  ADD
  ```

`A1164E: Opcode not supported on selected processor`

The processor selected on the `armasm` command line does not support this instruction.

See the following:

ARM Architecture Reference Manual.

`A1165E: Too many actual parameters, expecting <actual> parameters`
`A1166E: Syntax error following label`
`A1167E: Invalid line start`
`A1168E: Translate not allowed in pre-indexed form`
`A1169E: Missing close square bracket`
`A1170E: Immediate 0x<adr> out of range for this operation, must be below (0x<adr>)`

This error is given when a `DCB`, `DCW` or `DCWU` directive is used with an immediate that is too large. See the following in the *armasm User Guide*:

- *DCB*.
- *DCW and DCWU*.

`A1171E: Missing close bracket`
`A1172E: Bad rotator <rotator>, must be even and between 0 and 30`
`A1173E: ADR/L cannot be used on external symbols`

The ADR and ADRL pseudo-instructions can only be used with labels within the same code area. To load an out-of-area address into a register, use `LDR` instead.

`A1174E: Data transfer offset 0x<val> out of range. Permitted values are 0x<min> to 0x<max>`
`A1175E: Bad register range`
`A1176E: Branch offset 0x<val> out of range. Permitted values are 0x<min> to 0x<max>`

Branches are PC-relative, and have a limited range. If you are using numeric local labels, you can use the `ROUT` directive to limit their scope. This helps to avoid referring to the wrong label by accident.

See the following in the *armasm User Guide*:

Numeric local labels.

`A1179E: Bad hexadecimal number`
`A1180E: Missing close quote`
`A1181E: Bad operator`
`A1182E: Bad based <base> number`
`A1183E: Numeric overflow`

A1184E: Externals not valid in expressions
A1185E: Symbol missing
A1186E: Code generated in data area
An instruction has been assembled into a data area. This can happen if you have omitted the CODE attribute on the AREA directive.

See the following in the *armasm User Guide*:

AREA.

A1187E: Error in macro parameters
A1188E: Register value <val> out of range. Permitted values are <min> to <max>
A1189E: Missing '#'
A1190E: Unexpected '<entity>'
A1191E: Floating point register number out of range 0 to <maxi>
A1192E: Coprocessor register number out of range 0 to 15
A1193E: Coprocessor number out of range 0 to 15
A1194E: Bad floating-point number
A1195W: Small floating point value converted to 0.0
A1196E: Too late to ban floating point
A1198E: Unknown operand
This can occur when an operand is accidentally mistyped.

For example:

```
armasm --cpu=8-A.64 init.s -g -PD "ROM_RAM_REMAP SETL {FALS}"
```

must be:

```
armasm --cpu=8-A.64 init.s -g -PD "ROM_RAM_REMAP SETL {FALSE}"
```

See the following in the *armasm User Guide*:

Assembly time substitution of variables.

A1199E: Coprocessor operation out of range 0 to <maxi>
A1200E: Structure mismatch expect While/Wend
A1201E: Substituted line too long, maximum length <max>
A1202E: No pre-declaration of substituted symbol '<name>'
See the following in the *armasm User Guide*:

Assembly time substitution of variables.

A1203E: Illegal label parameter start in macro prototype
A1204E: Bad macro parameter default value
A1205E: Register <reg> occurs multiply in list
A1206E: Registers should be listed in increasing register number order
This warning is given if registers in, for example, LDM or STM instructions are not specified in increasing order and the --checkreglist option is used.
A1207E: Bad or unknown attribute
This error is given when an invalid attribute is given in the AREA directive. For example:

```
AREA test,CODE,READONLY,HALFWORD
```

HALFWORD is invalid, so remove it.

See the following in the *armasm User Guide*:

AREA.

A1209E: ADRL cannot be used with PC as destination
A1210E: Non-zero data within uninitialized area '<name>'
A1211E: Missing open square bracket
A1212E: Division by zero

A1213E: Attribute <entity> cannot be used with attribute <entity>
A1214E: Too late to define symbol '<sym>' as register list
A1215E: Bad register list symbol
A1216E: Bad string escape sequence
A1217E: Error writing to code file <codeFileName>: <reason>
A1219E: Bad APSR, CPSR or SPSR designator

For example:

```
MRS r0, PSR
```

It is necessary to specify which status register to use (CPSR or SPSR), such as, for example:

```
MRS r0, CPSR
```

A1220E: BLX <address> must be unconditional
A1221E: Area attribute '<entity>' not supported for <entity> object file format
A1223E: Comdat Symbol '<name>' is not defined
A1224E: <entity> format does not allow PC-relative data transfers between areas
A1225E: ASSOC attribute is not allowed in non-comdat areas
A1226E: SELECTION attribute is not allowed in non-comdat areas
A1227E: Comdat Associated area '<name>' undefined at this point in the file
A1228E: Comdat Associated area '<name>' is not an area name
A1229E: Missing COMDAT symbol
A1230E: Missing '}' after COMDAT symbol
A1234E: Undefined or Unexported Weak Alias for symbol '<sym>'
A1237E: Invalid register or register combination for this operation
A1238E: Immediate value must be word aligned when used in this operation
A1240E: Immediate value cannot be used with this operation
A1241E: Must have immediate value with this operation
A1242E: Offset must be word aligned when used with this operation
A1243E: Offset must be halfword aligned with this operation
A1244E: Missing '!'
A1245E: B or BL from Thumb code to ARM code
A1246E: B or BL from ARM code to Thumb code
A1247E: BLX from ARM code to ARM code, use BL

This occurs when there is a `BLX label` branch from A32 code to A32 code within this assembler file. This is not permitted because `BLX label` always results in a change of instruction set state. The usual solution is to use `BL` instead.

A1248E: BLX from Thumb code to Thumb code, use BL

This occurs when there is a `BLX label` branch from T32 code to T32 code within this assembler file. This is not permitted because `BLX label` always results in a change of instruction set state. The usual solution is to use `BL` instead.

A1249E: Post indexed addressing mode not available
A1250E: Pre indexed addressing mode not available for this instruction, use [Rn, Rm]
A1253E: Thumb branch to external symbol cannot be relocated: not representable in <fmt>
A1254E: Halfword literal values not supported

In the following example, change the `LDRH` into `LDR`, which is the standard way of loading constants into registers:

```
LDRH R3, =constant
```

A1256E: DATA directive can only be used in CODE areas
A1259E: Invalid PSR field specifier, syntax is <PSR>_ where <PSR> is either CPSR or SPSR
A1260E: PSR field '<entity>' specified more than once

A1261E: MRS cannot select fields, use APSR, CPSR or SPSR directly
This is caused by an attempt to use fields for CPSR or SPSR with an MRS instruction. For example:

```
MRS r0, CPSR_c
```

A1262U: Expression storage allocator failed
A1265U: Structure mismatch: IF or WHILE unmatched at end of INCLUDE file
A1267E: Bad GET or INCLUDE for file <filename>
A1268E: Unmatched conditional or macro
A1269U: unexpected GET on structure stack
A1270E: File "<entity>" not found
A1271E: Line too long, maximum line length is <MaxLineLength>
A1272E: End of input file
A1273E: '\\' should not be used to split strings
A1274W: '\\' at end of comment
A1283E: Literal pool too distant, use LTORG to assemble it within 1KB
For T32 code, a literal pool must be within 1KB of an LDR instruction that is trying to access it. See also messages A1284E and A1471W.
A1284E: Literal pool too distant, use LTORG to assemble it within 4KB
For A32 code, a literal pool must be within 4KB of an LDR instruction that is trying to access it. To solve this, add an LTORG directive into your assembly source code at a convenient place. See the following in the *armasm User Guide*:
- *Load addresses to a register using LDR Rd, =label*.
- *LTORG*.

A1285E: Bad macro name
A1286E: Macro already exists
A1287E: Illegal parameter start in macro prototype
A1288E: Illegal parameter in macro prototype
A1289E: Invalid parameter separator in macro prototype
A1290E: Macro definition too big, maximum length <max>
A1291E: Macro definitions cannot be nested
A1310W: Symbol attribute not recognized
A1311U: macro definition attempted within expansion
A1312E: Assertion failed
A1313W: Missing END directive at end of file
The assembler requires an END directive to know when the code in the file terminates. You can add comments or other such information in free format after this directive.
A1314W: Reserved instruction (using NV condition)
A1315E: NV condition not supported on targeted CPU
A1316E: Shifted register operand to MSR has undefined effect
A1319E: Undefined effect (using PC as Rs)
A1320E: Undefined effect (using PC as Rn or Rm in register specified shift)
A1321E: Undefined effect (using PC as offset register)
A1322E: Unaligned transfer of PC, destination address must be 4 byte aligned, otherwise result is UNPREDICTABLE
This error is reported when you try to use an LDR instruction to load the PC from a non word-aligned address. For example:

```
AREA Example, CODE
LDR pc, [pc, #6] ; Error – offset must be a multiple of 4
END
```

This code gives an UNPREDICTABLE result.

A1323E: Reserved instruction (Rm = Rn with post-indexing)
A1324E: Undefined effect (PC + writeback)

A1327E: Non portable instruction (LDM with writeback and base in register list, final value of base unpredictable)

In the LDM instruction, if the base register <Rn> is specified in <registers>, and base register writeback is specified, the final value of <Rn> is UNKNOWN.

A1328E: Non portable instruction (STM with writeback and base not first in register list, stored value of base unpredictable)

In the STM instruction, if <Rn> is specified in <registers> and base register writeback is specified:

- If <Rn> is the lowest-numbered register specified in <register_list>, the original value of <Rn> is stored.
- Otherwise, the stored value of <Rn> is UNKNOWN.

A1329E: Unpredictable instruction (forced user mode transfer with write-back to base)

This is caused by an instruction such as PUSH {r0}^ where the ^ indicates access to user registers. Writeback to the base register is not available with this instruction. Instead, the base register must be updated separately. For example:

```
SUB sp, sp,#4
STMID sp, {r0}^
```

Another example is replacing STMFD R0!, {r13, r14}^ with:

```
SUB r0, r0,#8
STM r0, {r13, r14}^
```

See also message A1085W.

A1331E: Unpredictable instruction (PC as source or destination)
A1332E: Unpredictable effect (PC-relative SWP)
A1334E: Undefined effect (use of PC/PSR)
A1335E: Useless instruction (PC cannot be written back)
A1337E: Useless instruction (PC is destination)
A1338E: Dubious instruction (PC used as an operand)
A1339E: Unpredictable if RdLo and RdHi are the same register
A1341E: Branch to unaligned destination, expect destination to be <max> byte aligned
A1342W: <name> of symbol in another AREA will cause link-time failure if symbol is not close enough to this instruction
A1344I: host error: out of memory
A1355U: A Label was found which was in no AREA

This can occur if no white space precedes an assembler directive. Assembler directives must be indented. For example use:

```
    IF :DEF: FOO
    ; code
    ENDIF
```

instead of:

```
IF :DEF: FOO
 ; code
ENDIF
```

Symbols beginning in the first column are assumed to be labels.

`A1356E: Instruction not supported on targeted CPU`

This occurs if you try to use an instruction that is not supported by the selected architecture or processor.

For example:

```
SMULBB r0,r0,r1 ;
```

This can be assembled with:

```
armasm --cpu 5TE
```

See the following:

ARM Architecture Reference Manual.

`A1406E: Bad decimal number`
`A1407E: Overlarge floating point value`
`A1408E: Overlarge (single precision) floating point value`
`A1409W: Small (single precision) floating value converted to 0.0`
`A1411E: Closing '>' missing from vector specifier`
`A1412E: Bad vector length, should be between <min> and <max>`
`A1413E: Bad vector stride, should be between <min> and <max>`
`A1414E: Vector wraps round over itself, length * stride should not be greater than <max>`
`A1415E: VFPASSERT must be followed by 'VECTOR' or 'SCALAR'`
`A1416E: Vector length does not match current vector length <len>`
`A1417E: Vector stride does not match current vector stride`
`A1418E: Register has incorrect type '<type>' for instruction, expect floating point/double register type`
`A1419E: Scalar operand not in a scalar bank`
`A1420E: Lengths of vector operands are different`
`A1421E: Strides of vector operands are different`
`A1422E: This combination of vector and scalar operands is not allowed`
`A1423E: This operation is not vectorizable`
`A1424E: Vector specifiers not allowed in operands to this instruction`
`A1425E: Destination vector must not be in a scalar bank`
`A1426E: Source vector must not be in a scalar bank`
`A1427E: Operands have a partial overlap`
`A1428E: Register list contains registers of varying types`
`A1429E: Expected register list`

The assembler reports this when `FRAME SAVE` and `FRAME RESTORE` directives are not given register lists.

See the following in the *armasm User Guide*:

- *FRAME RESTORE*.
- *FRAME SAVE*.

`A1430E: Unknown frame directive`
`A1431E: Frame directives are not accepted outside of PROCs/FUNCTIONs`

See the following in the *armasm User Guide*:

Frame directives.

`A1432E: Floating-point register type not consistent with selected floating-point architecture`
`A1433E: Only the writeback form of this instruction exists`

The addressing mode specified for the instruction did not include the writeback specifier (that is, a `'!'` after the base register), but the instruction set only supports the writeback form of the instruction. Either use the writeback form, or replace with instructions that have the required behavior.

`A1434E: Architecture attributes '<attr1>' and '<attr2>' conflict`

`A1435E: {PCSTOREOFFSET} is not defined when assembling for an architecture`
`{PCSTOREOFFSET}` is only defined when assembling for a processor, not for an architecture.

`A1437E: {ARCHITECTURE} is undefined`
`{ARCHITECTURE}` is only defined when assembling for an architecture, not for a processor.

`A1446E: Bad or unknown attribute '<attr>'. Use --apcs /interwork instead`
For example:

```
AREA test1, CODE, READONLY

AREA test, CODE, READONLY, INTERWORK
```

This code might have originally been intended to work with the legacy ARM Software Development Toolkit (SDT). The `INTERWORK` area attribute is obsolete. To eliminate the error, do the following:

- remove the `", INTERWORK"` from the `AREA` line.
- assemble with `armasm --apcs /interwork foo.s` instead.

`A1447W: Missing END directive at end of file, but found a label named END`
This is caused by the `END` directive not being indented.

`A1448E: Deprecated form of PSR field specifier used (use _f)`

`A1449E: Deprecated form of PSR field specifier used (use _c)`

`A1450E: Deprecated form of PSR field specifier used (use _cxsf for future compatibility)`
`armasm` supports the full range of `MRS` and `MSR` instructions, in the following forms:

```
MRS(cond) Rd, CPSR
MRS(cond) Rd, SPSR
MSR(cond) CPSR_fields, Rm
MSR(cond) SPSR_fields, Rm
MSR(cond) CPSR_fields, #immediate
MSR(cond) SPSR_fields, #immediate
```

where `fields` can be any combination of `cxsf`.

Legacy versions of the assembler permitted other forms of the `MSR` instruction to modify the control field and flags field:

- `cpsr` or `cpsr_all`, control and flags field
- `cpsr_flg`, flags field only
- `cpsr_ctl`, control field only.

Similar control and flag settings apply for SPSR.

These forms are deprecated and must not be used. If your legacy code contains them, the assembler reports:

```
Deprecated form of PSR field specifier used (use _cxsf)
```

To avoid the warning, in most cases you can modify your code to use `_c`, `_f`, `_cf` or `_cxsf` instead.

See the following in the *armasm User Guide*:

- *Conditional execution in A32 state*.
- *Conditional execution in T32 state*.
- *General-purpose registers in AArch32 state*.
- *Access to the inline barrel shifter in AArch32 state*.

See the following FAQ:

armasm: use of MRS and MSR instructions ('Deprecated form of PSR field specifier').

`A1454E: FRAME STATE RESTORE directive without a corresponding FRAME STATE REMEMBER`
See the following in the *armasm User Guide*:

- *Frame directives*.
- *FRAME STATE REMEMBER*.
- *FRAME STATE RESTORE*.

`A1456W: INTERWORK area directive is obsolete. Continuing as if --apcs /inter selected`
For example, the following code generates this warning:

```
AREA test, CODE, READONLY, INTERWORK
```

This code might have originally been intended to work with the legacy ARM Software Development Toolkit (SDT). The `INTERWORK` area attribute is obsolete. To eliminate the warning, do the following:

1. Remove the `", INTERWORK"` from the `AREA` line.
2. Assemble with `armasm --apcs /interwork foo.s` instead.

See also message A1446E.

`A1457E: Cannot mix INTERWORK and NOINTERWORK code areas in same file`
`INTERWORK` and (default) `NOINTERWORK` code areas cannot be mixed in the same file. This code might have originally been intended to work with the ARM Software Development Toolkit (SDT). The `INTERWORK AREA` attribute is obsolete in the ARM Compiler toolchain.

For example:

```
AREA test1, CODE, READONLY
  …
AREA test2, CODE, READONLY, INTERWORK
```

To eliminate the error, carry out the following steps:

1. Move the two `AREA`s into separate assembly files, for example, `test1.s` and `test2.s`.
2. Remove `, INTERWORK` from the `AREA` line in `test2.s`.
3. Assemble `test1.s` with `armasm --apcs /nointerwork`.
4. Assemble `test2.s` with `armasm --apcs /interwork`.
5. At link time, the linker adds any necessary interworking veneers.

See also message A1446E.

`A1458E: DCFD or DCFDU not allowed when fpu is None`
`A1459E: Cannot B or BL to a register`
This form of the instruction is not permitted. See the following for the permitted forms:

ARM Architecture Reference Manual.

`A1461E: Specified processor or architecture does not support Thumb instructions`
It is likely that you are specifying an architecture or processor using the `--cpu` option and incorporating Thumb code in the `AREA` that is generating this error.

For example, in the following command line, the selected architecture, ARMv4, does not support Thumb code:

```
armasm --cpu 4 code.s
```

`A1462E: Specified memory attributes do not support this instruction`
`A1463E: SPACE directive too big to fit in area, area size limit 2^32`
`A1464W: ENDP/ENDFUNC without corresponding PROC/FUNC`

`A1466W: Operator precedence means that expression would evaluate differently in C`
`armasm` has always evaluated certain expressions in a different order to C. This warning might help C programmers from being caught out when writing in assembly language.
To avoid the warning, do either of the following:

- Modify the code to make the evaluation order explicit, by adding brackets.
- Suppress the warning with the `--unsafe` switch.

See the following in the *armasm User Guide*:

Operator precedence.

`A1467W: FRAME ADDRESS with negative offset <offset> is not recommended`
`A1468W: FRAME SAVE saving registers above the canonical frame address is not recommended`
`A1469E: FRAME STATE REMEMBER directive without a corresponding FRAME STATE RESTORE`
See the following in the *armasm User Guide*:

- *Frame directives*.
- *FRAME STATE REMEMBER*.
- *FRAME STATE RESTORE*.

`A1471W: Directive <directive> may be in an executable position`
This can occur with, for example, the `LTORG` directive (see messages A1283E and A1284E). `LTORG` instructs the assembler to dump literal pool `DCD` data at this position.

To prevent this warning, the data must be placed where the processor cannot execute them as instructions. A good place for an `LTORG` is immediately after an unconditional branch, or after the return instruction at the end of a subroutine.

As a last resort, you could add a branch over the `LTORG` to avoid the data being executed. For example:

```
  B unique_label
  LTORG
unique_label
```

`A1475E: At least one register must be transferred, otherwise result is UNPREDICTABLE`
`A1476E: BX r15 at non word-aligned address is UNPREDICTABLE`
`A1477E: This register combination results in UNPREDICTABLE behavior`
This error is generated when you are assembling an instruction that has UNPREDICTABLE results on execution. You must rewrite your code to avoid this UNPREDICTABLE behavior. For example, the following instructions all cause this error when assembling to Thumb, and the target architecture is ARMv6T2 or later:

```
ADD sp, r0, #100 ; error - UNPREDICTABLE use of SP
CMP pc, #1 ; error - UNPREDICTABLE use of PC
PUSH    {r0, pc} ; error - use of an UNPREDICTABLE register combination
```

`A1479W: Requested alignment <alignreq> is greater than area alignment <align>, which has been increased`

This is warning about an `ALIGN` directive that has a coarser alignment boundary than its containing `AREA`. This is not permitted. To compensate, the assembler automatically increases the alignment of the containing `AREA` for you. A simple test case that gives the warning is:

```
AREA test, CODE, ALIGN=3
ALIGN 16
mov pc, lr
END
```

In this example, the alignment of the `AREA (ALIGN=3)` is 2^3=8 byte boundary, but the `mov pc,lr` instruction is on a 16-byte boundary, causing the error.

Note

The two alignment types are specified in different ways.

This warning can also occur when using `AREA ... ALIGN=0` to align a code section on a byte boundary. This is not possible. Code sections can only be aligned on:

- a four-byte boundary for A32 code, so use `"ALIGN=2"`
- a two-byte boundary for T32 code, so use `"ALIGN=1"`.

See the following in the *armasm User Guide*:

- *ALIGN*.
- *AREA*.

`A1480W: Macro cannot have same name as a directive or instruction`

`A1481E: Object file format does not support this area alignment`

`A1482E: Shift option out of range, allowable values are from <min> to <max>`

`A1484W: Obsolete shift name 'ASL', use LSL instead`

The ARM architecture does not have an `ASL` shift operation. The ARM barrel shifter only has the following shift types:

- `ROR`.
- `ASR`.
- `LSR`.
- `LSL`.

An arithmetic (that is, signed) shift left is the same as a logical shift left, because the sign bit always gets shifted out.

If the name `ASL` is used, the assembler reports this warning and converts the `ASL` to `LSL`.

See the following in the *armasm User Guide*:

- *--unsafe*.
- *ASR*.

`A1485E: LDM/STM instruction exceeds maximum register count <max> allowed with --split_ldm`

`A1486E: ADR/ADRL of a symbol in another AREA is not supported in ELF`

The `ADR` and `ADRL` pseudo-instructions can only be used with labels within the same code section. To load an out-of-area address into a register, use `LDR` instead.

`A1487W: Obsolete instruction name 'ASL', use LSL instead`

This warning is given when the `ASL` instruction is used in pre-UAL Thumb code, that is, when you assemble using the `--16` command-line option, or you use the `CODE16` directive. See the corresponding ARM `ASL` message A1484W.

`A1488W: PROC/FUNC at line <lineno> in '<filename>' without matching ENDP/ENDFUNC`

`A1489E: <FPU> is undefined`

`A1490E: <CPU> is undefined`

{CPU} is only defined by assembling for a processor and not an architecture.

A1491W: Internal error: Found relocation at offset <offset> with incorrect alignment
This might indicate an assembler fault. Contact your supplier.

A1492E: Immediate 0x<val> out of range for this operation. Permitted values are 0x<min> to 0x<max>

A1493E: REQUIRE must be in an AREA

A1495W: Target of branch is a data address
armasm determines the type of a symbol and detects branches to data. To suppress this warning, specify --diag-suppress 1495.

A1496W: Absolute relocation of ROPI address with respect to symbol '<symbol>' at offset <offset> may cause link failure
For example, when assembling the following code with --apcs /ropi, this warning is given. This is because it generates an absolute relocation (R_ARM_ABS32) to a PI code symbol.

```
 AREA code, CODE
codeaddr DCD codeaddr
 END
```

A1497W: Absolute relocation of RWPI address with respect to symbol '<symbol>' at offset <offset> may cause link failure
For example, when assembling the following code with --apcs /rwpi, this warning is given. This is because it generates an absolute relocation (R_ARM_ABS32) to a PI data symbol.

```
 AREA data, DATA
dataaddr DCD dataaddr
 END
```

A1498E: Unexpected characters following Thumb instruction
For example, the following instruction is valid in both UAL and pre-UAL code:

```
ADD r0, r0, r1
```

However, the following instruction is invalid in pre-UAL Thumb code. The unexpected characters are , ASR #1.

```
ADD r0, r0, r1, ASR #1
```

A1499E: Register pair is not a valid contiguous pair

A1500E: Unexpected characters when expecting '<eword>'

A1501E: Shift option out of range, allowable values are 0, 8, 16 or 24

A1502W: Register <reg> is a caller-save register, not valid for this operation

A1505E: Bad expression type, expect logical expression

A1506E: Accumulator should be in form accx where x ranges from 0 to <max>

A1507E: Second parameter of register list must be greater than or equal to the first

A1508E: Structure mismatch expect Conditional

A1509E: Bad symbol type, expect label, or weak external symbol

A1510E: Immediate 0x<imm> cannot be represented by 0-255 and a rotation

A1511E: Immediate cannot be represented by combination of two data processing instructions

A1512E: Immediate 0x<val> out of range for this operation. Permitted values are <min> to <max>

A1513E: Symbol not found or incompatible Symbol type for '<name>'

A1514E: Bad global name '<name>'

A1515E: Bad local name '<name>'

A1516E: Bad symbol '<name>', not defined or external

A1517E: Unexpected operator equal to or equivalent to <operator>

A1539E: Link Order dependency '<name>' not an area

A1540E: Cannot have a link order dependency on self

A1541E: <code> is not a valid condition code

A1542E: Macro names <name1> and <name2>[parameter] conflict

A1543W: Empty macro parameter default value

A1544E: Invalid empty PSR field specifier, field must contain at least one of c,x,s,f

`A1545U: Too many sections for one <objfmt> file`

`A1546W: Stack pointer update potentially breaks 8 byte stack alignment`

The stack must be eight-byte aligned on an external boundary so pushing an odd number of registers causes this warning. For example:

```
PUSH {r0}
```

This warning is suppressed by default. To enable it, use `--diag_warning 1546`.

See the following in the *armasm User Guide*:

--diag_warning=tag{, tag}.

`A1547W: PRESERVE8 directive has automatically been set`

Example:

```
PUSH {r0,r1}
```

This warning is given because you have not explicitly set the `PRESERVE8` directive, but the assembler has set it automatically. This warning is suppressed by default. To enable it, use `--diag_warning 1547`.

See the following in the *armasm User Guide*:

- *--diag_warning=tag{, tag}*.
- *REQUIRE8 and PRESERVE8*.

`A1548W: Code contains LDRD/STRD indexed/offset from SP but REQUIRE8 is not set`

This warning is given when the `REQUIRE8` directive is not set when required. For example:

```
PRESERVE8
STRD r0,[sp,#8]
```

See the following in the *armasm User Guide*:

REQUIRE8 and PRESERVE8.

`A1549W: Setting of REQUIRE8 but not PRESERVE8 is unusual`

Example:

```
PRESERVE8 {FALSE}
REQUIRE8
STRD r0,[sp,#8]
```

`A1550U: Input and output filenames are the same`

`A1551E: Cannot add Comdef area <name> to non-comdat group`

`A1560E: Non-constant byte literal values not supported`

`A1561E: MERGE and STRING sections must be data sections`

`A1562E: Entry size for Merge section must be greater than 0`

`A1563W: Instruction stalls CPU for <stalls> cycle(s)`

The assembler can give information about possible interlocks in your code caused by the pipeline of the processor chosen by the `--cpu` option. To do this, assemble with `armasm --diag_warning 1563`.

Note

If the `--cpu` option specifies a multi-issue processor such as Cortex-A8, the interlock warnings are unreliable.

See also warning `A1746W`.

`A1572E: Operator SB_OFFSET_11_0 only allowed on LDR/STR instructions`

`A1573E: Operator SB_OFFSET_19_12 only allowed on Data Processing instructions`

`A1574E: Expected one or more flag characters from "<str>"`

`A1575E: BLX with bit[0] equal to 1 is architecturally UNDEFINED`

`A1576E: Bad coprocessor register name symbol`

A1577E: Bad coprocessor name symbol

A1578E: Bad floating point register name symbol '<sym>'

A1581W: Added <no_padbytes> bytes of padding at address <address>

By default, the assembler warns when it adds padding bytes to the generated code. This occurs whenever an instruction or directive is used at an address that requires a higher alignment, for example, to ensure A32 instructions start on a four-byte boundary after some T32 instructions, or where there is a `DCB` followed by a `DCD`.

For example:

```
 AREA Test, CODE, READONLY
 THUMB
ThumbCode
 MOVS r0, #1
 ADR r1, ARMProg
 BX r1
 ; ALIGN ; <<< uncomment to avoid the first warning
 ARM
ARMProg
 ADD r0,r0,#1
 BX LR
 DCB 0xFF
 DCD 0x1234
 END
```

This code results in the following warnings:

```
A1581W: Added 2 bytes of padding at address 0x6
8 00000008 ARM
A1581W: Added 3 bytes of padding at address 0x11
13 00000014 DCD 0x1234
```

The warning can also occur when using `ADR` in T32-only code. The `ADR` T32 pseudo-instruction can only load addresses that are word aligned, but a label within T32 code might not be word aligned. Use `ALIGN` to ensure four-byte alignment of an address within T32 code.

See the following in the *armasm User Guide*:

- *ADR (PC-relative)*.
- *ADR (register-relative)*.
- *ALIGN*.
- *DCB*.
- *DCD and DCDU*.

A1582E: Link Order area '<name>' undefined

A1583E: Group symbol '<name>' undefined

A1584E: Mode <mode> not allowed for this instruction

A1585E: Bad operand type (<typ1>) for operator <op>

A1586E: Bad operand types (<typ1>, <typ2>) for operator <op>

A1587E: Too many registers <count> in register list, maximum of <max>

A1588E: Align only available on VLD and VST instructions

A1589E: Element index must remain constant across all registers

A1590E: Mix of subscript and non-subscript elements not allowed

A1593E: Bad Alignment, must match transfer size UIMM * <dt>

A1595E: Bad Alignment, must match <st> * <dt>, or 64 when <st> is 4

A1596E: Invalid alignment <align> for dt st combination

A1597E: Register increment of 2 not allowed when dt is 8

A1598E: Bad Register list length

A1599E: Out of range subscript, must be between 0 and <max_index>

A1600E: Section type must be within range SHT_LOOS and SHT_HIUSER

A1601E: Immediate cannot be represented

A1603E: This instruction inside IT block has UNPREDICTABLE results

A1604W: Thumb Branch to destination without alignment to <max> bytes

A1606E: Symbol attribute <attr1> cannot be used with attribute <attr2>

A1607E: Thumb-2 wide branch instruction used, but offset could fit in Thumb-1 narrow branch instruction
A1608W: MOV pc,<rn> instruction used, but BX <rn> is preferred
A1609W: MOV <rd>,pc instruction does not set bit zero, so does not create a return address

This warning is caused when the current value of the PC is copied into a register while executing in Thumb state. An attempt to create a return address in this fashion fails because bit[0] is not set. Attempting to `BX` to this instruction causes a state change (to ARM).

To create a return address, you can use:

```
MOV r0, pc
ADDS r0, #1
```

This warning can then be safely suppressed with:

```
--diag_suppress 1609
```

A1611E: Register list increment of 2 not allowed for this instruction
A1612E: <type> addressing not allowed for <instr>
A1613E: Invalid register or register combination for this operation, <registers>, expected one of <expected>
A1614E: Scalar access not allowed when dt is 64
A1615E: Store of a single element or structure to all lanes is UNDEFINED
A1616E: Instruction, offset, immediate or register combination is not supported by the current instruction set

This error can be caused by attempting to use an invalid combination of operands. For example, in Thumb:

```
MOV r0, #1 ; /* Not permitted */
MOVS r0, #1 ; /* Ok */
```

See the following in the *armasm User Guide*:

A32 and T32 Instructions.

A1617E: Specified width is not supported by the current instruction set
A1618E: Specified instruction is not supported by the current instruction set
A1619E: Specified condition is not consistent with previous IT
A1620E: Error writing to file '<filename>': <reason>
A1621E: CBZ or CBNZ from Thumb code to ARM code
A1622E: Negative register offsets are not supported by the current instruction set
A1623E: Offset not supported by the current instruction set
A1624W: Branch from Thumb code to ARM code
A1625W: Branch from ARM code to Thumb code
A1626W: BL from Thumb code to ARM code
A1627W: BL from ARM code to Thumb code

This occurs when there is a branch from ARM code to Thumb code (or vice-versa) within this file. The usual solution is to move the Thumb code into a separate assembler file. Then, at link-time, the linker adds any necessary interworking veneers.

A1630E: Specified processor or architecture does not support ARM instructions

ARM M-profile processors, for example Cortex-M3 and Cortex-M1, implement only the Thumb instruction set, not the ARM instruction set. It is likely that the assembly file contains some ARM-specific instructions and is being built for one of these processors.

A1631E: Only left shifts of 1, 2 and 3 are allowed on load/stores
A1632E: Else forbidden in IT AL blocks
A1633E: LDR rx,= pseudo instruction only allowed in load word form
A1634E: LDRD/STRD has no register offset addressing mode in Thumb
A1635E: CBZ/CBNZ can not be made conditional
A1636E: Flag setting MLA is not supported in Thumb
A1637E: Error reading line: <reason>

A1638E: Writeback not allowed on register offset loads or stores in Thumb
A1639E: Conditional DCI only allowed in Thumb mode
A1640E: Offset must be a multiple of four
A1641E: Forced user-mode LDM/STM not supported in Thumb
A1642W: Relocated narrow branch is not recommended
A1643E: Cannot determine whether instruction is working on single or double precision values.
A1644E: Cannot use single precision registers with FLDMX/LSTMX
A1645W: Substituted <old> with <new>

armasm can be configured to issue a warning in cases where it chooses to substitute an instruction. For example:

- ADD *negative_number* is the same as SUB *positive_number*
- MOV *negative_number* is the same as MVN *positive_number*
- CMP *negative_number* is the same as CMN *positive_number*.

For the T32 instruction set, UNPREDICTABLE single register LDMs are transformed into LDRs.

This warning is suppressed by default, but can be enabled with `--diag_warning 1645`.

For example, when the following code is assembled with `--diag_warning 1645`:

```
AREA foo, CODE
ADD r0, #-1
MOV r0, #-1
CMP r0, #-1
```

the assembler reports:

```
Warning: A1645W: Substituted ADD with SUB
3 00000000 ADD r0, #-1
Warning: A1645W: Substituted MOV with MVN
4 00000004 MOV r0, #-1
Warning: A1645W: Substituted CMP with CMN
5 00000008 CMP r0, #-1
```

and the resulting code generated is:

```
foo
0x00000000: e2400001 ..@. SUB r0,r0,#1
0x00000004: e3e00000 .... MVN r0,#0
0x00000008: e3700001 ..p. CMN r0,#1
```

A1647E: Bad register name symbol, expected Integer register

An integer (core) register is expected at this point in the syntax.

A1648E: Bad register name symbol, expected Wireless MMX SIMD register
A1649E: Bad register name symbol, expected Wireless MMX Status/Control or General Purpose register
A1650E: Bad register name symbol, expected any Wireless MMX register
A1651E: TANDC, TEXTRC and TORC instructions with destination register other than R15 is undefined
A1652W: FLDMX/FSTMX instructions are deprecated in ARMv6. Please use FLDMD/FSTMD instructions to save and restore unknown precision values.
A1653E: Shift instruction using a status or control register is undefined
A1654E: Cannot access external symbols when loading/storing bytes or halfwords
A1655E: Instruction is UNPREDICTABLE if halfword/word/doubleword is unaligned
A1656E: Target must be at least word-aligned when used with this instruction
A1657E: Cannot load a byte/halfword literal using WLDRB/WLDRH =constant
A1658W: Support for <opt> is deprecated

An option passed to armasm is deprecated.

See the following in the *armasm User Guide*:

Assembler command-line options.

A1659E: Cannot B/BL/BLX between ARM/Thumb and Thumb-2EE

A1660E: Cannot specify scalar index on this register type
A1661E: Cannot specify alignment on this register
A1662E: Cannot specify a data type on this register type
A1663E: A data type has already been specified on this register
A1664E: Data type specifier not recognized
A1665E: Data type size must be one of 8, 16, 32 or 64
A1666E: Data type size for floating-point must be 32 or 64
A1667E: Data type size for polynomial must be 8 or 16
A1668E: Too many data types specified on instruction
A1669E: Data type specifier not allowed on this instruction
A1670E: Expected 64-bit doubleword register expression
A1671E: Expected 128-bit quadword register expression
A1672E: Expected either 64-bit or 128-bit register expression
A1673E: Both source data types must be same type and size
A1674E: Source operand 1 should have integer type and be double the size of source operand 2
A1675E: Data types and sizes for destination must be same as source
A1676E: Destination type must be integer and be double the size of source
A1677E: Destination type must be same as source, but half the size
A1678E: Destination must be untyped and same size as source
A1679E: Destination type must be same as source, but double the size
A1680E: Destination must be unsigned and half the size of signed source
A1681E: Destination must be unsigned and have same size as signed source
A1682E: Destination must be un/signed and source floating, or destination floating and source un/signed, and size of both must be 32-bits
A1683E: Data type specifiers do not match a valid encoding of this instruction
A1684E: Source operand type should be signed or unsigned with size between <min> and <max>
A1685E: Source operand type should be signed, unsigned or floating point with size between <min> and <max>
A1686E: Source operand type should be signed or floating point with size between <min> and <max>
A1687E: Source operand type should be integer or floating point with size between <min> and <max>
A1688E: Source operand type should be untyped with size between <min> and <max>
A1689E: Source operand type should be <n>-bit floating point
A1690E: Source operand type should be signed with size between <min> and <max>
A1691E: Source operand type should be integer, floating point or polynomial with size between <min> and <max>
A1692E: Source operand type should be signed, unsigned or polynomial with size between <min> and <max>
A1693E: Source operand type should be unsigned or floating point with size between <min> and <max>
A1694E: Instruction cannot be conditional in the current instruction set

Conditional instructions are not permitted in the specified instruction set. The instruction `MOVEQ`, for example, is permitted in A32 code, and in T32 code in architectures in which the `IT` instruction is available.

A1695E: Scalar index not allowed on this instruction
A1696E: Expected either 32-bit, 64-bit or 128-bit register expression
A1697E: Expected either 32-bit or 64-bit VFP register expression
A1698E: Expected 32-bit VFP register expression
A1699E: 64-bit data type cannot be used with these registers
A1700E: Source operand type should be integer with size between <min> and <max>
A1701E: 16-bit polynomial type cannot be used for source operand
A1702E: Register Dm can not be scalar for this instruction
A1704E: Register Dm must be in the range D0-D<upper> for this data type

A1705W: Assembler converted Qm register to D<rnum>[<idx>]
A1706E: Register Dm must be scalar
A1708E: 3rd operand to this instruction must be a constant expression
A1709E: Expected ARM or scalar register expression
A1710E: Difference between current and previous register should be <diff>
A1711E: Scalar registers cannot be used in register list for this instruction
A1712E: This combination of LSB and WIDTH results in UNPREDICTABLE behavior
A1713E: Invalid field specifiers for APSR: must be APSR_ followed by at least one of n, z, c, v, q or g
A1714E: Invalid combination of field specifiers for APSR
A1715E: PSR not defined on target architecture
A1716E: Destination for VMOV instruction must be ARM integer, 32-bit single-precision, 64-bit doubleword register or 64-bit doubleword scalar register
A1717E: Source register must be an ARM integer, 32-bit single-precision or 64-bit doubleword scalar register
A1718E: Source register must be an ARM integer register or same as the destination register
A1719W: This PSR name is deprecated and may be removed in a future release
A1720E: Source register must be a 64-bit doubleword scalar register
A1721E: Destination register may not have all-lanes specifier
A1722E: Labels not allowed inside IT blocks
A1723W: __RELOC is deprecated, please use the new RELOC directive
A1724E: RELOC may only be used immediately after an instruction or data generating directive
A1725W: 'armasm inputfile outputfile' form of command-line is deprecated
A1726W: Decreasing --max_cache below 8MB is not recommended
A1727W: Immediate could have been generated using the 16-bit Thumb MOVS instruction
A1728E: Source register must be same type as destination register
A1729E: Register list may only contain 32-bit single-precision or 64-bit doubleword registers
A1730E: Only IA or DB addressing modes may be used with these instructions
A1731E: Register list increment of 2 or more is not allowed for quadword registers
A1732E: Register list must contain between 1 and 4 contiguous doubleword registers
A1733E: Register list must contain 2 or 4 doubleword registers, and increment 2 is only allowed for 2 registers
A1734E: Register list must contain <n> doubleword registers with increment 1 or 2
A1735E: Post-indexed offset must equal the number of bytes loaded/stored (<n>)
A1736E: Number of registers in list must equal number of elements
A1737E: PC or SP can not be used as the offset register
A1738E: Immediate too large for this operation
A1739W: Constant generated using single VMOV instruction; second instruction is a NOP
A1740E: Number of bytes in FRAME PUSH or FRAME POP directive must not be less than zero
A1741E: Instruction cannot be conditional
A1742E: Expected LSL #Imm
A1744E: Alignment on register must be a multiple of 2 in the range 16 to 256

`A1745W: This register combination is DEPRECATED and may not work in future architecture revisions`

This warning is generated when all of the following conditions are satisfied:

- You are using a deprecated register combination, for example:

```
PUSH    {r0, pc}
```

- You are assembling for a target architecture that supports 32-bit T32 instructions, in other words ARMv6T2 or later.
- You are assembling to A32 code.

Note

- When assembling to T32, rather than A32 code, and the target architecture is ARMv6T2 or later, the assembler generates error `A1477E` instead.
- When assembling for an architecture or processor that does not support 32-bit T32 instructions, in other words ARM® architectures before ARMv6T2, by default no diagnostic is emitted.

`A1746W: Instruction stall diagnostics may be unreliable for this CPU`

This warning is shown when you enable message `A1563W` for a processor that is not modeled accurately by the assembler. It indicates that you cannot rely on the output of `A1563W` when improving your code.

See also warning `A1563W`.

`A1753E: Unrecognized memory barrier option`
`A1754E: Cannot change the type of a scalar register`
`A1755E: Scalar index has already been specified on this register`
`A1756E: Data type must be specified on all registers`
`A1757W: Symbol attributes must be within square brackets; Any other syntax is deprecated`
`A1758W: Exporting multiple symbols with this directive is deprecated`
`A1759E: Specified processor or architecture does not support Thumb-2EE instructions`
`A1760W: Build Attribute <from> is '<attr>'`
`A1761W: Difference in build attribute from '<diff>' in <from>`
`A1762E: Branch offset 0x<val> out of range of 16-bit Thumb branch, but offset encodable in 32-bit Thumb branch`

This is caused when assembling for T32 if an offset to a branch instruction is too large to fit in a 16-bit branch. The `.W` suffix can be added to the instruction to instruct the assembler to generate a 32-bit branch.

`A1763W: Inserted an IT block for this instruction`

This indicates that the assembler has inserted an IT block to permit a number of conditional instructions in T32 code. For example:

```
MOVEQ r0,r1
```

This warning is off by default. It can be enabled using `--diag_warning A1763`.

`A1764W: <name> instructions are deprecated in architecture <arch> and above`
`A1765E: Size of padding value on ALIGN must be 1, 2 or 4 bytes`

This is caused when the optional `padsize` attribute is used with an `ALIGN` directive, but has an incorrect size. It does not refer to the parameter to align to. The parameter can be any power of 2 from 2^0 to 2^31.

`A1766W: Size of padding value for code must be a minimum of <size> bytes; treating as data`
`A1767E: Unexpected characters following attribute`
`A1768E: Missing '='`
`A1769E: Bad NEON or VFP system register name symbol`
`A1771E: Bad floating-point bitpattern when expecting <exp>-bit bitpattern`

A1772E: Destination type must be signed or unsigned integer, and source type must be 32-bit or 64-bit floating-point
A1773E: Floating-point conversion only possible between 32-bit single-precision and 64-bit double-precision types
A1774E: Fixed-point conversion only possible for 16-bit or 32-bit signed or unsigned types
A1775E: Conversion between these types is not possible
A1776E: This operation is not available for 32-bit single-precision floating point types
A1777E: <n> is out of range for symbol type; value must be between <min> and <max>
A1778E: <n> is out of range for symbol binding; value must be between <min> and <max>
A1779E: DCDO cannot be used on READONLY symbol '<key>'
A1780E: Unknown ATTR directive
A1781E: Tag #<id> cannot be set by using ATTR
A1782E: Tag #<id> should be set with ATTR <cmd>
A1783E: Attribute scope must be a label or section name
A1784W: Reference to weak definition '<sym>' not relocated
A1785E: Macro '<macuse>' not found, but '<macdef>' exists
A1786W: This instruction using SP is deprecated and may not work in future architecture revisions

This warning is generated when all of the following conditions are satisfied:

- You explicitly use the SP in a deprecated way, for example:

```
ADD sp, r0, #100
```

- You are assembling for a target architecture that supports 32-bit T32 instructions, in other words ARMv6T2 or later.
- You are assembling to A32 code.

ARM deprecates the explicit use of the SP in A32 instructions in any way that is not possible in the corresponding T32 instruction. Such deprecated register uses are still possible in A32 instructions for backwards compatibility and you can suppress this warning by using the assembler's command line option `--diag_suppress=1786`. However, ARM recommends you modify your code, because it might not work in future architecture revisions.

You can replace the deprecated use of the SP shown in the example with a sequence like:

```
ADD r1, r0, #100
MOV sp, r1
```

Note

- When assembling to T32, rather than A32 code, and the target architecture is ARMv6T2 or later, the assembler generates error `A1477E` instead.
- When assembling for an architecture or processor that does not support 32-bit T32 instructions, in other words ARM architectures before ARMv6T2, by default no diagnostic is emitted.

A1787W: Use of VFP Vector Mode is deprecated in ARMv7

`A1788W: Explicit use of PC in this instruction is deprecated and may not work in future architecture revisions`

This warning is generated when all of the following conditions are satisfied:

- You explicitly use the PC in a deprecated way, for example:

```
CMP pc, #1
```

- You are assembling for a target architecture that supports 32-bit T32 instructions, in other words ARMv6T2 or later.
- You are assembling to A32 code.

ARM deprecates the explicit use of the SP in A32 instructions in any way that is not possible in the corresponding T32 instruction. Such deprecated register uses are still possible in A32 instructions for backwards compatibility and you can suppress this warning by using the assembler's command line option `--diag_suppress=1786`. However, ARM recommends you modify your code, because it might not work in future architecture revisions.

Note

- When assembling to T32, rather than A32 code, and the target architecture is ARMv6T2 or later, the assembler generates error `A1477E` instead.
- When assembling for an architecture or processor that does not support 32-bit T32 instructions, in other words ARM architectures before ARMv6T2, by default no diagnostic is emitted.

`A1789W: Explicit use of PC in this instruction is deprecated and and may not work in future architecture revisions, except as destination register`

`A1790W: Writeback ignored in Thumb LDM loading the base register`

This is caused by incorrectly adding an exclamation mark to indicate base register writeback.

For example:

```
LDM r0!, {r0-r4}
```

is not a legal instruction because r0 is the base register and is also in the destination register list. In this case, the assembler ignores the writeback and generates:

```
LDM r0, {r0-r4}
```

`A1791W: Previous value of tag #<id> will be overridden`
`A1792E: Undefined build attributes tag`
`A1793E: Conversion only possible between 16-bit and 32-bit floating point`
`A1794E: Conversion operations require two data types`
`A1795E: Source and destination vector must contain <n> elements`
`A1796E: Register type not consistent with data type`
`A1797E: Specified FPU is not compatible with CPU architecture`
`A1798W: Output is not WYSIWYG (<output>)`
`A1799W: Output has not been checked for WYSIWYG property`
`A1800W: No output for line`
`A1801E: Instruction is UNPREDICTABLE in current instruction set`
`A1803E: Bad system instruction name`
`A1804E: Bad CP14 or CP15 register name for instruction`
`A1805E: Register is Read-Only`
`A1806E: Register is Write-Only`
`A1807W: Instruction executes as NOP on target CPU`
`A1808E: Generated object file may be corrupt (<reason>)`

A1809W: Instruction aligns PC before using it; section ought to be at least 4 byte aligned

This warning is generated when all the following conditions apply:If these conditions are all met, and the code section containing this instruction is not placed at a 4-byte aligned address when linking, the instruction might operate on or with the wrong address at runtime. This is because the instruction aligns the PC to a 4-byte address before using it.

- You are using a PC-relative offset in a T32 instruction that requires the PC to be word-aligned.
- The code section containing this instruction has less than 4-byte alignment.
- The instruction is not relocated at link time (because of a relocation emitted by the assembler).

The following example shows an LDR instruction in T32 that is diagnosed by this warning because the section has an alignment of 2 bytes:

```
AREA ||.text||, CODE, READONLY, ALIGN=1
THUMB
LDR r0, [pc, #8]  ; gives warning A1809W
```

A1810E: Base register writeback value unclear; use '[rn,#n]!' or '[rn],#n' syntax
A1811E: Size of fill value must be 1, 2 or 4 bytes and a factor of fill size
A1812W: Instruction cannot be assembled in the opposite instruction set
A1813W: 32-bit instruction used where 16-bit could have been used
A1814E: No output file
A1815E: SHT_ARM_EXIDX sections require a link order dependency to be set
A1816E: Unknown opcode '<name>' in CODE16, but exists in THUMB
A1817W: ATTR tag #<id> setting ignored in <scope>
A1818W: ATTR COMPAT flag <flag> and vendor '<vendor>' setting ignored in <scope>
A1819W: ATTR compatible with tag #<id> setting ignored in <scope>
A1820E: Register and processor mode not valid for instruction
A1821E: Expected constant or register expression
A1822E: Expected list of 32-bit extension registers
A1823E: Expected list of 64-bit extension registers
A1824E: Expected core register or 32-bit, 64-bit or 128-bit extension register
A1825E: Expected constant or 32-bit extension register
A1826E: Expected constant or 64-bit extension register
A1827E: Expected constant or 128-bit extension register
A1828E: Expected core register or 32-bit extension register
A1829E: Expected core register or 64-bit extension register
A1830E: Expected core register or 128-bit extension register
A1831E: Expected constant, floating-point constant, core register or 64-bit extension register
A1832E: Expected floating-point constant, core register or 32-bit extension register
A1833E: Expected constant or '{option}', where option is a constant from 0 to 255
A1834E: Expected register or address expression
A1835E: Too few data types specified on instruction
A1836E: Expected '<dt>' data type for destination
A1837E: Expected '<dt>' data type for first source
A1838E: Unexpected characters when expecting '<word1>' or '<word2>'
A1839E: Destination register must be scalar
A1840E: First source register must be scalar
A1841E: Alignment specified on base register not valid for instruction
A1842E: Syntax not allowed for a pseudo-instruction
A1843E: Literal load not supported for instruction
A1844E: Literal type not supported
A1845E: Register type not available in current instruction set
A1846E: Invalid field specifiers for CPSR or SPSR: must be followed by at least one of c, x, s or f

A1847E: Expression requiring more than one relocation not allowed

This can occur during the assembly of A32 instructions when trying to access data in another area. For example, using:

```
LDR r0, [pc, #label - . - 8]
```

or its equivalent:

```
LDR r0, [pc, #label-{PC}-8]
```

where `label` is defined in a different `AREA`.

Change your code to use the simpler, equivalent syntax:

```
LDR r0, label
```

This works if `label` is either in the same area or in a different area.

A1848W: State change in IT block
A1849E: Scalar index on operand out of range for data type
A1850E: Width must be before any data type qualifiers
A1851E: Invalid line start - local label not supported on this directive
A1852E: Use of VFP Vector Mode is not supported on target FPU
A1853E: Alignment of instruction or target must be at least <n>-byte aligned
A1854E: Unknown opcode '<name>', maybe wrong target CPU?
A1856E: Shifted register operand not allowed
A1857E: Specified shift not allowed
A1858E: Flag setting form of this instruction not available
A1859E: Flag preserving form of this instruction not available
A1860E: Register operands must be from R0 to R7 in this instruction
A1861E: Option '<opt>' is obsolete.
A1862E: Data type size for floating-point must be 16 or 32
A1863E: Data type size for floating-point must be 32
A1864E: Data type size for floating-point must be 16, 32 or 64
A1865W: '#' not seen before constant expression
A1866E: Wireless MMX Control register not defined on target CPU
A1867E: Immediate 0x<val> out of range for this operation. Permitted values are multiples of <mult> from 0x<min> to 0x<max>
A1868E: Bitfield LSB out of range. Permitted values are 0 to <max>
A1869E: Register <field> must not be PC
A1870W: Area '<name>' has incorrect attributes, expect '<attrs>'
A1871E: Immediate 0x<imm> cannot be represented by 0-255 shifted left by 0-23 or duplicated in all, odd or even bytes

This error occurs when `armasm` cannot encode the instruction with the specified immediate.

For example, in ARM Compiler 4.1 and above, the following instruction causes this error, in Thumb state:

```
  ADDS    r0, r1, #-20
```

A workaround is to use the equivalent `SUBS` instruction instead:

```
  SUBS    r0, r1, #20
```

A1872E: Shift by register not allowed
A1873E: Only the non-writeback form of this instruction exists
A1874E: Specified register list cannot be loaded or stored in target instruction set
A1875E: Register Rn must be from R0 to R7 in this instruction

Change the specified register to be in the range R0 to R7.

A1876W: Use of '|' as a synonym for the :OR: operator is deprecated.
A1877E: Specified register for <field> not allowed in current instruction set
A1878E: Offset must be <reqalign>-byte aligned when used with this operation

A1879E: Specified addressing mode not available
A1880E: Data transfer size not available
A1881E: <mode> load/store mode is not permitted
A1882E: Destination and first source register must be same
A1883E: Destination and second source register must be same
A1884E: Specified AIF bits not available on target CPU
A1885E: Cannot change processor mode in current instruction set
A1886E: Invalid operand size for add with carry-in: must be 16 or 32
A1887E: Specified source data type not allowed; must be one of: <str>
A1888E: Specified destination data type not allowed; must be one of: <str>
A1889E: Specified register type not available on target architecture
A1890E: Specified shift results in UNPREDICTABLE behaviour
A1891E: With this register combination, writeback results in UNPREDICTABLE behaviour
A1892W: Writeback with this register combination is deprecated and may not work in future architecture revisions
A1893E: The specified flags result in UNPREDICTABLE behaviour
A1894E: The specified immediate results in UNPREDICTABLE behaviour
A1895E: The specified condition results in UNPREDICTABLE behaviour
A1896E: Specified alignment not supported on this instruction
A1897E: Bitfield width out of range. Permitted values are 1 to <max>
A1898E: Target cannot be relocated. No suitable relocation exists for this instruction
A1899E: Specified operator is only allowed on the following instructions: <instrs>
A1900W: Deprecated system instruction name
A1901E: Specified system instruction not supported on target architecture
A1902E: Specified special register not supported on target architecture
A1903E: Line not seen in first pass; cannot be assembled

This occurs if an instruction or directive does not appear in pass 1 but appears in pass 2 of the assembler.

The following example shows when a line is not seen in pass 1:

```
    AREA x,CODE
    [ :DEF: foo
num EQU 42 ; Assembler does not see this line during pass 1 because
           ; foo is not defined at this point during pass 1
    ]
foo DCD num
    END
```

A1905U: Pre-processor step failed for '<filename>'
A1906W: Unfinished IT block
A1907W: Test for this symbol has been seen and may cause failure in the second pass.

This diagnostic is suppressed by default. Enable it to identify situations that might result in errors A1903E, A1908E, or A1909E.

A1908E: Label '<name>' value inconsistent: in pass 1 it was <val1>; in pass 2 it was <val2>

The following example generates this error because in pass 1 the value of x is 0x0004+r9, and in pass 2 the value of x is 0x0000+r0:

```
    map 0, r0
    if :lnot: :def: sym
        map 0, r9
        field 4
    endif
x   field 4
sym LDR r0, x
```

A1909E: Line not seen in second pass; cannot be assembled

This occurs if an instruction or directive appears in pass 1 but does not appear in pass 2 of the assembler.

The following example shows when a line is not seen in pass 2:

```
    AREA x,CODE
    [ :LNOT: :DEF: foo
    MOV r1, r2  ; Assembler does not see this line during pass 2 because
                ; foo is already defined
    ]
foo MOV r3, r4
    END
```

A1911E: Immediate 0x<val> out of range for this operation. Immediate value must be 0
A1912E: In this instruction, register <field> must not be PC when flag-setting is specified
A1913E: Specified operand type not allowed in this position
A1914E: Expected expression
A1915W: Relocation is not recommended on this instruction or directive
A1916E: Unknown built-in variable '<name>'
A1917E: Expected vector register expression
A1921E: Expected 8-bit byte register expression
A1922E: Expected 16-bit halfword register expression
A1923E: Expected list of vector registers
A1925E: Coprocessor number must be 14 or 15 on this architecture
A1927E: Expected core register, 64-bit extension register or vector register
A1931W: This instruction inside IT block is deprecated
A1932E: Register type not allowed on this architecture
A1933E: Option '<opt>' not supported on target architecture
A1934E: Shift by <shift> not allowed. Permitted values are <allowed>
A1936E: Literal pool too distant, use LTORG to assemble it within <distance>
A1937E: Conversions to fixed point only support rounding mode Z
A1938E: Coprocessor number must be 14 on this architecture
A1939W: This mnemonic is deprecated
A1940E: Execute only is not compatible with <option>
A1941E: Unable to align to a non-multiple of <nopsize> using NOP instructions
A1942E: Data declarations are not permitted in execute-only sections
A1943E: INCBIN cannot be used in execute-only sections
A1944E: Literal pool entries cannot be generated in execute-only sections
A1992E: MOVT of external symbol must follow corresponding MOVW instruction
A1993E: This operator requires a relocation that is not supported in <objfmt>
A1994E: This directive is not supported in <objfmt>
A1995E: Weak definitions are not supported in <objfmt>
A1996E: TYPE must only be used after WEAK on IMPORT
A1997E: Expected alias for weak extern symbol
A1998E: Comdat Associated area must have Comdat Associative selection type
A1999E: Comdat Associated area cannot be another Comdat Associated area
A9511E: Product definition file was not found

`armasm` cannot find the required product license mapping (`.elmap`) files.

This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- `armasm` is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- `armasm` is looking for a non-existent `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 2
Linker Errors and Warnings

Describes the error and warning messages for the linker, `armlink`.

It contains the following sections:

- *2.1 Suppressing armlink error and warning messages* on page 2-38.
- *2.2 List of the armlink error and warning messages* on page 2-39.

2.1 Suppressing armlink error and warning messages

You can use command-line options to suppress or downgrade some of the diagnostic messages that the linker produces.

All linker warnings are suppressible with `--diag_suppress`, in the same way as compiler warnings. For example:

```
--diag_suppress 6306
```

Some errors such as L6220E, L6238E and L6784E can be downgraded to a warning by using:

```
--diag_warning
```

2.2 List of the armlink error and warning messages

A list of the error and warning messages that `armlink` produces.

`L6000U: Out of memory.`
This error is reported by RVCT v4.0 and earlier. For more details on why you might see this error and possible solutions, see the description for error `L6815U`.

`L6001U: Could not read from file <filename>.`

`L6002U: Could not open file <filename>: <reason>`
This indicates that the linker was unable to open a file specified on the linker command line. This can indicate a problem accessing the file or a fault with the command line. Some common examples of this message are:

- `L6002U: Could not open file /armlib/{libname}: No such file or directory`

  Specify the library path with `--libpath`.
- `Error : armlink : L6002: Could not open file errors=ver.txt`

  This is caused by a missing double-dash (`--`) in front of `errors=ver.txt`. If you do not prefix options with `--` or `-`, the linker treats them as input files and fails the link step because it is unable to load all the specified files.

See the following in the *armlink User Guide*:

--libpath=pathlist.

See the following in the *Getting Started Guide*:

Toolchain environment variables.

`L6003U: Could not write to file <filename>.`
An file I/O error occurred while reading, opening, or writing to the specified file.

`L6004U: Incomplete library member list <list> for <library>.`
This can occur if there is whitespace in the list of library objects.

The following example fails with `Fatal error: L6004U: Missing library member in member list for x.lib`:

```
armlink x.lib(foo.o, bar.o)
```

The following example succeeds:

```
armlink x.lib(foo.o,bar.o)
```

Another less common cause is a corrupt library, or possibly a library in an unsupported format.

`L6005U: Extra characters on end of member list for <library>.`

`L6006U: Overalignment value not specified with OVERALIGN attribute for execution region <regionname>.`
See the following in the *armlink User Guide*:

`L6007U: Could not recognize the format of file <filename>.`
The linker can recognize object files in ELF format and library files in AR format. The specified file is either corrupt, or is in a file format that the linker cannot recognize.

`L6008U: Could not recognize the format of member <mem> from <lib>.`
The linker can recognize library member objects in the ELF file format. The specified library member is either corrupt, or is in a file format that the linker cannot recognize.

`L6009U: File <filename> : Endianness mismatch.`
The endianness of the specified file or object did not match the endianness of the other input files. The linker can handle input of either big endian or little endian objects in a single link step, but not a mixed input of some big and some little endian objects.

`L6010U: Could not reopen stderr to file <filename>: <reason>`
An file I/O error occurred while reading, opening, or writing to the specified file.

`L6011U: Invalid integer constant : <number>.`
Specifying an illegal integer constant causes this. An integer can be entered in hexadecimal format by prefixing `&`, `0x`, or `0X`.

`L6015U: Could not find any input files to link.`
The linker must be provided with at least one object file to link.

For example, if you try to link with:

```
armlink lib.a -o foo.axf
```

the linker reports this error.

You must instead use, for example:

```
armlink foo_1.o foo_2.o lib.a -o foo.axf
```

`L6016U: Symbol table missing/corrupt in object/library <object>.`
This can occur when linking with libraries built with the GNU tools. This is because GNU `ar` can generate incompatible information.

The workaround is to replace `ar` with `armar` and use the same command-line arguments. Alternatively, the error is recoverable by using `armar -s` to rebuild the symbol table.

`L6017U: Library <library> symbol table contains an invalid entry, no member at offset 0x<offset>.`
The library might be corrupted. Try rebuilding it.

`L6018U: <filename> is not a valid ELF file.`

`L6019U: <filename> is not a valid 64 bit ELF file.`

`L6020U: <filename> is not a valid 32 bit ELF file.`

`L6022U: Object <objname> has multiple <table>.`
The object file is faulty or corrupted. This might indicate a compiler fault. Contact your supplier.

`L6024U: Library <library> contains an invalid member name.`
The file specified is not a valid library file, is faulty or corrupted. Try rebuilding it.

`L6025U: Cannot extract members from a non-library file <library>.`
The file specified is not a valid library file, is faulty or corrupted. Try rebuilding it.

`L6026U: ELF file <filename> has neither little or big endian encoding`
The ELF file is invalid. Try rebuilding it.

`L6027U: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) has invalid/unknown type.`
This might indicate a compiler fault. Contact your supplier.

`L6028U: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) has invalid offset.`
This might indicate a compiler fault. Contact your supplier.

`L6029U: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is wrt invalid/missing symbol.`
The relocation is with respect to a symbol that is either:

- invalid or missing from the object symbol table
- a symbol that is not suited to be used by a relocation.

This might indicate a compiler fault. Contact your supplier.

`L6030U: Overalignment <overalignment> for region <regname> must be at least 4 and a power of 2`
See the following in the *armlink User Guide*:

- *Execution region attributes*.
- *Syntax of an input section description*.
- *Overalignment of execution regions and input sections*.

`L6031U: Could not open scatter description file <filename>: <reason>`
An I/O error occurred while trying to open the specified file. This could be because of an invalid filename.

`L6032U: Invalid <text> <value> (maximum <max_value>) found in <object>`

`L6033U: Symbol <symbolname> in <objname> is defined relative to an invalid section.`

`L6034U: Symbol <symbolname> in <objname> has invalid value.`

This is most often caused by a section-relative symbol having a value that exceeds the section boundaries.

`L6035U: Relocation #<rel_class>:<rel_number> in ZI Section <objname>(<secname>) has invalid type.`

ZI Sections cannot have relocations other than of type R_ARM_NONE.

`L6036U: Could not close file <filename>: <reason>`

An I/O error occurred while closing the specified file.

`L6037U: '<arg>' is not a valid argument for option '<option>'.`

The argument is not valid for this option. This could be because of a spelling error, or because of the use of an unsupported abbreviation of an argument.

`L6038U: Could not create a temporary file to write updated SYMDEFS.`

An I/O error occurred while creating the temporary file required for storing the SYMDEFS output.

`L6039W: Relocation from #<rel_class>:<rel_number> in <objname>(<secname>) with respect to <symname>. Skipping creation of R-type relocation. No corresponding R-type relocation exists for type <rel_type>.`

`--reloc` is used with objects containing relocations that do not have a corresponding R-type relocation.

`L6041U: An internal error has occurred (<clue>).`

Contact your supplier.

`L6042U: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is wrt a mapping symbol(#<idx>, Last Map Symbol = #<last>).`

Relocations with respect to mapping symbols are not permitted. This might indicate a compiler fault. Contact your supplier.

`L6043U: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is with respect to an out of range symbol(#<val>, Range = 1-<max>).`

Relocations can only be made wrt symbols in the range (1-n), where n is the number of symbols.

`L6047U: The size of this image (<actual_size> bytes) exceeds the maximum allowed for this version of the linker`

`L6048U: The linker is unable to continue the link step (<id>). This version of the linker will not create this image.`

`L6049U: The linker is unable to continue the link step (<id>). This version of the linker will not link with one or more given libraries.`

`L6050U: The code size of this image (<actual_size> bytes) exceeds the maximum allowed for this version of the linker.`

`L6058E: Syntax error parsing linker script <script> at line <lineno> : <token>`

The link `ld` script has a syntax error at the line number.

`L6064E: ELF Executable file <filename> given as input on command line`

This might be because you specified an object file as output from from the compiler without specifying the `-c` compiler option. For example:

```
armclang --target aarch64-arm-none-eabi file.c -o file.o

armlink file.o -o file.axf
```

`L6065E: Load region <name> (size <size>) is larger than maximum writable contiguous block size of 0x80000000.`

The linker attempted to write a segment larger than 2GB. The size of a segment is limited to 2GB.

`L6175E: EMPTY region <regname> cannot have any section selectors.`

`L6176E: A negative max_size cannot be used for region <regname> without the EMPTY attribute.`

Only regions with the `EMPTY` attribute are permitted to have a negative `max_size`.

`L6177E: A negative max_size cannot be used for region <regname> which uses the +offset form of base address.`

Regions using the `+offset` form of base address are not permitted to have a negative max-size.

`L6188E: Special section <sec1> multiply defined by <obj1> and <obj2>.`

A special section is one that can only be used once, such as `"Veneer$$Code"`.

`L6195E: Cannot specify both '<attr1>' and '<attr2>' for region <regname>`

See the following in the *armlink User Guide*:

- *Load region attributes*.
- *Execution region attributes*.
- *Address attributes for load and execution regions*.
- *Inheritance rules for load region address attributes*.
- *Inheritance rules for execution region address attributes*.
- *Inheritance rules for the RELOC address attribute*.

`L6200E: Symbol <symbolname> multiply defined by <object1> and <object2>.`

A common example of this:

```
Symbol __stdout multiply defined (by retarget.o and stdio.o).
```

This means that there are two conflicting definitions of `__stdout` present in `retarget.o` and `stdio.o`. The one in `retarget.o` is your own definition. The one in `stdio.o` is the default implementation, which was probably linked-in inadvertently.

`stdio.o` contains a number of symbol definitions and implementations of file functions like `fopen`, `fclose`, and `fflush`.

`stdio.o` is being linked-in because it satisfies some unresolved references.

To identify why `stdio.o` is being linked-in, you must use the `--verbose` link option switch. For example:

```
armlink [... your normal options...] --verbose --list err.txt
```

Then study `err.txt` to see exactly what the linker is linking in, from where, and why.

You might have to either:

- eliminate the calls like `fopen`, `fclose`, and `fflush`
- re-implement the `_sys_xxxx` family of functions.

See the following in the *ARM C and C++ Libraries and Floating-Point Support User Guide*:

Tailoring input/output functions in the C and C++ libraries.

`L6201E: Object <objname> contains multiple entry sections.`

The input object specifies more than one entry point. Use the `--entry` command-line option to select the entry point to use.

See the following in the *armlink User Guide*:

--entry=location.

`L6202E: <objname>(<secname>) cannot be assigned to non-root region '<regionname>'`
A root region is a region that has an execution address the same as its load address. The region does not therefore require moving or copying by the scatter-load initialization code.
Certain sections must be placed in a root region in the image, including:

- `__main.o`.
- The linker-generated table (`Region$$Table`).
- Scatter-loading (`__scatter*.o`) objects from the library.
- Decompressor (`__dc*.o`) objects from the library.

If a required section is not placed in a root region, the linker reports, for example:

```
anon$$obj.o(Region$$Table) cannot be assigned to a non-root region 'RAM'.
```

You can use `InRoot$$Sections` to include all required sections in a root region:

```
ROM_LOAD 0x0000 0x4000
{
  ROM_EXEC 0x0000 0x4000 ; root region
  {
    vectors.o (Vect, +FIRST) ; Vector table
    * (InRoot$$Sections) ; All library sections
                         ; that must be in a root region
                         ; for example, __main.o, __scatter*.o,
                         ; dc*.o and * Region$$Table
  }
  RAM 0x10000 0x8000
  {
    * (+RO, +RW, +ZI) ; all other sections
  }
}
```

`L6203E: Entry point (<address>) lies within non-root region <regionname>.`
The image entry point must correspond to a valid instruction in a root-region of the image.

`L6204E: Entry point (<address>) does not point to an instruction.`
The image entry point you specified with the `--entry` command-line option must correspond to a valid instruction in the root-region of the image.

See the following in the *armlink User Guide*:

--entry=location.

`L6205E: Entry point (<address>) must be word aligned for ARM instructions.`
This message is displayed because the image entry point you specified with the `--entry` command-line option is not word-aligned. For example, you specified `--entry=0x8001` instead of `--entry=0x8000`.

See the following in the *armlink User Guide*:

--entry=location.

`L6206E: Entry point (<address>) lies outside the image.`
The image entry point you specified with the `--entry` command-line option is outside the image. For example, you might have specified an entry address of `0x80000` instead of `0x8000`, as follows:

```
armlink --entry=0x80000 test.o -o test.axf
```

See the following in the *armlink User Guide*:

--entry=location.

`L6208E: Invalid argument for --entry command: '<arg>'`
See the following in the *armlink User Guide*:

--entry=location.

`L6209E: Invalid offset constant specified for --entry (<arg>)`
See the following in the *armlink User Guide*:

--entry=location.

`L6210E: Image cannot have multiple entry points. (<address1>,<address2>)`
One or more input objects specifies more than one entry point for the image. Use the `--entry` command-line option to select the entry point to use.

See the following in the *armlink User Guide*:

--entry=location.

`L6211E: Ambiguous section selection. Object <objname> contains more than one section.`
This can occur when using the linker option `--keep` on an assembler object that contains more than one `AREA`. The linker must know which `AREA` you want to keep.
To solve this, use more than one `--keep` option to specify the names of the AREAs to keep, such as:

```
--keep boot.o(vectors) --keep boot.o(resethandler) …
```

Note

Using assembler files with more than one `AREA` might give other problems elsewhere, so this is best avoided.

`L6213E: Multiple First section <object2>(<section2>) not allowed. <object1>(<section1>) already exists.`
Only one `FIRST` section is permitted.

`L6214E: Multiple Last section <object2>(<section2>) not allowed. <object1>(<section1>) already exists.`
Only one `LAST` section is permitted.

`L6215E: Ambiguous symbol selection for --First/--Last. Symbol <symbol> has more than one definition.`
See the following in the *armlink User Guide*:

`L6216E: Cannot use base/limit symbols for non-contiguous section <secname>`

The exception handling index tables generated by the compiler are given the section name `.ARM.exidx`. For more information, see *Exception Handling ABI for the ARM Architecture*.

At link time these tables must be placed in the same execution region and be contiguous. If you explicitly place these sections non-contiguously using specific selector patterns in your scatter file, then this error message is likely to occur. For example:

```
LOAD_ROM 0x00000000
{
  ER1 0x00000000
  {
    file1.o (+RO) ; from a C++ source
    * (+RO)
  }
  ER2 0x01000000
  {
    file2.o (+RO) ; from a C++ source
  }
  ER3 +0
  {
    * (+RW, +ZI)
  }
}
```

This might produce the following error if exception handling index tables are in both `file1.o` and `file2.o`, because the linker cannot place them in separate regions:

```
Error: L6216E: Cannot use base/limit symbols for non-contiguous section .ARM.exidx
```

Also, the `.init_array` sections must be placed contiguously within the same region for their base and limit symbols to be accessible.

The correct code is:

```
LOAD_ROM 0x00000000
{
  ER1 0x00000000
  {
    file1.o (+RO)   ; from a C++ source
    * (.ARM.exidx)  ; Section .ARM.exidx must be placed
                    ; explicitly, otherwise it is shared between
                    ; two regions and the linker is unable to
                    ; decide where to place it.
    *(.init_array)  ; Section .init_array must be placed
                    ; explicitly, otherwise it is shared between
                    ; two regions and the linker is unable to
                    ; decide where to place it.
    * (+RO)
  }
  ER2 0x01000000
  {
    file2.o (+RO)   ; from a C++ source
  }
  ER3 +0
  {
    * (+RW, +ZI)
  }
}
```

In this example, the base and limit symbols are contained in `.init_array` in a single region.

See the following in the *ARM C and C++ Libraries and Floating-Point Support User Guide*:

C++ initialization, construction and destruction.

`L6217E: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) with respect to <symbol>. R_ARM_SBREL32 relocation to imported symbol`

`L6218E: Undefined symbol <symbol> (referred from <objname>).`

Some common examples of this are:

- User Error. There is a reference to an undefined or incorrectly defined symbol.
- `Undefined symbol __ARM_switch8 or __ARM_ll_<xxxx> functions`

  The helper functions are automatically generated into the object file by the compiler.

  Note

  An undefined reference error can, however, still be generated if linking objects from legacy projects where the helper functions are in the `h_xxx` libraries (`h` indicates that these are compiler helper libraries, rather than standard C library code).

  Re-compile the object or ensure that these libraries can be found by the linker.

- When attempting to refer to a function or entity in C from a function or entity in C++. This is caused by C++ name mangling, and can be avoided by marking C functions `extern "C"`.
- `Undefined symbol thunk{v:0,-44} to Foo_i::~Foo_i() (referred from Bar_i.o)`

  The symbol `thunk{v:0,-44} to Foo_i::~Foo_i()` is a wrapper function round the regular `Foo_i::~Foo_i()`.
  `Foo_i` is a derived class of some other base class, therefore:
  — it has a base-class vtable for when it is referred to by a pointer to that base class
  — the base-class vtable has an entry for the thunk
  — the destructor thunk is output when the actual (derived class) destructor is output.

  Therefore, to avoid the error, ensure this destructor is defined.
- `Undefined symbol main (referred from kernel.o)`

  The linker is reporting that your application does not include a `main()` function.

`L6219E: <type> section <object1>(<section1>) attributes {<attributes>} incompatible with neighboring section <object2>(<section2>).`

This error occurs when the default ordering rules used by the linker (RO followed by RW followed by ZI) are violated. This typically happens when one uses `+FIRST` or `+LAST`, for example in a scatter file, attempting to force RW before RO.

`L6220E: <type> region <regionname> size (<size> bytes) exceeds limit (<limit> bytes).`

Example:

```
Execution region ROM_EXEC size (4208184 bytes) exceeds limit (4194304 bytes).
```

This can occur where a region has been given an (optional) maximum length in the scatter file, but the size of the code and data being placed in that region has exceeded the limit. This error is suppressible with `--diag_suppress 6220`.

For example, this might occur when using `.ANY`*num* selectors with the `ALIGN` directive in a scatter file to force the linker to insert padding. You might be able to fix this using the `--any_contingency` option.

See the following in the *armlink User Guide*:

- *Placement of unassigned sections with the .ANY module selector*.
- *--any_contingency*.
- *--diag_suppress=tag[,tag,...]*.

`L6221E: <type1> region <regionname1> with <addrtype1> range [<base1>,<limit1>) overlaps with <type2> region <regionname2> with <addrtype2> range [<base2>,<limit2>).`

This error can occur even though information in the scatter-loading description file and map information generated by the linker indicate that the execution regions do not overlap.

In RVCT v4.0 and earlier, the linker did not provide as much information, making the message harder to understand:

`Warning L6221E: <type1> region <regionname1> overlaps with <type1> region <regionname2>`

Example `test.s` file:

```
    AREA area1, CODE
    BX lr

    AREA area2, READWRITE, NOINIT
    SPACE 10

    AREA area3, READWRITE
    DCD 10
    END
```

Example `scatter.txt` file:

```
LR1 0x8000
{
  ER1 +0
  {
    *(+ro)
  }
  ER2 +0
  {
    *(+zi)
  }
  ER3 +0
  {
    *(+rw)
  }
}
```

Built with:

```
armasm test.s
armlink -o test.axf --scatter scatter.txt test.o
```

Generates:

`Warning: L6221E: Execution region ER2 with Execution range [0x00008004,0x00008010) overlaps with Execution region ER3 with Load range [0x00008004,0x00008008).`

The linker might emit warning message `L6221E` when an execution region base address overlaps with the load address of another region. This could be due to an incorrect scatter file. The memory map of the image has a load view and an execution view, described by the scatter-loading file. A non-ZI section must have a unique load address and in most cases must have a unique execution address. From RVCT v3.1 onwards, the linker no longer assigns space to ZI execution regions. Therefore this warning might be because a load region `LR2` with a relative base address immediately follows a ZI execution region in a load region `LR1`.

Because the overlapping part might not have real code or data inside, the warning might be harmless.

From RVCT v4.0 build 821 onwards, you can use the following linker options to find out the addresses of each region, and any regions that overlap with a load region:

`--load_addr_map_info --map --list=map.txt`

You can do one of the following:

- Ignore the warning, only if after analysis it is possible to determine that the execution region is not going to corrupt the load region that has not yet been copied to its execution region address. Also, debug the application to confirm that it initializes and executes correctly.
- Adjust the base addresses of the execution regions.
- Use the `FIXED` scatter-loading attribute to make the load regions and execution regions have the same base addresses.

See the following in the *armlink User Guide*:

- *Scatter files containing relative base address load regions and a ZI execution region*.
- *Execution region attributes*.
- *Root execution regions and the FIXED attribute*.

`L6222E: Partial object cannot have multiple ENTRY sections, <e_oname>(<e_sname>) and <oname>(<sname>).`

Where objects are being linked together into a partially-linked object, only one of the sections in the objects can have an entry point.

Note

It is not possible in this case to use the linker option `--entry` to select one of the entry points.

`L6223E: Ambiguous selectors found for <objname>(<secname>) from Exec regions <region1> and <region2>.`

This occurs if the scatter file specifies `<objname>(<secname>)` to be placed in more than one execution region. This can occur accidentally when using wildcards (`*`). The solution is to make the selections more specific in the scatter file.

`L6224E: Could not place <objname>(<secname>) in any Execution region.`

This occurs if the linker cannot match an input section to any of the selectors in your scatter file. You must correct your scatter file by adding an appropriate selector.

See the following in the *armlink User Guide*:

Section placement with the linker.

`L6225E: Number <str...> is too long.`
`L6226E: Missing base address for region <regname>.`
`L6227E: Using --reloc with --rw-base without --split is not allowed.`

See the following in the *armlink User Guide*:

- *--reloc*.
- *--rw_base=address*.
- *--split*.

`L6228E: Expected '<str1>', found '<str2>'.`
`L6229E: Scatter description <file> is empty.`
`L6230E: Multiple execution regions (<region1>,<region2>) cannot select <secname>.`
`L6231E: Missing module selector.`
`L6232E: Missing section selector.`
`L6233E: Unknown section selector '+<selector>'.`
`L6234E: <ss> must follow a single selector.`

For example, in a scatter file:

```
:
* (+FIRST, +RO)
:
```

`+FIRST` means place this (single) section first. Selectors that can match multiple sections (for example, `+RO` or `+ENTRY`) are not permitted to be used with `+FIRST` (or `+LAST`). If used together, the error message is generated.

`L6235E: More than one section matches selector - cannot all be FIRST/LAST.`

See the following in the *armlink User Guide*:

- *Section placement with the FIRST and LAST attributes*.
- *Syntax of an input section description*.

`L6236E: No section matches selector - no section to be FIRST/LAST.`

The scatter file specifies a section to be `+FIRST` or `+LAST`, but that section does not exist, or has been removed by the linker because it believes it to be unused. Use the linker option `--info unused` to reveal which objects are removed from your project. Example:

```
ROM_LOAD 0x00000000 0x4000
{
 ROM_EXEC 0x00000000
 {
   vectors.o (Vect, +First) << error here
   * (+RO)
 }
 RAM_EXEC 0x40000000
 {
   * (+RW, +ZI)
 }
}
```

Some possible solutions are:

- Ensure `vectors.o` is specified on the linker command line.
- Link with `--keep vectors.o` to force the linker not to remove this, or switch off this optimization entirely, with `--no_remove`. ARM does not recommend this.
- ARM recommends that you add the `ENTRY` directive to `vectors.s`, to tell the linker that it is a possible entry point for your application. For example:

  ```
  AREA Vect, CODE
  ENTRY ; define this as an entry point
  Vector_table
  ...
  ```

  Then link with `--entry Vector_table` to define the real start of your code.

See the following in the *armlink User Guide*:

- *Section placement with the FIRST and LAST attributes*.
- *--entry=location*.
- *--info=topic[,topic,...]*.
- *--keep=section_id*.
- *--remove, --no_remove*.
- *Syntax of an input section description*.

See the following in the *armasm User Guide*:

ENTRY.

`L6237E: <objname>(<secname>) contains relocation(s) to unaligned data.`

`L6238E: <objname>(<secname>) contains invalid call from '<attr1>' function to '<attr2>' function <sym>.`

This linker error is given where a stack alignment conflict is detected in object code. The *ABI for the ARM Architecture* suggests that code maintains eight-byte stack alignment at its interfaces. This permits efficient use of `LDRD` and `STRD` instructions to access eight-byte aligned `double` and `long long` data types.

Symbols such as `~PRES8` and `REQ8` are Build Attributes of the objects:

- `PRES8` means the object preserves eight-byte alignment of the stack
- `~PRES8` means the object does not preserve eight-byte alignment of the stack (~ meaning NOT)
- `REQ8` means the object requires eight-byte alignment of the stack.

This link error typically occurs in two cases:

- Where assembler code (that does not preserve eight-byte stack alignment) calls compiled C/C++ code (that requires eight-byte stack alignment).
- Where attempting to link legacy objects that were compiled with older tools with objects compiled with recent tools. Legacy objects that do not have these attributes are treated as `~PRES8`, even if they do actually happen to preserve eight-byte alignment.

For example:

```
Error: L6238E: foo.o(.text) contains invalid call from '~PRES8' function to 'REQ8'
function foobar
```

This means that there is a function in the object `foo.o` (in the section named `.text`) that does not preserve eight-byte stack alignment, but which is trying to call function `foobar` that requires eight-byte stack alignment.

A similar warning that might be encountered is:

```
Warning: L6306W: '~PRES8' section foo.o(.text) should not use the address of 'REQ8'
function foobar
```

where the address of an external symbol is being referred to.

There are two possible solutions to work around this issue:

- Rebuild all your objects/libraries.

  If you have any assembler files, you must check that all instructions preserve eight-byte stack alignment, and if necessary, correct them.

  For example, change:

  ```
  STMFD sp!, {r0-r3, lr} ; push an odd number of registers
  ```

  to

  ```
  STMFD sp!, {r0-r3, r12, lr} ; push even number of registers
  ```

  The assembler automatically marks the object with the `PRES8` attribute if all instructions preserve eight-byte stack alignment, so it is no longer necessary to add the `PRESERVE8` directive to the top of each assembler file.
- If you have any legacy objects/libraries that cannot be rebuilt, either because you do not have the source code, or because the old objects must not be rebuilt (for example, for qualification/certification reasons), then you must inspect the legacy objects to check whether they preserve eight-byte alignment or not.

  Use `fromelf -c` to disassemble the object code. C/C++ code compiled with ADS 1.1 or later normally preserves eight-byte alignment, but assembled code does not.

  If your objects do indeed preserve eight-byte alignment, then the linker error L6238E can be suppressed with the use of `--diag_suppress 6238` on the linker command line.

By using this, you are effectively guaranteeing that these objects are `PRES8`.

The linker warning L6306W is suppressible with `--diag_suppress 6306`.

See the following FAQ:

8 Byte Stack Alignment.

`L6239E: Cannot call non-interworking <t2> symbol '<sym>' in <obj2> from <t1> code in <obj1>(<sec1>)`

Example:

```
Cannot call non-interworking ARM symbol 'ArmFunc' in object foo.o from THUMB code in
bar.o(.text)
```

This problem can be caused by `foo.c` not being compiled with the option `--apcs /interwork`, to enable A32 code to call T32 code (and T32 to A32) by linker-generated interworking veneers.

`L6241E: <objname>(<secname>) cannot use the address of '<attr1>' function <sym> as the image contains '<attr2>' functions.`

When linking with `--strict`, the linker reports conditions that might fail as errors, for example:

```
Error: L6241E: foo.o(.text) cannot use the address of '~IW' function main as the
image contains 'IW' functions.
```

`IW` means interworking, and `~IW` means non-interworking.

`L6242E: Cannot link object <objname> as its attributes are incompatible with the image attributes.`

Each object file generated by the compilation tools includes a set of attributes that indicates the options that it was built with. The linker checks the attributes of each object file it processes. If it finds attributes that are incompatible with those of object files it has loaded previously, it generates this error.

There are three common reasons for this error, each of which produces a different message:

- ```
  Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.
  require four-byte alignment of eight-byte datatypes clashes with require eight-byte alignment of eight-byte data types.
  ```

  This can occur when you try to link objects built using RVCT 2.0 or later with objects built using ADS or RVCT 1.2. In ADS and RVCT 1.2, `double` and `long long` data types were 4-byte aligned (unless you used the `-Oldrd` compiler option or the `__align` keyword). In RVCT 2.0, the ABI changed, so that `double` and `long long` data types are 8-byte aligned.

  This change means that ADS and RVCT 1.2 objects and libraries using `double` or `long long` data types are not directly compatible with objects and libraries built using RVCT 2.0 or later, and so the linker reports an attribute clash.

  To use RVCT 2.x or 3.0 C objects with legacy ADS C objects, compile the RVCT 2.x or 3.0 C code with the `--apcs /adsabi` command line option. This option was deprecated in RVCT 2.2 and removed from RVCT 3.1.

- ```
  Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.
  ... pure-endian double clashes with mixed-endian double.
  ```

  This can occur when you are linking objects built using the ARM Compiler toolchain, RVCT or ADS with legacy SDT objects or objects built using either of the compiler options `--fpu softfpa` or `--fpu fpa`. SDT used a non-standard format for little-endian `double` and big-endian `long long`. However ADS and RVCT use industry-standard `double` and `long long` types, except for when the `--fpu softfpa` or `--fpu fpa` options are used. (These options are only supported in RVCT 2.1 and earlier). If you attempt to link object files that use the different formats for little-endian `double` and big-endian `long long` then the linker reports this error.

  ARM recommends you rebuild your entire project using RVCT or the ARM Compiler toolchain. If you do not have the source code for an object or library, then try recompiling your code with `--fpu softfpa`.

- ```
  Error: L6242E: Cannot link object foo.o as its attributes are incompatible with the image attributes.
  ... FPA clashes with VFP.
  ```

  This error typically occurs when you attempt to link objects built with different `--fpu` options. ARM recommends you rebuild your entire project using the same `--fpu` options.

See the following FAQ:

Are legacy objects and libraries compatible with my project?.

`L6243E: Selector only matches removed unused sections - no section to be FIRST/LAST.`

All sections matching this selector have been removed from the image because they were unused. For more information, `use --info unused`.

`L6244E: <type> region <regionname> address (<addr>) not aligned on a <align> byte boundary.`

`L6245E: Failed to create requested ZI section '<name>'.`
  ```

`L6248E: <objname>(<secname>) in <attr1> region '<r1>' cannot have <rtype> relocation to <symname> in <attr2> region '<r2>'.`

This error can occur when you are trying to build position-independent (PI) code. Consider, for example the following code:

```
#include <stdio.h>
char *str = "test";
int main(void)
{
    printf ("%s",str);
}
```

When you compile and link this using:

```
armcc -c --apcs /ropi/rwpi pi.c
armlink --ropi --rwpi pi.o
```

the linker reports the following error message:

```
Error: L6248E: pi.o(.data) in PI region 'ER_RW' cannot have address type relocation
to .conststring in PI region 'ER_RO'.
```

This is because the compiler generates a global pointer `str` that must be initialized to the address of the string in the `.conststring` section. However, absolute addresses cannot be used in a PI system, so the link step fails.

To resolve this, you must re-write the code to avoid the explicit pointer. You can do this using either of the following methods:

- Use a global array instead of a global pointer, for example:

```
#include <stdio.h>
const char str[] = "test";
int main(void)
{
    printf ("%s",str);
}
```

- Use a local pointer instead of a global pointer, for example:

```
#include <stdio.h>
int main(void)
{
    char *str = "test";
    printf ("%s",str);
}
```

Note

If you are using an array of pointers, such as:

```
char * list[] = {"zero", "one", "two"};
```

the linker reports a separate error for each element in the array. In this case, ARM recommends you declare a two dimensional array for the list, with the first dimension as the number of elements in the array, and the second dimension as the maximum size of an element in the array, for example:

```
char list[3][5] = {"zero", "one", "two"};
```

You must change the `printf()` statement to, for example:

```
printf("%s", list[1]);
```

See compiler error number 1359.

`L6249E: Entry point (<address>) lies within multiple sections.`
`L6250E: Object <objname> contains illegal definition of special symbol <symbol>.`
`L6251E: Object <objname> contains illegal reference to special symbol <symbol>.`
`L6252E: Invalid argument for --xreffrom/--xrefto command: '<arg>'`

`L6253E: Invalid SYMDEF address: <number>.`
`L6254E: Invalid SYMDEF type : <type>.`
The content of the symdefs file is invalid.

See the following in the *armlink User Guide*:

Symdefs file format.

`L6255E: Could not delete file <filename>: <reason>`
An I/O error occurred while trying to delete the specified file. The file was either read-only, or was not found.
`L6257E: <object>(<secname>) cannot be assigned to overlaid Execution region '<ername>'.`
This message indicates a problem with the scatter file.

See the following in the *armlink User Guide*:

Scatter file syntax.

`L6258E: Entry point (<address>) lies in an overlaid Execution region.`
This message indicates a problem with the scatter file.

See the following in the *armlink User Guide*:

Scatter file syntax.

`L6259E: Reserved Word '<name>' cannot be used as a <type> region name.`
`<name>` is a reserved word, so choose a different name for your region.
`L6260E: Multiple load regions with the same name (<regionname>) are not allowed.`
This message indicates a problem with the scatter file.

See the following in the *armlink User Guide*:

Scatter file syntax.

`L6261E: Multiple execution regions with the same name (<regionname>) are not allowed.`
This message indicates a problem with the scatter file.

See the following in the *armlink User Guide*:

Scatter file syntax.

`L6263E: <addr> address of <regionname> cannot be addressed from <pi_or_abs> Region Table in <regtabregionname>`
The Region Table contains information used by the C-library initialization code to copy, decompress, or create ZI. This error message is given when the scatter file specifies an image structure that cannot be described by the Region Table.

The error message is most common when PI and non-PI load regions are mixed in the same image.

`L6265E: Non-PI Section <obj>(<sec>) cannot be assigned to PI Exec region <er>.`
This might be caused by explicitly specifying the wrong ARM library on the linker command-line. Either:

- remove the explicit specification of the ARM library
- replace the library, for example, `c_t.l`, with the correct library.

`L6266E: RWPI Section <obj>(<sec>) cannot be assigned to non-PI Exec region <er>.`
A file compiled with `--apcs=/rwpi` is placed in an execution region that does not have the PI attribute.
`L6271E: Two or more mutually exclusive attributes specified for Load region <regname>`
This message indicates a problem with the scatter file.
`L6272E: Two or more mutually exclusive attributes specified for Execution region <regname>`
This message indicates a problem with the scatter file.

`L6273E: Section <objname>(<secname>) has mutually exclusive attributes (READONLY and ZI)`

This message indicates a problem with the object file.

`L6275E: COMMON section <obj1>(<sec1>) does not define <sym> (defined in <obj2>(<sec2>))`

Given a set of `COMMON` sections with the same name, the linker selects one of them to be added to the image and discards all others. The selected `COMMON` section must define all the symbols defined by any rejected `COMMON` section, otherwise a symbol that was defined by a rejected section would become undefined again. The linker generates an error if the selected copy does not define a symbol that a rejected copy does. This error is normally caused by a compiler fault. Contact your supplier.

`L6276E: Address <addr> marked both as <s1>(from <sp1>(<obj1>) via <src1>) and <s2>(from <sp2>(<obj2>) via <src2>).`

The image cannot contain contradictory mapping symbols for a given address, because the contents of each word in the image are uniquely typed as A32 (`$a`) or T32 (`$t`) code, DATA (`$d`), or NUMBER. It is not possible for a word to be both A32 code and DATA. This might indicate a compiler fault. Contact your supplier.

`L6277E: Unknown command '<cmd>'.`

`L6278E: Missing expected <str>.`

`L6279E: Ambiguous selectors found for <sym> ('<sel1>' and '<sel2>').`

`L6280E: Cannot rename <sym> using the given patterns.`

See the following in the *armlink User Guide*:

RENAME steering file command.

`L6281E: Cannot rename both <sym1> and <sym2> to <newname>.`

See the following in the *armlink User Guide*:

RENAME steering file command.

`L6282E: Cannot rename <sym> to <newname> as a global symbol of that name exists (defined) in <obj>).`

See the following in the *armlink User Guide*:

RENAME steering file command.

`L6283E: Object <objname> contains illegal local reference to symbol <symbolname>.`

An object cannot contain a reference to a local symbol, because local symbols are always defined within the object itself.

`L6285E: Non-relocatable Load region <lr_name> contains R-Type dynamic relocations. First R-Type dynamic relocation found in <object>(<secname>) at offset 0x<offset>.`

This error occurs where there is a PI reference between two separate segments, if the two segments can be moved apart at runtime. When the linker sees that the two sections can be moved apart at runtime it generates a relocation (an R-Type relocation) that can be resolved if the sections are moved from their statically linked address. However the linker faults this relocation (giving error L6285E) because PI regions must not have relocations with respect to other sections as this invalidates the criteria for being position independent.

`L6286E: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) with respect to {symname|%s}. Value(<val>) out of range(<range>) for (<rtype>)`

This error typically occurs in the following situations:

- In handwritten assembly code, where there are not enough bits within the instruction opcode to hold the offset to a symbol.

  For example, the offset range is ±4095 for an A32 state `LDR` or `STR` instruction.
- When the linker is having difficulty placing veneers around a large code section in your image.

  When the linker places a veneer near a very large section it must decide whether to place the veneer before or after the section. When the linker has placed the veneer it might have to place more veneers, which could be placed between the original veneer and its target. This would increase the distance between the veneer and its target.

  The linker automatically allows for modest increases in distances between veneers and their targets. However, a large number of veneers placed between the original veneer and its target might result in the target moving out of range. If this occurs, the linker generates message L6286E.

  To work around this, you can move large code sections away from areas where the linker is placing many veneers. This can be done either by placing large sections in their own regions or by placing them first in the region they are located in using the `+FIRST` directive in the scatter-loading description file.

  For example:

  ```
  LOAD 0x0A000000 0x1000000
  {
     ROM1 +0x0
     {
        *(+RO)
     }
  }
  ```

  This can be changed to:

  ```
  LOAD 0x0A000000 0x1000000
  {
     ROM1 +0x0
     {
        *(+RO)
     }
     ROM1A +0x0
     {
        large.o (+RO)
     }
  }
  ```
- When `.ARM.exidx` exception-handling index tables are placed in different execution regions, or too far from exception handling code.

  The `.ARM.exidx` exception-handling index tables must be located in a single execution region. Also, the distance from these tables to the C++ code that uses C++ exception handling must be within the range -0x40000000 to 0x3fffffff. Otherwise, the linker reports the following error:

  `L6286: Value(0x9ff38980) out of range(-0x9ff38980) out of range(-0x40000000 - 0x3fffffff) for relocation #0 (R_ARM_PREL31), wrt symbol xxx in XXXX.o(.ARM.exidx)`

  This behavior is specified in the *Exception Handling ABI for the ARM Architecture* (EHABI). The EHABI states that the `R_ARM_PREL31` relocation, which `.ARM.exidx` uses, does not use the highest bit (bit 31) for calculating the relocation.

  The most likely cause of this is that C++ code that must access the `.ARM.exidx` sections, has been split and placed into separate execution regions, outside of the valid range (-0x40000000 to 0x3fffffff).

To resolve this error, if you have memory between the separated execution regions, place the `.ARM.exidx` section there with the selector `*(.ARM.exidx)`. For example:

```
LOAD_ROM 0x00000000
{
   ER1 0x00000000   ; The distance from ER2 to ER1 is out of
   {                ; range.
      file1.o (+RO) ; From a C++ source.
      * (+RO)
   }
   ERx 0x30000000
   {
      *(.ARM.exidx) ; ARM.exidx to ER1 and ER2 both in range.
   }
   ER2 0x60000000
   {
      file2.o (+RO) ; From a C++ source.
   }
   ER3 +0
   {
      * (+RW, +ZI)
   }
}
```

Otherwise, try placing the code into an execution region close enough to the tables (within the range of -0x40000000 to 0x3fffffff).

In other cases, make sure you have the latest patch installed from *Downloads*.

For more information, see the following:

What does "Error: L6286E: Value out of range for relocation" mean?

Exception Handling ABI for the ARM Architecture.

`L6287E: Illegal alignment constraint (<align>) specified for <objname>(<secname>).`
An illegal alignment was specified for an ELF object.

`L6291E: Cannot assign Fixed Execution Region <ername> Load Address:<addr>. Load Address must be greater than or equal to next available Load Address:<load_addr>.`
See the following in the *armlink User Guide*:

Execution region attributes.

`L6292E: Ignoring unknown attribute '<attr>' specified for region <regname>.`
This error message is specific to execution regions with the `FIXED` attribute. `FIXED` means make the load address the same as the execution address. The linker can only do this if the execution address is greater than or equal to the next available load address within the load region.

See the following in the *armlink User Guide*:

`L6294E: <type> region <regionname> spans beyond 32 bit address space (base <base>, size <size> bytes).`
This error message relates to a problem with the scatter file.

`L6295E: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) with respect to <symname> SBREL relocation requires image to be RWPI`

`L6296E: Definition of special symbol <sym1> is illegal as symbol <sym2> is absolute.`
See L6188E.

`L6300W: Common section <object1>(<section1>) is larger than its definition <object2>(<section2>).`
This might indicate a compiler fault. Contact your supplier.

`L6301W: Could not find file <filename>: <reason>`
The specified file was not found in the default directories.

`L6302W: Ignoring multiple SHLNAME entry.`
There can be only one `SHLNAME` entry in an edit file. Only the first such entry is accepted by the linker. All subsequent `SHLNAME` entries are ignored.

`L6304W: Duplicate input file <filename> ignored.`
The specified filename occurred more than once in the list of input files.

`L6305W: Image does not have an entry point. (Not specified or not set due to multiple choices.)`

The entry point for the ELF image was either not specified, or was not set because there was more than one section with an entry point linked-in. You must use linker option `--entry` to specify the single, unique entry, for example:

```
--entry 0x0
```

or

```
--entry <label>
```

The label form is typical for an embedded system.

`L6306W: '<attr1>' section <objname>(<secname>) should not use the address of '<attr2>' function <sym>.`

See L6238E.

`L6307W: Relocation #<rel_class>:<rel_num> in <objname>(<secname>) with respect to <sym>. Branch to unaligned destination.`

`L6308W: Could not find any object matching <membername> in library <libraryname>.`

The name of an object in a library is specified on the link-line, but the library does not contain an object with that name.

`L6309W: Library <libraryname> does not contain any members.`

A library is specified on the linker command-line, but the library does not contain any members.

`L6310W: Unable to find ARM libraries.`

This is most often caused by incorrect arguments to `--libpath`.

Set the correct path with the `--libpath` linker option. The default path for a Windows installation is:

`install_directory``\lib`

Ensure this path does not include any of the following:

- `\armlib`
- `\cpplib`
- any trailing slashes (\) at the end. These are added by the linker automatically.

Use `--verbose` or `--info libraries` to display where the linker is attempting to locate the libraries.

See the following in the *armlink User Guide*:

- *--info=topic[,topic,...]*.
- *--libpath=pathlist*.
- *--verbose*.

See the following in the *Getting Started Guide*:

- *Toolchain environment variables*.

`L6311W: Undefined symbol <symbol> (referred from <objname>).`

See L6218E.

`L6312W: Empty <type> region description for region <region>`

`L6313W: Using <oldname> as an section selector is obsolete. Please use <newname> instead.`

For example, use of `IWV$$Code` within the scatter file is obsolete. Replace `IWV$$Code` with `Veneer$$Code`.

`L6314W: No section matches pattern <module>(<section>).`

For example:

```
No section matches pattern foo.*o(ZI).
```

This can be caused by any of the following:

- The file `foo.o` is mentioned in your scatter file, but it is not listed on the linker command line. To resolve this, add `foo.o` to the link line.
- You are trying to place the ZI data of `foo.o` using a scatter file, but `foo.o` does not contain any ZI data. To resolve this, remove the `+ZI` attribute from the foo.o line in your scatter file.
- You have used `__attribute__((at(`*address*`)))` in your source code to place code and data at a specific address. You have also specified `*(.ARM.__AT_`*address*`)` in a scatter file, but you have not specified the address as eight hexadecimal digits. For example, if you specify `__attribute__((at(0x10000)))` in your source code, then you must specify the section name as `*(.ARM.__AT_0x00010000)` in the scatter file.

See the following in the *armlink User Guide*:

- *Methods of placing functions and data at specific addresses*.
- *Placement of sections at a specific address with __at*.

`L6315W: Ignoring multiple Build Attribute symbols in Object <objname>.`

An object can contain at most one absolute `BuildAttribute$$...` symbol. Only the first such symbol from the object symbol table is accepted by the linker. All subsequent ones are ignored.

`L6316W: Ignoring multiple Build Attribute symbols in Object <objname> for section <sec_no>`

An object can contain at most one `BuildAttribute$$...` symbol applicable to a given section. Only the first such symbol from the object symbol table is accepted by the linker. All subsequent ones are ignored.

`L6317W: <objname>(<secname>) should not use the address of '<attr1>' function <sym> as the image contains '<attr2>' functions.`

`L6318W: <objname>(<secname>) contains branch to a non-code symbol <sym>.`

This warning means that in the (usually assembler) file, there is a branch to a non-code symbol (in another `AREA`) in the same file. This is most likely a branch to a label or address where there is data, not code.

For example:

```
AREA foo, CODE
B bar
AREA bar, DATA
DCD 0
END
```

This results in the message:

```
init.o(foo) contains branch to a non-code symbol bar.
```

If the destination has no name:

```
BL 0x200 ; Branch with link to 0x200 bytes ahead of PC
```

the following message is displayed:

```
bootsys.o(BOOTSYS_IVT) contains branch to a non-code symbol <Anonymous Symbol>.
```

This warning can also appear when linking objects generated by GCC. GCC uses linker relocations for references internal to each object. The targets of these relocations might not have appropriate mapping symbols that permit the linker to determine whether the target is code or data, so a warning is generated. By contrast, `armcc` resolves all such references at compile-time.

`L6319W: Ignoring <cmd> command. Cannot find section <objname>(<secname>).`
For example, when building a Linux application, you might have:

```
--keep *(.init_array)
```

on the linker command-line in your makefile, but this section might not be present when building with no C++, in which case this warning is reported:

```
Ignoring --keep command. Cannot find section *(.init_array)
```

You can often ignore this warning, or suppress it with `--diag_suppress 6319`.

`L6320W: Ignoring <cmd> command. Cannot find argument '<argname>'.`
`L6323W: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) with respect to <sym>. Multiple variants exist. Using the <type> variant to resolve ambiguity`
`L6324W: Ignoring <attr> attribute specified for Load region <regname>.`
This attribute is applicable to execution regions only. If specified for a Load region, the linker ignores it.
`L6325W: Ignoring <attr> attribute for region <regname> which uses the +offset form of base address.`
This attribute is not applicable to regions using the `+offset` form of base address. If specified for a region, which uses the `+offset` form, the linker ignores it.

A region that uses the `+offset` form of base address inherits the `PI`, `RELOC`, or `OVERLAY` attributes from either:

- the previous region in the description
- the parent load region if it is the first execution region in the load region.

See the following in the *armlink User Guide*:

- *Inheritance rules for load region address attributes*.
- *Inheritance rules for execution region address attributes*.
- *Inheritance rules for the RELOC address attribute*.

`L6326W: Ignoring ZEROPAD attribute for non-root execution region <ername>.`
`ZEROPAD` only applies to root execution regions. A root region is a region whose execution address is the same as its load address, and so does not require moving or copying at run-time.

See the following in the *armlink User Guide*:

Execution region attributes.

`L6329W: Pattern <module>(<section>) only matches removed unused sections.`
All sections matching this pattern have been removed from the image because they were unused. For more information, use `--info unused`.
See the following in the *armlink User Guide*:

- *Elimination of unused sections*.
- *--info=topic[,topic,...]*.

`L6330W: Undefined symbol <symbol> (referred from <objname>). Unused section has been removed.`
This means that an unused section has had its base and limit symbols referenced. For more information, use `--info unused`.

See the following in the *armlink User Guide*:

`L6331W: No eligible global symbol matches pattern <pat>.`
`L6332W: Undefined symbol <sym1> (referred from <obj1>). Resolved to symbol <sym2>.`
`L6334W: Overalignment <overalignment> for region <regname> cannot be negative.`
See the following in the *armlink User Guide*:

Overalignment of execution regions and input sections.

`L6335W: ARM interworking code in <objname>(<secname>) may contain invalid tailcalls to ARM non-interworking code.`

The compiler is able to perform tailcall optimization for improved code size and performance. However, there is a problematic sequence for ARMv4T code in which a T32 interworking (IW) function calls (by a veneer) an A32 IW function, which tailcalls an A32 non-interworking (`~IW`) function. The return from the A32 non-IW function can pop the return address off the stack into the PC instead of using the correct `BX` instruction. The linker can detect this situation and report this warning.

T32 IW tailcalls to T32 non-IW do not occur because T32 tailcalls with `B` are so short ranged that they can only be generated to functions in the same ELF section which must also be T32.

The warning is pessimistic in that an object *might* contain invalid tailcalls, but the linker cannot be sure because it only looks at the attributes of the objects, not at the contents of their sections.

To avoid the warning, either recompile your entire code base, including any user libraries, with `--apcs /interwork`, or manually inspect the A32 IW function to check for tailcalls (that is, where function calls are made using an ordinary branch `B` instruction), to check whether this is a real problem. This warning can be suppressed with `--diag_suppress L6335W`.

`L6337W: Common code sections <o1>(<s1>) and <o2>(<s2>) have incompatible floating-point linkage`

`L6339W: Ignoring RELOC attribute for execution region <er_name>.`

Execution regions cannot explicitly be given the `RELOC` attribute. They can only gain this attribute by inheriting it from the parent load region or the previous execution region if using the `+offset` form of addressing.

See the following in the *armlink User Guide*:

Execution region attributes.

`L6340W: options first and last are ignored for link type of <linktype>`

The `--first` and `--last` options are meaningless when creating a partially-linked object.

`L6366E: <object> attributes<attr> are not compatible with the provided cpu and fpu attributes<cli> <diff>.`

`L6367E: <object>(<section>) attributes<attr> are not compatible with the provided cpu and fpu attributes<cli> <diff>`

`L6368E: <symbol> defined in <object>(<section>) attributes<attr> are not compatible with the provided cpu and fpu attributes<cli> <diff>`

`L6369E: <symbol> defined in <object>(ABSOLUTE) are not compatible with the provided cpu and fpu Attributes<cli> <diff>`

`L6370E: cpu <cpu> is not compatible with fpu <fpu>`

See the following in the *armlink User Guide*:

- *--cpu=name*.
- *--fpu=name*.

`L6371E: Adding attributes from cpu and fpu: <attrs>`

`L6372E: Image needs at least one load region.`

`L6373E: libattrs.map file not found in System Library directory <dir>. Library selection may be impaired.`

`L6384E: No Load Execution Region of name <region> seen yet at line <line>.`

This might be because you have used the current base address in a limit calculation in a scatter file. For example:

```
ER_foo 0 ImageBase(ER_foo)
```

`L6385W: Addition overflow on line <line>`

`L6386E: Exec Region Expressions can only be used in base address calculations on line <line>`

`L6387E: Load Region Expressions can only be used in ScatterAssert expressions on line <line>`

See the following in the *armlink User Guide*:

ScatterAssert function and load address related functions.

`L6388E: ScatterAssert expression <expr> failed on line <line>`

See the following in the *armlink User Guide*:

ScatterAssert function and load address related functions.

`L6389E: Load Region <name> on line <line> not yet complete, cannot use operations that depend on length of region`

`L6390E: Conditional operator (expr) ? (expr) : (expr) on line <line> has no : (expr).`

See the following in the *armlink User Guide*:

- *About Expression evaluation in scatter files*.
- *Expression rules in scatter files*.

`L6404W: FILL value preferred to combination of EMPTY, ZEROPAD and PADVALUE for Execution Region <name>.`

See the following in the *armlink User Guide*:

Execution region attributes.

`L6405W: No .ANY selector matches Section <name>(<objname>).`

See the following in the *armlink User Guide*:

Placement of unassigned sections with the .ANY module selector.

`L6406W: No space in execution regions with .ANY selector matching Section <name>(<objname>).`

This occurs if there is not sufficient space in the scatter file regions containing `.ANY` to place the section listed. You must modify your scatter file to ensure there is sufficient space for the section.

See the following in the *armlink User Guide*:

Placement of unassigned sections with the .ANY module selector.

`L6407W: Sections of aggregate size 0x<size> bytes could not fit into .ANY selector(s).`

This warning identifies the total amount of image data that cannot be placed in any `.ANY` selectors.

For example, `.ANY(+ZI)` is placed in an execution region that is too small for the amount of `ZI` data:

```
ROM_LOAD 0x8000
{
    ROM_EXEC 0x8000
    {
        .ANY(+RO,+RW)
    }
    RAM +0 0x{...} <<< region max length is too small
    {
        .ANY(+ZI)
    }
}
```

See the following in the *armlink User Guide*:

Placement of unassigned sections with the .ANY module selector.

`L6408W: Output is --fpic yet section <sec> from <obj> has no FPIC attribute.`
`L6409W: Output is --fpic yet object <obj> has no FPIC attribute.`
`L6410W: Symbol <sym> with non STV_DEFAULT visibility <vis> should be resolved statically, cannot use definition in <lib>.`

`L6411W: No compatible library exists with a definition of startup symbol <name>.`
`L6412W: Disabling merging for section <sec> from object <obj>, non R_ARM_ABS32 relocation from section <srcsec> from object <srcobj>`
`L6413W: Disabling merging for section <sec> from object <obj>, Section contains misaligned string(s).`
`L6414E: --ropi used without --rwpi or --rw-base.`

This error relates to options that are unsupported in ARM Compiler 6.0.

`L6415E: Could not find a unique set of libraries compatible with this image. Suggest using the --cpu option to select a specific library.`

See the following in the *armlink User Guide*:

--cpu=name.

`L6416E: Relocation <type> at <relclass>:<idx> <objname>(<secname>) cannot be veneered as it has an offset <offset> from its target.`
`L6417W: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is with respect to a reserved tagging symbol(#<idx>).`
`L6418W: Tagging symbol <symname> defined in <objname>(<secname>) is not recognized.`
`L6419W: Undefined symbol <symbol> (referred from <objname>) imported.`
`L6420E: Ignoring <oepname>(<secname>:<secnum>) as it is not of a recognized type.`
`L6422U: PLT generation requires an architecture with ARM instruction support.`

For the linker to generate a *Procedure Linkage Table* (PLT), you must be using a target that supports the A32 instruction set. For example, the linker cannot generate a PLT for a Cortex-M3 target.

`L6423E: Within the same collection, section <secname> cannot have different sort attributes.`
`L6424E: Within the same collection, section <secname1> and section <secname2> cannot be separated into different execution regions.`
`L6425E: Within the same collection, section <secname> cannot have their section names with different length.`
`L6426E: Within the same collection, section <secname> cannot have its name duplicated.`
`L6427E: Cannot rename <sym> to <newname> as it has already been renamed to <name>).`
`L6429U: Attempt to set maximum number of open files to <val> failed with error code <error>.`

An attempt to increase the number of file handles `armlink` can keep open at any one time has failed.

`L6431W: Ignoring incompatible enum size attribute on Symbol <symbol> defined in <object>(<section>).`
`L6432W: Ignoring incompatible enum size attribute on Object <object>(<section>).`
`L6433W: Ignoring incompatible enum size attribute on object <object>.`
`L6434W: Ignoring incompatible wchar_t size attribute on Symbol <symbol> defined in <object>(<section>).`
`L6435W: Ignoring incompatible wchar_t size attribute on Section <object>(<section>).`
`L6436W: Ignoring incompatible wchar_t size attribute on object <object>.`
`L6437W: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <armsym>. Branch relocation to untyped symol in object <armobjname>, target state unknown.`
`L6438E: __AT section <objname>(<secname>) address <address> must be at least 4 byte aligned.`
`L6439W: Multiply defined Global Symbol <sym> defined in <objname>(<secname>) rejected in favour of Symbol defined in <selobj>(<selsec>).`
`L6440E: Unexpected failure in link-time code generation`
`L6441U: System call to get maximum number of open files failed <error>.`
`L6442U: Linker requires a minimum of <min> open files, current system limit is <max> files.`

`L6443W: Data Compression for region <region> turned off. Region contains reference to symbol <symname> which depends on a compressed address.`

The linker requires the contents of a region to be fixed before it can be compressed and cannot modify it after it has been compressed. Therefore a compressible region cannot refer to a memory location that depends on the compression process.

`L6444I: symbol visibility : <symname> set to <visibility>.`

`L6445I: symbol visibility : <symname> merged to <set_vis> from existing <old_vis> and new <new_vis>.`

`L6447E: SHT_PREINIT_ARRAY sections are not permitted in shared objects.`

`L6448W: While processing <filename>: <message>`

`L6449E: While processing <filename>: <message>`

`L6450U: Cannot find library <libname>.`

`L6451E: <object> built permitting Thumb is forbidden in an ARM-only link.`

`L6452E: <object>(<section>) built permitting Thumb is forbidden in an ARM-only link.`

`L6453E: <symbol> defined in <object>(<section>) built permitting Thumb is forbidden in an ARM-only link.`

`L6454E: <symbol> defined in <object>(ABSOLUTE) built permitting Thumb is forbidden in an ARM-only link.`

`L6455E: Symbol <symbolname> has deprecated ARM/Thumb Synonym definitions (by <object1> and <object2>).`

`L6459U: Could not create temporary file.`

`L6462E: Reference to <sym> from a shared library only matches a definition with Hidden or Protected Visibility in Object <obj>.`

`L6463U: Input Objects contain <archtype> instructions but could not find valid target for <archtype> architecture based on object attributes. Suggest using --cpu option to select a specific cpu.`

See the following in the *armlink User Guide*:

--cpu=name.

`L6464E: Only one of --dynamic_debug, --emit-relocs and --emit-debug-overlay-relocs can be selected.`

See the following in the *armlink User Guide*:

- *--dynamic_debug*.
- *--emit_debug_overlay_relocs*.
- *--emit_relocs*.

`L6467W: Library reports remark: <msg>`

`L6468U: Only --pltgot=direct or --pltgot=none supported for --base_platform with multiple Load Regions containing code.`

See the following in the *armlink User Guide*:

- *--base_platform*.
- *--pltgot=type*.

`L6469E: --base_platform does not support RELOC Load Regions containing non RELOC Execution Regions. Please use +0 for the Base Address of Execution Region <ername> in Load Region <lrname>.`

See the following in the *armlink User Guide*:

- *--base_platform*.
- *Inheritance rules for the RELOC address attribute*.

`L6470E: PLT section <secname> cannot be moved outside Load Region <lrname>.`

`L6471E: Branch Relocation <rel_class>:<idx> in section <secname> from object <objname> refers to ARM Absolute <armsym> symbol from object <armobjname>, Suppress error to treat as a Thumb address.`

`L6475W: IMPORT/EXPORT commands ignored when --override_visibility is not specified`

The symbol you are trying to export, either with an `EXPORT` command in a steering file or with the `--undefined_and_export` command-line option, is not exported because of low visibility. See the following in the *armlink User Guide*:

- *--override_visibility*.
- *--undefined_and_export=symbol*.
- *EXPORT*.

`L6616E: Cannot increase size of RegionTable <sec_name> from <obj_name>`
`L6617E: Cannot increase size of ZISectionTable <sec_name> from <obj_name>`
`L6629E: Unmatched parentheses expecting ) but found <character> at position <col> on line <line>`

This message indicates a parsing error in the scatter file.

`L6630E: Invalid token start expected number or ( but found <character> at position <col> on line <line>`

This message indicates a parsing error in the scatter file.

`L6631E: Division by zero on line <line>`

This message indicates an expression evaluation error in the scatter file.

`L6632W: Subtraction underflow on line <line>`

This message indicates an expression evaluation error in the scatter file.

`L6634E: Pre-processor command in '<filename>'too long, maximum length of <max_size>`

This message indicates a problem with pre-processing the scatter file.

`L6635E: Could not open intermediate file '<filename>' produced by pre-processor: <reason>`

This message indicates a problem with pre-processing the scatter file.

`L6636E: Pre-processor step failed for '<filename>'`

This message indicates a problem with pre-processing the scatter file.

`L6637W: No input objects specified. At least one input object or library(object) must be specified.`

At least one input object or library(object) must be specified.

`L6638U: Object <objname> has a link order dependency cycle, check sections with SHF_LINK_ORDER`
`L6640E: PDTTable section not least static data address, least static data section is <secname>`

Systems that implement shared libraries with RWPI use a *process data table* (PDT). It is created at static link time by the linker and must be placed first in the data area of the image.

This message indicates that the scatter file does not permit placing the PDT first in the data area of the image.

To avoid the message, adjust your scatter file so that the PDT is placed correctly. This message can also be triggered if you accidentally build object files with `--apcs=/rwpi`.

`L6642W: Unused virtual function elimination might not work correctly, because <obj_name> has not been compiled with --vfe`
`L6643E: The virtual function elimination information in section <sectionname> refers to the wrong section.`

This message might indicate a compiler fault. Contact your supplier.

`L6644E: Unexpectedly reached the end of the buffer when reading the virtual function elimination information in section <oepname>(<sectionname>).`

This message might indicate a compiler fault. Contact your supplier.

`L6645E: The virtual function elimination information in section <oepname>(<sectionname>) is incorrect: there should be a relocation at offset <offset>.`

This message might indicate a compiler fault. Contact your supplier.

`L6646W: The virtual function elimination information in section <oepname>(<sectionname>) contains garbage from offset <offset> onwards.`

This message might indicate a compiler fault. Contact your supplier.

L6647E: The virtual function elimination information for <vcall_objectname>(<vcall_sectionname>) incorrectly indicates that section <curr_sectionname>(object <curr_objectname>), offset <offset> is a relocation (to a virtual function or RTTI), but there is no relocation at that offset.

This message might indicate a compiler fault. Contact your supplier.

L6649E: EMPTY region <regname> must have a maximum size.

See the following in the *armlink User Guide*:

Execution region attributes.

L6650E: Object <objname> Group section <sectionidx> contains invalid symbol index <symidx>.

L6651E: Section <secname> from object <objname> has SHF_GROUP flag but is not member of any group.

L6652E: Cannot reverse Byte Order of Data Sections, input objects are <inputendian> requested data byte order is <dataendian>.

L6654E: Rejected Local symbol <symname> referred to from a non group member <objname>(<nongrpname>)

This message might indicate a compiler fault. Contact your supplier.

L6656E: Internal error: the vfe section list contains a non-vfe section called <oepname>(<secname>).

This message might indicate a compiler fault. Contact your supplier.

L6664W: Relocation #<rel_class>:<rel_number> in <objname>(<secname>) is with respect to a symbol(#<idx> before last Map Symbol #<last>).

L6665W: Neither Lib$$Request$$armlib Lib$$Request$$cpplib defined, not searching ARM libraries.

The following code produces this warning:

```
  AREA Block, CODE, READONLY
  EXPORT func1
  ;IMPORT ||Lib$$Request$$armlib||
  IMPORT printf
func1
  LDR r0,=string
  BL printf
  BX lr
  AREA BlockData, DATA
string DCB "mystring"
  END
```

The linker has not been told to look in the libraries and so cannot find the symbol `printf`.

This also causes the following error:

```
L6218E: Undefined symbol printf (referred from L6665W.o).
```

If you do not want the libraries, then ignore this message. Otherwise, to fix both the error and the warning uncomment the line:

```
IMPORT ||Lib$$Request$$armlib||
```

L6679W: Data in output ELF section #<sec> '<secname>' was not suitable for compression (<data_size> bytes to <compressed_size> bytes).

L6682E: Merge Section <oepname>(<spname>) is a code section

L6683E: Merge Section <oepname>(<spname>) has an element size of zero

L6684E: Section <spname> from object <oepname> has SHF_STRINGS flag but not SHF_MERGE flag

L6685E: Section <spname> from object <oepname> has a branch reloc <rel_idx> to a SHF_MERGE section

L6688U: Relocation #<rel_class>:<rel_idx> in <oepname>(<spname>) references a negative element

L6689U: Relocation #<rel_class>:<rel_idx> in <oepname>(<spname>). Destination is in the middle of a multibyte character

`L6690U: Merge Section <spname> from object <oepname> has no symbols`
`L6703W: Section <er> implicitly marked as non-compressible.`
`L6707E: Padding value not specified with PADVALUE attribute for execution region <regionname>.`

See the following in the *armlink User Guide*:

Execution region attributes.

`L6708E: Could not process debug frame from <secname> from object <oepname>.`
`L6709E: Could not associate fde from <secname> from object <oepname>.`
`L6713W: Function at offset <offset> in <oepname>(<secname>) has no symbol.`
`L6714W: Exception index table section .ARM.exidx from object <oepname> has no data.`
`L6720U: Exception table <spname> from object <oepname> present in image, --noexceptions specified.`

See the following in the *armlink User Guide*:

--exceptions, --no_exceptions.

`L6721E: Section #<secnum> '<secname>' in <oepname> is not recognized and cannot be processed generically.`
`L6725W: Unused virtual function elimination might not work correctly, because there are dynamic relocations.`
`L6728U: Link order dependency on invalid section number <to> from section number <from>.`
`L6730W: Relocation #<rel_class>:<index> in <objname>(<secname>) with respect to <name>. Symbol has ABI type <type>, legacy type <legacy_type>.`

This warning relates to a change in linker behavior between RVCT 2.0 and 2.1.

Note

The following example produces a warning message only when `--strict_relocations` is used, or when the input objects are from RVCT 2.0 or earlier.

Example:

```
    AREA foo, CODE, READONLY
    CODE32
    ENTRY
    KEEP
func proc
    NOP
    ENDP
    DCD foo
    END
```

In RVCT 2.0 and earlier, the linker determines whether interworking is needed based on the content, which in this example is ARM code. In RVCT 2.1 and later, the linker follows the ABI, which defines that it is the type of the symbol, in this example `STT_SECTION` (which is interpreted as data), that determines whether interworking is applied.

The simplest solution is to move the data into a separate data area in the assembly source file.

Alternatively, you can use `--diag_suppress 6730` to suppress this warning.

`L6731W: Unused virtual function elimination might not work correctly, because the section referred to from <secname> does not exist.`
`L6733W: <objname>(<secname>) contains offset relocation from <lr1name> to <lr2name>, load regions must be rigidly relative.`
`L6738E: Relocation #<rel_class>:<relocnum> in <oepname>(<secname>) with respect to <wrtsym> is a GOT-relative relocation, but _GLOBAL_OFFSET_TABLE_ is undefined.`

Some GNU produced images can refer to the symbol named `_GLOBAL_OFFSET_TABLE_`. If there are no `GOT Slot` generating relocations and the linker is unable to pick a suitable address for the `GOT base` the linker issues this error message.

L6739E: Version '<vername>' has a dependency to undefined version '<depname>'.
L6740W: Symbol '<symname>' versioned '<vername>' defined in '<symverscr>' but not found in any input object.
L6741E: Versioned symbol binding should be 'local:' or 'global:'.
L6742E: Symbol '<symname>' defined by '<oepname>'. Cannot not match to default version symbol '<defversym>'
L6743E: Relocation #<rel_class>:<index> in <oepname>(<spname>) with respect to <symname> that has an alternate def. Internal consistency check failed
L6744E: Relocation #<rel_class>:<index> <oepname>(<spname>) with respect to undefined symbol <symname>. Internal consistency check:
L6745E: Target CPU <cpuname> does not Support ARM, <objname>(<secname>) contains ARM code
L6747W: Raising target architecture from <oldversion> to <newversion>.

If the linker detects objects that specify the obsolete ARMv3, it upgrades these to ARMv4 to be usable with ARM libraries.

L6748U: Missing dynamic array, symbol table or string table in file <oepname>.
L6751E: No such sorting algorithm <str> available.
L6753E: CallTree sorting needs Entry Point to lie within a CallTree Sort ER.
L6761E: Removing symbol <symname>.
L6762E: Cannot build '<type>' PLT entries when building a <imgtype>.
L6763W: '<optname>' cannot be used when building a shared object or DLL. Switching it off
L6764E: Cannot create a PLT entry for target architecture 4T that calls Thumb symbol <symname>
L6765W: Shared object entry points must be ARM-state when linking architecture 4T objects.

This can occur when linking with GNU C libraries. The GNU startup code `crt1.o` does not have any build attributes for the entry point, so the linker cannot determine which execution state (A32 or T32) the code runs in. Because the GNU C library startup code is A32 code, you can safely ignore this warning, or you can suppress it by using `--diag_suppress 6765`.

L6766W: PLT entries for architecture 4T do not support incremental linking.
L6769E: Relocation #<rel_class>:<relocnum> in <oepname>(<secname>) with respect to <wrtsym>. No GOTSLOTexists for symbol.
L6770E: The size and content of the dynamic array changed too late to be fixed.
L6771W: <oepname>(<secname>) contains one or more address-type relocations in RO data. Making section RW to be dynamically relocated at run-time.
L6772W: IMPORT <symname> command ignored when building --sysv.
L6774W: <objname>(<secname>) has debug frame entries of a bad length.
L6775W: <objname>(<secname>) has FDEs which use CIEs which are not in this section.
L6776W: The debug frame in <objname>(<secname>) does not describe an executable section.
L6777W: The debug frame in <objname>(<secname>) has <actual> relocations (expected <expected>)
L6778W: The debug frame in <objname>(<secname>) uses 64-bit DWARF.
L6780W: <origvis> visibility removed from symbol '<symname>' through <impexp>.
L6781E: Value(<val>) Cannot be represented by partition number <part> for relocation #<rel_class>:<rel_number> (<rtype>, wrt symbol <symname>) in <objname>(<secname>)
L6782W: Relocation #<rel_class>:<relnum> '<rtype>' in <oepname> may not access data correctly alongside <pltgot_type> PLT entries
L6783E: Mapping symbol #<symnum> '<msym>' in <oepname>(<secnum>:<secname>) defined at the end of, or beyond, the section size (symbol offset=0x<moffset>, section size=0x<moffset>, section size=0x<secsize>)

This indicates that the address for a section points to a location at the end of or outside of the ELF section. This can be caused by an empty inlined data section and indicates there might be a problem with the object file. You can use `--diag_warning 6783` to suppress this error.

`L6784E: Symbol #<symnum> '<symname>' in <oepname>(<secnum>:<secname>) with value <value> has size 0x<size> that extends to outside the section.`

The linker encountered a symbol with a size that extends outside of its containing section. This message is only a warning by default in the RVCT 2.2 build 503 and later toolchains. Use `--diag_warning 6784` to suppress this error.

`L6785U: Symbol '<symname>' marked for import from '<libname>' already defined by '<oepname>'`

`L6786W: Mapping symbol #<symnum> '<msym>' in <oepname>(<secnum>:<secname>) defined at unaligned offset=0x<moffset>`

`L6787U: Region table handler '<handlername>' needed by entry for <regionname> was not found.`

`L6788E: Scatter-loading of execution region <er1name> to [<base1>,<limit1>) will cause the contents of execution region <er2name> at [<base2>,<limit2>) to be corrupted at run-time.`

This occurs when scatter-loading takes place and an execution region is put in a position where it partially or wholly overwrites another execution region (which can be itself or another region).

For example, the following code generates this error:

```
LOAD_ROM 0x0000 0x4000
{
 EXEC1 0x4000 0x4000
 {
 * (+RW,+ZI)
 }
 EXEC2 0x0000 0x4000
 {
 * (+RO)
 }
}
```

and reports:

```
Error: L6788E: Scatter-loading of execution region EXEC2 will cause the contents of
execution region EXEC2 to be corrupted at run-time.
```

This code does not generate the error:

```
LOAD_ROM 0x0000 0x4000
{
 EXEC1 0x0000 0x4000
 {
 * (+RO)
 }
 EXEC2 0x4000 0x4000
 {
 * (+RW,+ZI)
 }
}
```

See the following in the *armlink User Guide*:

Information about scatter files.

`L6789U: Library <library> member <filename> : Endianness mismatch.`

`L6790E: Relocation #<rel_class>:<relnum> in <objname>(<secname>) with respect to <symname>. May not IMPORT weak reference through GOT-generating relocation`

`L6791E: Unknown personality routine <pr> at 0x<offset> <oepname>(<secname>).`

`L6792E: Descriptor at offset 0x<offset> <oepname>(<secname>).`

`L6793E: Expecting Landing pad reference at offset 0x<offset> in cleanup descriptor <oepname>(<secname>).`

`L6794E: Expecting Landing pad reference at offset 0x<offset> in catch descriptor <oepname>(<secname>).`

`L6795E: Expecting RTTI reference at offset 0x<offset> in catch descriptor <oepname>(<secname>).`

`L6796E: Descriptor at offset 0x<offset> <oepname>(<secname>) overruns end of section.`

`L6797E: Data at Offset 0x<offset> in exception table <oepname>(<secname>) overruns end of section`
`L6798E: Expecting RTTI reference at offset 0x<offset> in Function Specification descriptor <oepname>(<secname>).`
`L6799E: Expecting Landing Pad reference at offset 0x<offset> in Function Specification descriptor <oepname>(<secname>).`

A landing pad is code that cleans up after an exception has been raised. If the linker detects old-format exception tables, it automatically converts them to the new format.

This message does not appear unless you are using a later version of the linker with an earlier version of the compiler.

`L6800W: Cannot convert generic model personality routine at 0x<offset> <oepname>(<secname>).`

A personality routine unwinds the exception handling stack. If the linker detects old-format exception tables then it automatically converts them to the new format. This message indicates a fault in the compiler. Contact your supplier.

`L6801E: <objname>(<secname>) containing <secarmthumb> code cannot use the address of '~IW (The user intended not all code should interwork)' <funarmthumb> function <sym>.`

The linker can diagnose where a non-interworking (`~IW`) function has its address taken by code in the other state. This error is disabled by default, but can be enabled by linking with `--strict`. The error can be downgraded to a warning with `--diag_warning 6801` and subsequently suppressed completely if required with `--diag_suppress 6801`

Where code, for example, in `a.c` uses the address of a non-interworking function in `t.c`:

```
armcc -c a.c
armcc --thumb -c t.c
armlink t.o a.o --strict
```

reports:

```
Error: L6801E: a.o(.text) containing ARM code cannot use the address of '~IW' Thumb function foo.
```

`L6802E: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <armsym>. Thumb Branch to non-Thumb symbol in <armobjname>(<armsecname>).`
`L6803W: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <armsym>. Thumb Branch is unlikely to reach target in<armobjname>(<armsym>).`
`L6804W: Legacy use of symbol type STT_TFUNC detected`
`L6805E: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <armsym>. Branch to untyped Absolute symbol in <armobjname>, target state unknown`
`L6806W: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <othersym>. Branch to untyped symbol in <otherobjname>(<othersecname>), ABI requires external code symbols to be of type STT_FUNC.`
`L6807E: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to <othersym>. Branch to untyped symbol in same section. State change is required.`
`L6809W: Relocation <rel_class>:<idx> in <objname>(<secname>) is of deprecated type <rtype>, please see ARMELF for ABI compliant alternative`
`L6810E: Relocation <rel_class>:<idx> in <objname>(<secname>) is of obsolete type <rtype>`

Relocation errors and warnings are most likely to occur if you are linking object files built with previous versions of the ARM tools.

To show relocation errors and warnings, use the `--strict_relocations` switch. This option enables you to ensure ABI compliance of objects. It is off by default, and deprecated and obsolete relocations are handled silently by the linker.

See the following in the *armlink User Guide*:

--strict_relocations, --no_strict_relocations.

`L6812U: Unknown symbol action type, please contact your supplier.`

`L6813U: Could not find Symbol <symname> to rename to <newname>.`

See the following in the *armlink User Guide*:

RENAME steering file command.

`L6815U: Out of memory. Allocation Size:<alloc_size> System Size:<system_size>.`

This error is reported by ARM Compiler v4.1 and later. It provides information about the amount of memory available and the amount of memory required to perform the link step.

This error occurs because the linker does not have enough memory to link your target object. This is not common, but might be triggered for a number of reasons, such as:

- Linking very large objects or libraries together.
- Generating a large amount of debug information.
- Having very large regions defined in your scatter file.

In these cases, your workstation might run out of virtual memory.

This issue might also occur if you use the `FIXED` scatter-loading attribute. The `FIXED` attribute forces an execution region to become a root region in ROM at a fixed address. The linker might have to add padding bytes between the end of the previous execution region and the `FIXED` region, to generate the ROM image. The linker might run out of memory if large amounts of padding are added when the address of the `FIXED` region is far away from the end of the execution region. The link step might succeed if the gap is reduced.

See the following in the *armlink User Guide*:

- *Execution region attributes*.
- *Root execution regions and the FIXED attribute*.

While the linker can generate images of almost any size, it requires a larger amount of memory to run and finish the link. Try the following solutions to improve link-time performance, to avoid the `Out of memory` error:

1. Shut down all non-essential applications and processes when you are linking.

   For example, if you are running under Eclipse, try running your linker from the command-line, or exiting and restarting Eclipse between builds.
2. Use the `--no_debug` linker option.

   This command tells the linker to create the object without including any debug information.

   See the following in the *armlink User Guide*:

   --debug, --no_debug.

   Note

   It is not possible to perform source level debugging if you use this option.
3. Reduce debug information.

   If you do not want to use the `--no_debug` option, there are other methods you can use to try to reduce debug information.

   You can also use the `fromelf` utility to strip debug information from objects and libraries that you do not have to debug.

   See the following in the *fromelf User Guide*:

   --strip=option[,option,...].
4. Use partial linking.

   You can use partial linking to split the link stage over a few smaller operations. Doing this also stops duplication of the object files in memory in the final link.

   See the following in the *armlink User Guide*:

   --partial.
5. Increase memory support on Windows operating systems.

On some Windows operating systems it is possible to increase the virtual address space from 2GB (the default) to 3GB.

For more information, see the following Microsoft article:

Memory Support and Windows Operating Systems.

6. Use the `--no_eager_load_debug` linker option.

   This option is available in RVCT 4.0 build 697 and later. It causes the linker to remove debug section data from memory after object loading. This lowers the peak memory usage of the linker at the expense of some linker performance, because much of the debug data has to be loaded again when the final image is written.

   See the following in the *armlink User Guide*:

   --eager_load_debug, --no_eager_load_debug.

If you are still experiencing the same problem, raise a support case.

```
L6828E: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to
<symname>, Branch source address <srcaddr> cannot reach next available pool at
[<pool_base>,<pool_limit>). Please use the --veneer_pool_size option to increase the
contingency.
```

The `--veneer_inject_type=pool` veneer generation model requires branches to veneers in the pool to be able to reach the pool limit, which is the highest possible address a veneer can use. If a branch is later found that cannot reach the pool limit, and `armlink` is able to fit all the veneers in the pool into the lower pool limit, then `armlink` reduces the pool limit to accommodate the branch. Error message L6828 is issued only if `armlink` is unable to lower the pool limit.

See the following in the *armlink User Guide*:

--veneer_inject_type=type.

```
L6898E: Relocation #<rel_class>:<idx> in <objname>(<secname>) with respect to
<armsym>. ARM branch to non-ARM/Thumb symbol in <armobjname>(<armsecname>).
L6899E: Existing SYMDEFS file '<filename> 'is read-only.
L6900E: Expected parentheses to specify priority between AND and OR operators.
L6901E: Expected symbol name.
L6902E: Expected a string.
L6903E: Cannot execute '<text>' in '<clause>' clause of script.
L6904E: Destination symbol of rename operation clashes with another rename.
L6905E: Source symbol of rename operation clashes with another rename.
L6906E: (This is the rename operation which it clashes with.)
L6907E: Expected an expression.
L6910E: Expected a phase name.
L6912W: Symbol <symname> defined at index <idx> in <oepname>(<secname>), has ABI
symbol type <symtype> which is inconsistent with mapping symbol type <maptype>.
L6913E: Expected execution region name.
L6914W: option <spurious> ignored when using --<memoption>.
```

`L6915E: Library reports error: <msg>`

The message is typically one of the following:

- ```
  Error: L6915E: Library reports error: scatter-load file declares no heap or stack
  regions and __user_initial_stackheap is not defined.
  ```

  or

  ```
  Error: L6915E: Library reports error: The semihosting __user_initial_stackheap
  cannot reliably set up a usable heap region if scatter loading is in use
  ```

  It is most likely that you have not re-implemented `__user_setup_stackheap()` or you have not defined `ARM_LIB_STACK` or `ARM_LIB_HEAP` regions in the respective scatter file.

  ——— **Note** ———

  `__user_setup_stackheap()` supersedes the deprecated function `__user_initial_stackheap()`.

  ———————————

  See the following in the *ARM C and C++ Libraries and Floating-Point Support User Guide*:
  — *__user_setup_stackheap()*.
  — *Legacy function __user_initial_stackheap()*.

  See the following in the *armlink User Guide*:

  Reserving an empty region.

- ```
  Error: L6915E: Library reports error: __use_no_semihosting was requested but
  <function> was referenced.
  ```

  Where `<function>` represents `__user_initial_stackheap`, `_sys_exit, _sys_open`, `_sys_tmpnam`, `_ttywrch, system`, `remove`, `rename`, `_sys_command_string`, `time`, or `clock`.

  This error can appear when retargeting semihosting-using functions, to avoid any `SVC` or `BKPT` instructions being linked-in from the C libraries.

  Ensure that no semihosting-using functions are linked in from the C library by using:

  ```
  #pragma import(__use_no_semihosting)
  ```

  See the following in the *ARM C and C++ Libraries and Floating-Point Support User Guide*:

  Using the libraries in a nonsemihosting environment.

  If there are still semihosting-using functions being linked in, the linker reports this error.

  To resolve this, you must provide your own implementations of these C library functions.

  The `emb_sw_dev` directory contains examples of how to re-implement some of the more common semihosting-using functions. See the file `retarget.c`.

  See the following for more information on semihosting-using C library functions:

  ARM C and C++ Libraries and Floating-Point Support User Guide.

  ——— **Note** ———

  The linker does not report any semihosting-using functions such as, for example, `__semihost()`, in your own application code.

  ———————————

  To identify which semihosting-using functions are still being linked-in from the C libraries:

  1. Link with `armlink --cpu=8-A.32 --verbose --list err.txt`
  2. Search `err.txt` for occurrences of `__I$use$semihosting`

     For example:

     ```
     ...
     Loading member sys_exit.o from c_4.l.
     ```

```
reference : __I$use$semihosting
definition: _sys_exit
...
```

This shows that the semihosting-using function `_sys_exit` is linked-in from the C library. To prevent this, you must provide your own implementation of this function.

- `Error: L6915E: Library reports error:__use_no_heap was requested, but <reason> was referenced`

  If `<reason>` represents `malloc`, `free`, `__heapstats`, or `__heapvalid`, the use of `__use_no_heap` conflicts with these functions.

- `Error: L6915E: Library reports error:__use_no_heap_region was requested, but <reason> was referenced`

  If `<reason>` represents `malloc`, `free`, `__heapstats`, `__heapvalid`, or `__argv_alloc`, the use of `__use_no_heap_region` conflicts with these functions.

```
L6916E: Relocation #<rel_class>:<idx> in <oepname>(<spname>). R_ARM_CALL for
conditional BL instruction).
L6917E: Relocation #<rel_class>:<idx> in <oepname>(<spname>). R_ARM_JUMP24 for BLX
instruction.
L6918W: Execution region <ername> placed at 0x<eraddr> needs padding to ensure
alignment <spalign> of <oepname>(<spname>).
L6922E: Section <objname>(<secname>) has mutually exclusive attributes (READONLY and
TLS)
L6923E: Relocation #<rel_class>:<idx> in <oepname>(<spname>) with respect to
<symname>. TLS relocation <type> to non-TLS symbol in <symobjname>(<symsecname>).
L6924E: Relocation #<rel_class>:<idx> in <oepname>(<spname>) with respect to
<symname>. Non-TLS relocation <type> to STT_TLS symbol in <symobjname>(<symsecname>).
L6925E: Ignoring <token> attribute for region <region>. MemAccess support has been
removed.
L6926E: Relocation #<rel_class>:<idx> in <oepname>(<spname>) has incorrect relocation
type <rtype> for instruction encoding 0x<bl>.
L6927E: Relocation #<rel_class>:<idx> in <oepname>(<spname>) has incorrect relocation
type <rtype> for instruction encoding 0x<bl1><bl2>.
L6932W: Library reports warning: <msg>
L6935E: Debug Group contents are not identical, <name> with signature sym <sig> from
objects (<new>) and (<old>)
L6936E: Multiple RESOLVE clauses in library script for symbol '<sym>'.
L6937E: Multiple definitions of library script function '<func>'.
L6939E: Missing alignment for region <regname>.
L6940E: Alignment <alignment> for region <regname> must be at least 4 and a power of
2 or MAX.
L6941W: chmod system call failed for file <filename> error <perr>
L6942E: Execution Region <ername> contains multiple <type>, sections:
L6966E: Alignment <alignment> for region <regname> cannot be negative.
L6967E: Entry point (<address>) points to a THUMB instruction but is not a valid
THUMB code pointer.
L6968E: Could not parse Linux Kernel version \"<kernel>\".
L6969W: Changing AT Section <name> type from RW to RO in <ername>.
```

`L6971E: <objname>(<secname>) type <type> incompatible with <prevobj>(<prevname>) type <prevtype> in er <ername>.`

You might see this message when placing `__at` sections with a scatter file. For example, the following code in `main.c` and the related scatter file gives this error:

```
int variable __attribute__((at(0x200000)));
```

```
LR1 0x0000 0x20000
{
    ER1 0x0 0x2000
    {
        *(+RO)
    }
    ER2 0x8000 0x2000
    {
        main.o
    }
    RAM 0x200000 (0x1FF00-0x2000)
    {
        *(+RW, +ZI)
    }
}
```

The variable has the type ZI, and the linker attempts to place it at address `0x200000`. However, this address is reserved for RW sections by the scatter file. This produces the error:

```
Error: L6971E: stdio_streams.o(.data) type RW incompatible with
main.o(.ARM.__AT_0x00200000) type ZI in er RAM.
```

To fix this, change the address in your source code, for example:

```
int variable __attribute__((at(0x210000)));
```

See the following in the *armlink User Guide*:

- *Methods of placing functions and data at specific addresses*.
- *Placement of sections at a specific address with __at*.

`L6972E: <objname>(<secname>) with required base 0x<required> has been assigned base address 0x<actual>.`

`L6973E: Error placing AT section at address 0x<addr> in overlay ER <ername>.`

For example, you attempted to use `__attribute__((at(`*address*`)))` to place a section when building a DLL or application with an overlay region. `__attribute__((at(`*address*`)))` requires that you specify a fixed location in a scatter file with `.ARM.__at_`*address*. In this case, you must also specify the `--no_autoat` linker option.

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __at*.
- *--autoat, --no_autoat*.

`L6974E: AT section <name> does not have a base address.`

See the following in the *armlink User Guide*:

Placement of sections at a specific address with __at.

`L6975E: <objname>(<secname>) cannot have a required base and SHF_MERGE.`
`L6976E: <objname>(<secname>) cannot have a required base and SHF_LINK_ORDER.`
`L6977E: <objname>(<secname>) cannot be part of a contiguous block of sections`
`L6978W: <objname>(<secname>) has a user defined section type and a required base address.`
`L6979E: <objname>(<secname>) with required base address cannot be placed in Position Independent ER <ername>.`
`L6980W: FIRST and LAST ignored for <objname>(<secname>) with required base address.`

See the following in the *armlink User Guide*:

Section placement with the FIRST and LAST attributes.

`L6981E: __AT incompatible with BPABI and SystemV Image types`

See the following in the *armlink User Guide*:

Restrictions on placing __at sections.

`L6982E: AT section <objname>(<spname>) with base <base> limit <limit> overlaps address range with AT section <obj2name>(<sp2name>) with base <base2> limit <limit2>.`

See the following in the *armlink User Guide*:

Placement of sections at a specific address with __at.

`L6983E: AT section <objname>(<spname>) with required base address <base> out of range for ER <ername> with base <erbase> and limit <erlimit>.`

This can occur if you specify `__attribute__((at(address)))` in your code, `.ARM.__at_address` in your scatter file, and `--no_autoat` option on the linker command line. In this case, the address part of `.ARM.__at_address` must be specified as eight hexadecimal digits. For example:

```
int x1 __attribute__((at(0x4000)));  // defined in function.c
; scatter file
LR1 0x0
{
    ...
        function.o(.ARM.__at_0x00004000)
    ...
}
```

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __at*.
- *--autoat, --no_autoat*.

`L6984E: AT section <objname>(<spname>) has required base address <base> which is not aligned to section alignment <alignment>.`

See the following in the *armlink User Guide*:

Placement of sections at a specific address with __at.

`L6985E: Unable to automatically place AT section <objname>(<spname>) with required base address <base>. Please manually place in the scatter file using the --no_autoat option.`

See the following in the *armlink User Guide*:

- *Placement of sections at a specific address with __at*.
- *--autoat, --no_autoat*.

`L9511E: Product definition file was not found`

`armlink` cannot find the required product license mapping (`.elmap`) files.

This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- `armlink` is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- `armlink` is looking for a non-existent `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 3
ELF Image Converter Errors and Warnings

Describes the error and warning messages for the ELF image converter, `fromelf`.

It contains the following sections:

- *3.1 List of the fromelf error and warning messages* on page 3-79.

3.1 List of the fromelf error and warning messages

Lists the error and warning messages that `fromelf` produces.

`Q0105E: Load region #<segindex> extends beyond top of address space.`
`Q0106E: Out of Memory.`
`Q0107E: Failed writing output file '<filename>': <reason>`
`Q0108E: Could not create output file '<filename>': <reason>`
`Q0119E: No output file specified.`
`Q0120E: No input file specified.`
`Q0122E: Could not open file '<filename>': <reason>`

If `<reason>` is `Invalid argument`, this might be because you have invalid characters on the command line.

For example, on Windows you might have used the escape character `\` when specifying a filter with an archive file:

```
fromelf --elf --strip=all t.a\(test*.o\) -o filtered/
```

On Windows, use:

```
fromelf --elf --strip=all t.a(test*.o) -o filtered/
```

See the following in the *fromelf User Guide*:

input_file.

`Q0128E: File i/o failure.`

This error can occur if you specify a directory for the `--output` command-line option, but you did not terminate the directory with a path separator. For example, `--output=my_elf_files/`.

See the following in the *fromelf User Guide*:

--output=destination.

`Q0129E: Not a 32 bit ELF file.`
`Q0130E: Not a 64 bit ELF file.`
`Q0131E: Invalid ELF identification number found.`

This error is given if you attempt to use fromelf on a file which is not in ELF format, or which is corrupted. Object (`.o`) files and executable (`.axf`) files are in ELF format.

`Q0132E: Invalid ELF section index found <idx>.`
`Q0133E: Invalid ELF segment index found <idx>.`
`Q0134E: Invalid ELF string table index found <idx>.`
`Q0135E: Invalid ELF section entry size found.`
`Q0136E: ELF Header contains invalid file type.`
`Q0137E: ELF Header contains invalid machine name.`
`Q0138E: ELF Header contains invalid version number.`

See Q0131E.

`Q0147E: Failed to create Directory <dir>: <reason>`

If `<reason>` is `File exists`, this might be because you have specified a directory that has the same name as a file that already exists. For example, if a file called `filtered` already exists, then the following command produces this error:

```
fromelf --elf --strip=all t.a(test*.o) -o filtered/
```

The path separator character `/` informs fromelf that `filtered` is a directory.

See the following in the *fromelf User Guide*:

--output=destination.

`Q0171E: Invalid st_name index into string table <idx>.`
See Q0131E.

`Q0172E: Invalid index into symbol table <idx>.`
See Q0131E.

`Q0186E: This option requires debugging information to be present`
The `--fieldoffsets` option requires the image to be built with dwarf debug tables.

`Q0425W: Incorrectly formed virtual function elimination header in file`
This might indicate a compiler fault. Contact your supplier.

`Q0426E: Error reading vtable information from file`
This might indicate a compiler fault. Contact your supplier.

`Q0427E: Error getting string for symbol in a vtable`
This might indicate a compiler fault. Contact your supplier.

`Q0433E: Diagnostic style <style> not recognised`

`Q0440E: No relocation sections for <secname>`

`Q0447W: Unknown Diagnostic number (<num>)`

`Q0448W: Read past the end of the compressed data while decompressing section '<secname>' #<secnum> in <file>`
This might indicate an internal fault. Contact your supplier.

`Q0449W: Write past the end of the uncompressed data buffer of size <bufsize> while decompressing section '<secname>' #<secnum> in <file>`
This might indicate an internal fault. Contact your supplier.

`Q0450W: Section '<secname>' #<secnum> in file <file> uses a mixture of legacy and current ABI relocation types.`

`Q0451W: Option '--strip symbols' used without '--strip debug' on an ELF file that has debug information.`

`Q0452W: Option '--strip filesymbols' used without '--strip debug' on an ELF file that has debug information.`

`Q0453W: Stripping path names from '<path1>' and '<path2>' produces a duplicate file name '<filename>'.`

`Q0454E: In ELF file: <details>`

`Q9511E: Product definition file was not found`
`fromelf` cannot find the required product license mapping (`.elmap`) files.
This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- `fromelf` is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- `fromelf` is looking for a non-existent `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 4
Librarian Errors and Warnings

Describes the error and warning messages for the ARM librarian, `armar`.

It contains the following sections:

- *4.1 List of the armar error and warning messages* on page 4-82.

4.1 List of the armar error and warning messages

Lists the error and warning messages that `armar` produces.

`L6800U: Out of memory`

`L6825E: Reading archive '<archive>' : <reason>`

`L6826E: '<archive>' not in archive format`

`L6827E: '<archive>': malformed symbol table`

`L6828E : '<archive>': malformed string table`

`L6829E: '<archive>': malformed archive (at offset <offset>)`

`L6830E: Writing archive '<archive>' : <reason>`

`L6831E: '<member>' not present in archive '<archive>'`

`L6832E: Archive '<archive>' not found : <reason>`

`L6833E: File '<filename>' does not exist`

`L6835E: Reading file '<filename>' : <reason>`

`L6836E: '<filename>' already exists, so will not be extracted`

`L6838E: No archive specified`

`L6839E: One of the actions -[<actions>] must be specified`

`L6840E: Only one action option may be specified`

`L6841E: Position '<position>' not found`

`L6842E: Filename '<filename>' too long for file system`

`L6843E: Writing file '<filename>' : <reason>`

`L6874W: Minor variants of archive member '<member>' include no base variant`

Minor variants of the same function exist within a library. Find the two equivalent objects and remove one of them.

`L6875W: Adding non-ELF object '<filename>' to archive '<name>'`

`L9511E: Product definition file was not found`

`armar` cannot find the required product license mapping (`.elmap`) files.

This might be because:

- The `.elmap` files are missing, for example if your installation is corrupt.
- `armar` is looking for the `.elmap` files in the wrong place because the `ARM_PRODUCT_PATH` environment variable is incorrectly set.
- `armar` is looking for a non-existent `.elmap` file because the `ARM_TOOL_VARIANT` environment variable is incorrectly set.

Chapter 5
Other Errors and Warnings

Describes error and warning messages that might be displayed by any of the tools.

It contains the following sections:

- *5.1 Internal faults and other unexpected failures* on page 5-84.
- *5.2 List of other error and warning messages* on page 5-85.

5.1 Internal faults and other unexpected failures

Internal faults indicate that the tool has failed an internal consistency check or has encountered some unexpected input that it could not deal with. They might point to a potential issue in the tool itself.

For example:

```
Internal fault: [0x76fd03:600448]
```

contains:

- The message description (`Internal fault`).
- A six hex digit fault code for the error that occurred (`0x76fd03`).
- The version number (60 is ARM Compiler 6.0).
- The build number (0448 in this example).

If you see an internal fault, contact your supplier.

To facilitate the investigation, try to send only the single source file or function that is causing the error, plus the command-line options used.

If the internal fault is caused by the assembler, and the source code contains pre-processor macros, it might be necessary to use the compiler to preprocess the source before sending it to your supplier.

5.2 List of other error and warning messages

A list of the error and warning messages that any of the tools in the ARM Compiler toolchain produce.

Note

When the message is displayed, the *X* prefixing the message number is replaced by an appropriate letter relating to the tool. For example, the code `X3900U` is displayed as `L3900U` by the linker when you have specified an unrecognized option.

`X3900U: Unrecognized option '<dashes><option>'.`
: `<option>` is not recognized by the tool. This could be because of a spelling error or the use of an unsupported abbreviation of an option.

`X3901U: Missing argument for option '<option>'.`

`X3902U: Recursive via file inclusion depth of <limit> reached in file '<file>'.`

`X3903U: Argument '<argument>' not permitted for option '<option>'.`
: Possible reasons include malformed integers or unknown arguments.

`X3904U: Could not open via file '<file>'.`

`X3905U: Error when reading from via file '<file>'.`

`X3906U: Malformed via file '<file>'.`

`X3907U: Via file '<file>' command too long for buffer.`

`X3908U: Overflow: '<string>' will not fit in an integer.`

`X3910W: Old syntax, please use '<hyphens><option><separator><parameter>'.`

`X3912W: Option '<option>' is deprecated.`

`X3913W: Could not close via file '<file>'.`

`X3915W: Argument '<argument>' to option '<option>' is deprecated`

`X3916U: Unexpected argument for option '<dashes><option>'`

`X3917U: Concatenated options cannot have arguments: -<option> <arg>`

`X9905E: cannot use --apcs=/hardfp without floating point hardware`

`X9906E: cannot use --apcs=/hardfp with fpu <fpu_option>`

`X9907E: unable to select no floating point support`

`X9908E: --fpmode=none overrides --fpu choice`