2.1. List of the armasm error and warning messages

This section lists the error and warnings for armasm.

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 *Unary operators* in the *RVCT Assembler Guide*.

A1020E

`Bad predefine: <directive>`

The operand to the `--predefine` (`-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 upon 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>`.

A1143E

`COMMON directive not supported for %s format output`

A1144E

`DCDO directive not supported for %s format output`

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 <cpu> command line option was not a recognized processor name. Check the spelling of the argument.

Use `--cpu=list` to list the supported CPUs.

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 because the given name is not valid, there is no space, a read-only file with the same name already exists, or the file is in use by another process. Check that the correct path for the file is specified.

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` or `--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.

A1075E

`The specified errors file '<filename>' must not be a source file`

The filename argument to the`--depend` or `--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.

A1085E

`Forced user-mode LDM/STM must not be followed by use of banked R8-R14`

The ARM architecture does not allow you to access the banked registers on the instruction following a USER registers LDM or STM. The *ARM Architecture Reference Manual* says this form of LDM must not be followed by an instruction, which accesses banked registers (a following NOP is a good way to ensure this)

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 given when no `AREA` is given (see A1105E).

A1099E

`Structure stack overflow max stack size <max>`

A1100E

`Structure stack underflow`

A1105E

`Area directive missing`

This is given when no `AREA` is given (see also 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 *Numeric expressions* in the *RVCT Assembler Guide*.

A1111E

`Expected constant or address expression`

A1112E

`Expected address expression`

A1113E

`Expected string expression`

A string expression was expected after, for example, `SETS`. See *String expressions* in the *RVCT Assembler Guide*.

A1114E

`Expected register relative expression`

For example, the generic form:

```
LDR r4,[r9,offset]
```

must be rewritten as:

```
LDR r4,[r9,#offset]
```

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 *Using macros* in the *RVCT Assembler Guide*.

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
```

Can be changed 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 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`.

These subtractive relocations were allowed with SDT AOF, but not with ELF, so this error message can sometimes appear when migrating an SDT project to RVCT. To resolve this 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.

Another example that shows the error is:

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

A1145E

```
Undefined exported symbol '<sym>'
```

A1146E

```
Unable to open output file <codeFileName>: <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
  ```

  typically requires a definitions file to be included, 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||
  ```

typically requires the symbol to be imported, 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 labelled 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 local label is a number in the range 0-99, optionally followed by a name.

See *Local labels* in the *RVCT Assembler Guide*.

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 some white space on the left hand side margin, before the instruction, for example change:

  ```
  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 *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 if a MOV or MVN instruction is used with a constant that cannot be assembled.

See *Direct loading with MOV and MVN* in the *RVCT Assembler Guide*.

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 section. 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<mini> to 0x<maxi>
```

A1175E

```
Bad register range
```

A1176E

```
Branch offset 0x<val> out of range.  Permitted values are 0x<mini> to 0x<maxi>
```

Branches are PC relative, and have a limited range. If you are using "local labels", you can use the ROUT directive to limit the scope of local labels, to help avoid referring to a wrong label by accident.

See *Local labels* in the *RVCT Assembler Guide*.

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

A1187E

Error in macro parameters

A1188E

Register value <val> out of range. Permitted values are <mini> to <maxi>

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 miss-typed.

For example:

```
armasm init.s -g -PD "ROM_RAM_REMAP SETL {FALS}"
```

must be:

```
armasm init.s -g -PD "ROM_RAM_REMAP SETL {FALSE}"
```

See *Assembly time substitution of variables* in the *RVCT Assembler Guide*.

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 *Assembly time substitution of variables* in the *RVCT Assembler Guide*.

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

Example:

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

The HALFWORD and INTERWORK attributes are obsolete. Remove them.

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

A1247E

BLX from ARM code to ARM code, use BL

This occurs when there is a BLX <label> branch from ARM code to ARM code within this assembler file. This is not allowed because BLX <label> always results in a state change. 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 Thumb code to Thumb code within this assembler file. This is not allowed because BLX <label> always results in a state change. 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

Example:

```
LDRH R3, =constant
```

Change the LDRH into LDR, which is the standard way of loading constants into registers.

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, such as:

```
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 Thumb code, the literal pool must be within 1KB of the LDR instruction to access it. See A1284E and A1471W.

A1284E

`Literal pool too distant, use LTORG to assemble it within 4KB`

For ARM code, the literal pool must be within 4KB of the LDR instruction to access it. To solve this, add an `LTORG` directive into your assembler source file at a convenient place.

See *Loading with LDR Rd, =const* and *LTORG* in the *RVCT Assembler Guide*.

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`

The macro definition is invalid.

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

A1323E

Reserved instruction (Rm = Rn with post-indexing)

A1324E

Undefined effect (PC + writeback)

A1327W

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

LDM Operand restriction:

- If the base register <Rn> is specified in <registers>, and base register writeback is specified, the final value of <Rn> is UNPREDICTABLE.

A1328W

Non portable instruction (STM with writeback and base not first in register list, stored value of base unprec

STM Operand restrictions if <Rn> is specified as <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 UNPREDICTABLE.

A1329W

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. The *ARM Architecture Reference Manual* specifies that 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 A1085W

A1331W

Unpredictable instruction (PC as source or destination)

A1332W

Unpredictable effect (PC-relative SWP)

A1334E

Undefined effect (use of PC/PSR)

A1335W

Useless instruction (PC cannot be written back)

A1337W

Useless instruction (PC is destination)

A1338W

Dubious instruction (PC used as an operand)

A1339W

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 instructi

A1344I

host error: out of memory

A1355U

A Label was found which was in no AREA

Example:

This can occur where no white-space precedes an assembler directive.

Assembler directives must be indented with white-space, for example use:

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

instead of:

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

Symbols in the left hand column one are assumed to be labels, hence the error message.

A1356W

`Instruction not supported on targeted CPU`

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

For example:

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

can be assembled with:

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

The processor selected on the armasm command line does not support this instruction. See the *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 VFP instructions are malformed.

See *NEON and VFP Programming* in the *RVCT Assembler Guide*.

A1430E

`Unknown frame directive`

A1431E

`Frame directives are not accepted outside of PROCs/FUNCTIONs`

Invalid FRAME directive.

See *Frame directives* in the *RVCT Assembler Guide*.

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 desired behavior.

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`

Example:

```
AREA test1, CODE, READONLY
AREA test, CODE, READONLY, INTERWORK
```

This code might have originally been intended to work with SDT. The `INTERWORK` area attribute is now obsolete. To eliminate the warning:

- 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 statement not being correctly indented or missing.

A1448W

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

A1449W

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

A1450W

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

The ARM assembler (armasm) supports the full range of MRS and MSR instructions, in the form:

```
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`.

Note

`MSR CPSR_c, #immediate` is a legitimate instruction (despite what is written in early versions of the *ARM Architecture Reference Manual*), so a sequence of two instructions like:

```
MOV r0, #0x1F
MSR CPSR_c, r0
```

as commonly found in boot code, can be combined into one instruction, such as:

```
MSR CPSR_c, #0x1F ; go to System mode, IRQ & FIQ enabled
```

Earlier releases of the assembler allowed 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 now 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 simply modify your code to use `_c`, `_f`, `_cf` or `_cxsf` instead.

For more information, see:

- *Instruction capabilities* in the *RVCT Assembler Guide*.
- the FAQ *armasm: use of MRD and MSR instructions ('Deprecated form of PSR field specifier')*.

A1454E

```
FRAME STATE RESTORE directive without a corresponding FRAME STATE REMEMBER
```

Invalid `FRAME` directive.

See *Frame directives* in the *RVCT Assembler Guide*.

A1456W

```
INTERWORK area directive is obsolete. Continuing as if --apcs /inter selected
```

Example:

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

This code might have originally been intended to work with SDT. The `INTERWORK` area attribute is now obsolete. To eliminate the warning:

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

A1457E

```
Cannot mix INTERWORK and NOINTERWORK code areas in same file
```

`INTERWORK` and (default) `NOINTERWORK` code areas cannot be mixed in same file. This code might have originally been intended to work with SDT. The `INTERWORK` area attribute is obsolete in RVCT.

Example:

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

To eliminate the error:

1. move the two AREAs into separate assembler files such as, for example, `test1.s` and `test2.s`
2. remove the `", 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.

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 allowed. See the *ARM Architecture Reference Manual* for the allowed forms.

A1461E

`Specified processor or architecture does not support Thumb instructions`

It is likely that you are specifying a specific architecture or cpu using the `--cpu` option and then incorporating some Thumb code in the AREA that is generating this error.

For example:

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

StrongARM is an architecture 4 (not 4T) processor and does not support Thumb code.

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 assembler.

To avoid the warning, either:

- modify the code to make the evaluation order explicit (that is, add more brackets)
- suppress the warning with `--unsafe` switch.

See *Operator precedence* in the *RVCT Assembler Guide*.

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`

Invalid FRAME directive.

See *Frame directives* in the *RVCT Assembler Guide*.

A1471W

`Directive <directive> may be in an executable position`

This can occur with, for example, the LTORG directive (see A1283E & A1284E). LTORG instructs the assembler to dump literal pool DCD data at this position.

To prevent this warning from occurring, 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
```

A1475W

`At least one register must be transferred, otherwise result is UNPREDICTABLE`

A1476W

`BX r15 at non word-aligned address is UNPREDICTABLE`

A1477W

`This register combination results in UNPREDICTABLE behavior`

A1479W

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

This is warning about an `ALIGN` directive which has a coarser alignment boundary than its containing `AREA`, which is not allowed. 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, hence the error.

> **Note**
>
> The two alignment types are specified in different ways.

See *ALIGN* and *AREA* in the *RVCT Assembler Guide*.

A1480W

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

A1481E

`Object file format does not support this area alignment`

This can occur when using `AREA ... ALIGN=0` to align a code section on a byte boundary, which is not possible. Code sections can only be aligned on four-byte boundary for ARM code, and two-byte boundary for Thumb code. Use `"ALIGN=2"` instead for ARM code, or `"ALIGN=1"` for Thumb code.

A1482E

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

A1484E

`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, and LSL.

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

Earlier versions of the assembler would silently convert ASL to LSL. This error can be downgraded to a warning by using the `--unsafe` switch.

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.

A1487E

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

The Thumb instruction ASL is now faulted. See the corresponding ARM ASL message A1484E.

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<mini> to 0x<maxi>`

A1493E

`REQUIRE must be in an AREA`

A1495E

`Target of branch is a data address`

RVCT 2.2 and later are able to determine the type of a symbol and detect branches to data. This warning can be suppressed with `--diag-suppress 1495`

A1496E

`Absolute relocation of ROPI address with respect to symbol '<symbol>' at offset <offset> may cause link failu`

For example, when assembling with `--apcs /ropi`:

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

because this generates an absolute relocation (`R_ARM_ABS32`) to a PI code symbol.

A1497E

`Absolute relocation of RWPI address with respect to symbol '<symbol>' at offset <offset> may cause link failu`

For example, when assembling with `--apcs /rwpi`:

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

because this generates an absolute relocation (`R_ARM_ABS32`) to a PI data symbol.

A1498E

`Unexpected characters following Thumb instruction`

For example:

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

is accepted as a valid instruction, for both ARM and Thumb, but:

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

is a valid instruction for ARM, but not for Thumb, so the unexpected characters are `", 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 <mini> to <maxi>`

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

A1544W

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

A1545E

Too many sections for one <objfmt> file

A1546W

Stack pointer update potentially breaks 8 byte stack alignment

Example:

```
PUSH {r0}
```

The stack must be eight-byte aligned on an external boundary so pushing an odd number of registers causes this warning to be given. This warning is suppressed by default. To enable this warning use `--diag_warning 1546`.

See the *RVCT Assembler Guide* for more information.

A1547W

PRESERVE8 directive has automatically been set

Example:

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

This warning has been given because the PRESERVE8 directive has not been explicitly set by the user, but the assembler has set this itself automatically. This warning is suppressed by default. To enable this warning use `--diag_warning 1547`.

See the *RVCT Assembler Guide* for more information.

A1548W

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

Example:

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

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

A1549W

Setting of REQUIRE8 but not PRESERVE8 is unusual

Example:

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

A1550E

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.

This can be enabled with `armasm --diag_warning 1563`

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

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>`

The assembler warns by default when padding bytes are added to the generated code. This occurs whenever an instruction/directive is used at an address that requires a higher alignment, for example, to ensure ARM instructions start on a four-byte boundary after some Thumb instructions, or where there is a DCB followed by DCD.

For example:

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

Results in the 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 Thumb-only code. The ADR Thumb pseudo-instruction can only load addresses that are word aligned, but a label within Thumb code might not be word aligned. Use `ALIGN` to ensure four-byte alignment of an address within Thumb code.

A1582E

`Link Order area '<name>' undefined`

A1583E

`Group symbol '<name>' undefined`

A1584W

`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

A1603W

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 as bit0 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, <rcvd>, expected one of <expect>

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 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 *RVCT Assembler Guide* for more information about the operands permitted for specific 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
```

A1624E

```
Branch from Thumb code to ARM code
```

A1625E

```
Branch from ARM code to Thumb code
```

A1626E

```
BL from Thumb code to ARM code
```

A1627E

```
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
```

Certain processors such as Cortex-M3 or 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 warn when it substitutes an instruction when assembling.

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 Thumb-2, unpredictable single register LDMs are transformed into LDRs.

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

For example:

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

When assembled with:

```
armasm --diag_warning 1645
```

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
```

A1646W

`VMOV pseudo-instruction for a register to register move is deprecated. Please use a VORR instruction instead`

This message relates to Wireless MMX.

A1647E

`Bad register name symbol, expected Integer register`

This message relates to Wireless MMX.

A1648E

`Bad register name symbol, expected Wireless MMX SIMD register`

This message relates to Wireless MMX.

A1649E

`Bad register name symbol, expected Wireless MMX Status/Control or General Purpose register`

This message relates to Wireless MMX.

A1650E

`Bad register name symbol, expected any Wireless MMX register`

This message relates to Wireless MMX.

A1651E

`TANDC, TEXTRC and TORC instructions with destination register other than R15 is undefined`

This message relates to Wireless MMX.

A1652W

`FLDMX/FSTMX instructions are deprecated in ARMv6.  Please use FLDMD/FSTMD instructions to save and restore ur`

A1653E

`Shift instruction using a status or control register is undefined`

A1654E

`Cannot access external symbols when loading/storing bytes or halfwords`

A1655W

`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`

The option passed to armasm is now deprecated. Use `armasm --help` to view the currently available options, or refer to the *RVCT Assembler Guide*.

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

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 allowed in the specified instruction set. The instruction moveq, for example, is only allowed in ARM and Thumb-2 assembler, but not Thumb-1.

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

A1705E
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

A1712W
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

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

A1723E
__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

A1726E
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

A1746W

Instruction stall diagnostics may be unreliable for this CPU

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 Thumb-2 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 a IT block to allow a number of conditional instructions in Thumb-2. 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>

A1779W

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 in ARMv7

This is caused by statements like:

```
ADD sp, r0, #imm
```

This can be replaced with a sequence like:

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

For more information, see *Diagnostic messages A1745W, A1477W and A1786W*.

A1787W

Use of VFP Vector Mode is deprecated in ARMv7

A1788W

Explicit use of PC in this instruction is deprecated

A1789W

Explicit use of PC in this instruction is deprecated, 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

A1801W

Instruction is UNPREDICTABLE in current instruction set

A1803E

Bad system instruction name

A1804E

Bad CP14 or CP15 register name for instruction

A1805W

Register is Read-Only

A1806W

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

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>

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