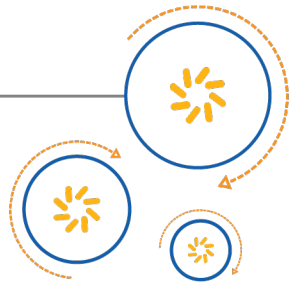
Qualcomm Technologies, Inc.



Qualcomm Snapdragon LLVM ARM Utilities

User Guide

80-VB419-103 Rev. A

October 7, 2016

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Contents

Tables

1 Introduction

1.1 Purpose

The Snapdragon LLVM ARM utilities are a set of software tools which create and manage object code. They are used with compilers, debuggers, and profilers to support software development for the Qualcomm® Snapdragon™ processor.

The utilities include the following tools:

- Assembler
- Linker
- Archiver
- Object file symbol lister
- Object file copier
- Object file viewer
- Archive indexer
- Object file section size lister
- Object file string lister
- Object file stripper
- C++ filter
- Address converter
- ELF file viewer

While the Snapdragon LLVM ARM utilities are designed to be feature-compatible with the GNU binutils, a few differences exist. These differences are highlighted in this document.

1.2 Conventions

Square brackets enclose optional items (e.g., `[label]`).

The vertical bar character | is used to indicate a choice of items (e.g., `add`|`del`).

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, <*number*>.

Commands to be entered appear in a different font, for example, `copy a:*.* b:`.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 Functional overview

Table 2-1 lists the Snapdragon LLVM ARM utilities.

Table 2-1 Snapdragon LLVM ARM utilities

Utility	Command	Description
Assembler	`clang`	Translate assembly source files into object files
Linker	`arm-link (a.k.a ld.qcld)`	Combine object files into executable
Archiver	`arm-ar`	Create, modify, and extract files from archives
Object file symbols	`arm-nm`	List symbols from object files
Object file copier	`arm-elfcopy`	Copy and translate object files
Object file viewer	`llvm-objdump`	Display contents of object files
Archive indexer	`arm-ranlib`	Generate index to archive contents
Object file size	`arm-size`	List section sizes and total size
Object file strings	`arm-strings`	List strings from object files
Object file stripper	`arm-strip`	Remove symbols from object file
C++ filter	`arm-c++filt`	Filter to demangle encoded C++ symbols
Address converter	`arm-addr2line`	Convert addresses to file and line
ELF file viewer	`arm-readelf`	Display contents of ELF format files

2.1 Using the utilities

The ARM utilities are virtually identical to the corresponding ARM GNU Binutils. Except for the `arm` prefix, their command names are mostly identical, and they support many of the GNU command options.

2.2 System requirements

The ARM utilities are part of the software development tools for the Snapdragon ARM processor, which run on the Windows® and Linux® operating systems.

2.3 Acknowledgements

This document includes content derived from the LLVM Project (under the terms of the LLVM Release License) and the ELF Tool Chain Project (under the terms of the BSD License).

This section presents the following license statements:

- LLVM Release License
- BSD License

LLVM Release License

The description of the Snapdragon LLVM ARM object file viewer utility is subject to the conditions and disclaimers defined in the LLVM Release License:

llvm.org/releases/3.6.0/LICENSE.TXT

BSD License

The descriptions of the Snapdragon LLVM ARM utilities listed below are subject to the following conditions and disclaimers.

Archiver Copyright (c) 2007, 2009-2012, Joseph Koshy

Object file symbols Copyright (c) 2007, Hyogeol Lee

Object file copier Copyright (c) 2008-2009, 2011, Joseph Koshy

Archive indexer Copyright (c) 2007,2009-2012, Joseph Koshy

Object file size Copyright (c) 2007, S.Sam Arun Raj

Copyright (c) 2008, 2011, Joseph Koshy

Object file strings Copyright (c) 2007, S.Sam Arun Raj

Object file stripper Copyright (c) 2011, Joseph Koshy

C++ filter Copyright (c) 2011, Kai Wang

Address converter Copyright (c) 2009,2010, Joseph Koshy

ELF file viewer Copyright (c) 2009,2011, Joseph Koshy

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

3 Assembler

The assembler translates ARM assembly language into object code. Object files are stored in ELF format.

The assembler is mostly compatible with the syntax and directives supported in the GNU assembler, and supports all features such as macros, conditional text, and include files.

The Snapdragon LLVM ARM assembler uses the same driver (`clang`) as the Snapdragon LLVM ARM compiler. Please refer the Snapdragon LLVM ARM User Guide for compiling assembly files. The Snapdragon LLVM ARM assembler uses the same syntax as GNU assembler and supports all GNU directives

Command

To start the assembler from a command line, type:

`clang –target <triple> [option...]`

Options

The assembler supports the most commonly-used options in the GNU assembler. Option names can be truncated as long as they uniquely identify the option.

```
-g
-help
-I=pathname
-o filename
```

3.1 Assembler information

`-help`

List common assembler options.

Input and output files

`-o filename`

Output file name.

Assembly translation

`-I=pathname`

Pathname used to search for assembly include files (default current directory).

Code generation

`-g`

Generate DWARF debug information for assembly source files.

Target

`-triple=`*`string`*

Target triple. To list the possible values, run the assembler with the –version option.

3.2 ARM assembly language basic syntax

3.2.1 Symbols

A *symbol* consists of the following characters:

- `A` ... `Z`, `a` ... `z`
- `0` ... `9`
- `_` , `.` , or `$`

Symbols must begin with a letter, and are case-sensitive.

3.2.2 Whitespace

Whitespace is defined as one or more blanks or tab characters. It is used in assembly code to separate symbols. The assembler interprets whitespace as a single space character.

NOTE Whitespace does not appear in character constants.

3.2.3 Keywords

Keywords are symbols that are reserved by the assembler and cannot be redefined in the assembly code.

Registers, assembly language instructions, and assembly directive parameters are all defined as keywords.

3.2.4 Directives

Assembler directives are commands that are embedded in the assembly code. They are used to declare items such as constants and external symbols, and to control assembler features such as macros and include files.

Directives always begin with a period character (.) immediately followed by the directive name and (optionally) one or more parameters. The names are case-sensitive. The parameters are delimited by whitespace.

For example:

```
.set  const_val,  1
.include "myfile.s"
```

RELATED INFORMATION

"Directives" on page 22

3.2.5 Numbers

Numbers can be expressed in several numeric formats. The formats are specified with the number's prefix character.

Format	Prefix	Example
Binary	0b or 0B	`0b1111`
Octal	0	`077`
Decimal	non-0 digit	`1028`
Hexadecimal	0x or 0X	`0Xffff`
Floating point	0e or 0E	`0e3.14159E-3`

Floating point numbers can include the following elements:

- A sign character (+ or -)
- An integer part consisting of zero or more decimal digits
- A fractional part consisting of '.' followed by zero or more decimal digits
- An exponent part consisting of the following elements:
  - The prefix `E` or `e`
  - A sign character (+ or -)
  - One or more decimal digits

An integer or fractional part is required, while all the other elements are optional.

3.2.6 Characters

Characters consist of a left quote immediately followed by that character. For example:

```
'a
```

String constants are delimited by double-quote characters. For example:

```
"mydata"
```

Special characters can be used in characters and strings by prefixing them with a backslash character. For example:

```
'\\
```

This example represents a single backslash character – the first backslash acts as the prefix character for the second.

NOTE When used in a numeric expression, the value of a character constant is its 8-bit ASCII code value.

Table 3-1 lists the escape characters that can be used in characters and strings.

Table 3-1 Escape characters

Code	Character
`\b`	Backspace
`\f`	Form feed
`\n`	Newline
`\r`	Return
`\t`	Tab
`\\`	Backslash
`\"`	Double quote
`\NNN`	Octal character value. NNN specifies 3 octal digits.
`\xhex...`	Hex character value. hex... specifies N hexadecimal digits.

3.2.7 Expressions

Expressions consist of one or more symbols (representing numeric or address values) which are operated on by one or more operators.

Expressions can be used wherever numeric or address values are allowed, including variables and instructions.

NOTE Expressions must evaluate to either an absolute value or an offset into a section. Otherwise, they will be flagged with an error message.

RELATED INFORMATION

"Sections" on page 18

3.2.8 Operators

Operators are characters which represent arithmetic functions such as addition or multiplication.

Prefix operators accept a single operand, which must be an absolute value. The result is absolute.

Operator	Description
-	Two's complement arithmetic negate
~	Bitwise logical NOT

Infix operators accept two operands: + and -

Either operator can operate on relocatable values, while for all the other infix operators, the operand values must be absolute, and the result is absolute.

Table 3-2 Infix operators

Operator	Description	Precedence
*	Multiply	High
/	Divide	
%	Remainder	
< <<	Shift left	
>> >>	Shift right	
|	Bitwise inclusive OR	Intermediate
&	Bitwise AND	
^	Bitwise exclusive OR	
!	Bitwise OR NOTE	
-	Two's complement arithmetic negate	Low
~	Bitwise logical NOT	

Infix operators have differing precedence – operators with the highest precedence are evaluated first in an expression, while operators with equal precedence are evaluated left to right in an expression.

3.2.9 Relocatable operands

If an absolute operand is added to a relocatable operand, the result belongs to the section of the relocatable operand. Two relocatable operands cannot be added if they belong to different sections.

If an absolute operand is subtracted from a relocatable operand, the result belongs to the section of the relocatable operand. If both operands are relocatable and in the same section, the result of a

subtraction is absolute. Two relocatable operands cannot be subtracted if they belong to different sections.

3.2.10 Statements

A statement is normally considered to be equivalent to one line in the assembly source code. It is terminated with a newline character (`\n`).

A single statement can span multiple lines in the assembly source code if a backslash character (`\`) is placed immediately in front of each line-terminating newline character.

NOTE Every statement in an assembly source file (including the last statement in the file) must be terminated with a newline.

3.2.11 Comments

Delimited comments begin with the character sequence `/*` and end with `*/`. They can span multiple lines, and cannot be nested. For example:

```
/*  delimited  comment  */
```

The presence of `#` as the first character of a line makes the whole line as a comment.

For ARM, the presence of a `@` on a line indicates the start of a comment that extends to the end of that line.

For AArch64, the presence of `//` on a line indicates the start of a comment that extends to the end of that line.

3.3 Sections

Assembly language programs are organized into collections of units known as sections. For example:

```
.section        .text
// ARM instructions
.section        .data
// variable declarations
```

Each section is defined to occupy a range of consecutive addresses. When the linker merges object files into a single executable image, it relocates the starting address of each section in the object files so the sections do not overlap in memory.

Table 3-3 lists the sections that can be found in an assembly program.

Table 3-3 Assembly sections

Section name	Description
`.test`	Program code and data. .rodata is used to stre read-only data.
`.data`	

Table 3-3 Assembly sections (Continued)

Section name	Description
.rodata	
.bss	Uninitialized variables or common data. .bss does not occupy any space in an object file, because the section data is always initialized to zero when the section is loaded into memory.

The linker normally assigns all sections of the same type to occupy contiguous addresses in memory. This can be changed by creating a linker script.

NOTE The system-specific memory sections must be assigned to specific memory areas – this is done using the linker.

Code and data cannot be stored together in the same system-specific section.

User-defined sections

User-defined sections are defined when a program needs to store code or data in a non- standard configuration. They are defined by specifying section names that differ from the predefined sections. For example:

```
.section my_strings
```

Note that a user-defined section declaration does not specify a section type (code, data, etc.) – the assembler automatically determines the section type from the type of objects stored in the section.

To make the program more readable, it is common practice to prefix the name of a user- defined section with the section type. For example:

```
.section .rodata.my_strings
```

3.4 Symbols

This section describes how symbols are used in the ARM assembly language.

3.4.1 Symbol assignment

Symbols are assigned values with the .set directive. For example:

```
.set  my_symbol,  78
```

3.4.2 Temporary symbols

Temporary symbols work like regular symbols in the program source code, but are normally not saved in the object file, and are thus not visible when linking, debugging, or profiling.

Temporary symbols are prefixed with `.L`. For example:

```
.L_temp_label
```

The -L option (causes the assembler to save temporary symbols in the object file as local symbols. If the linker is also directed to save local symbols, the resulting temporary symbols can be used in debugging and profiling.

NOTE Temporary symbols differ from ELF local and global symbol types.

3.4.3 Labels

Labels are symbols immediately followed by a colon. For example:

```
my_label:
```

NOTE Do not use labels in embedded assembly functions, as they can cause problems when profiling. Instead, use local labels.

3.4.4 Local labels

Local labels can be specified as either symbolic labels or numeric labels.

Symbolic local labels are declared and used like regular labels, but specify a temporary symbol as the label symbol. For example:

```
.L_loc:
```

Numeric local labels consist of an integer immediately followed by a colon. For example:

```
99:
```

Numeric local labels can re-use the same set of numeric label names (`0:`, `1:`, `2:`, ...) throughout a program.

To reference the preceding definition of a numeric label, write `N`b (where `N` specifies the integer in the label). To reference a subsequent label definition, write `N`f. For example:

```
99:  nop;
99b;    //  jump  to  preceding  local  label  99 99:  nop;
```

A program can reference only the immediately preceding and following instances of a numeric label symbol.

NOTE Symbolic local label names can be saved in an object file.

RELATED INFORMATION

"Temporary symbols" on page 19

3.4.5 Dot symbol

When used as a symbol, the period character specifies the current address that the assembler is generating code for. For example:

```
my_var: .word    .
```

The period following the `.word` directive specifies the value of the data allocated by `.word`. The value is the memory address of the data.

RELATED INFORMATION

".word" on page 35

3.4.6 Symbol attributes

```
Value
```

Symbol values are generally 32-bit numbers. The values may be explicitly assigned in the program code, or implicitly assigned by the assembler.

The value of a data symbol is the section-relative offset of the data item. This value is relocatable, and is resolved by the linker.

The value of an undefined symbol is 0. This value is resolved by the linker.

The value of a common symbol is the number of bytes to reserve for the symbol in common memory. This value is resolved by the linker.

`Type`

The symbol type indicates whether a symbol identifies a function or object. It is set with

`.stype or .stabs.`
`Size`

The symbol size indicates the size (in bytes) of the item identified by the symbol. It is set with `.size` or `.stabs.`

`Scope`

The symbol scope indicates whether a symbol is local or global. The scope can be made global with `.global`.

`Visibility`

The symbol visibility indicates whether a symbol is protected, hidden, or internal.

Protected symbols always resolve to symbol definitions in the current component, even if definitions exist in other components which might normally supersede the local definition.

Hidden symbols are protected symbols that are not visible in other components. Internal symbols are hidden symbols that require extra processing.

The visibility is set with `.protected`, `.hidden`, or `.internal`.

3.4.7 Descriptor

The symbol descriptor contains an arbitrary 16-bit value. It is set with `.stabs`.

3.4.8 Symbol modifiers

Symbol modifiers are used to transform the value of the referenced symbol. The following modifers are supportedin the format `symbol@modifier`:

- `GOT`
- `GOTOFF`
- `GOTTPOFF`
- `GOT_PREL`

`#:lower16:` and `#:upper16` can be used as prefix for the values of `movw` and `movt`, respectively. For example, to load the 32-bit value of `foo` to `r0` enter:

```
MOVW r0, #:lower16:foo
MOVT r0, #:upper16:foo
```

3.5 Directives

Assembler directives are used to declare sections, constants, data, and symbols, and control assembler features such as macros and include files. Directives are prefixed with a period character (.) and may be followed by one or more arguments.

The assembler recognizes the following directives.

Sections

```
.section name
.section name [,"flags" [,type]]
.text [subsection]
.data [subsection]
```

Symbols

```
.equ symbol,  expr
.set symbol, expr
.equiv symbol,  expr
.global symbol
.globl symbol
.common symbol, length [,align]  [,access]
.comm symbol, length [,align]  [,access]
.lcommon symbol, length [,align]  [,access]
.lcomm symbol, length [,align]  [,access]
.symver name, name2@nodename
.symver name, name2@@nodename
.symver name, name2@@@nodename
```

Attributes

```
.size symbol, expr
.type symbol, type
.hidden symbol [,symbol]...
.internal symbol [,symbol]...
.protected symbol [,symbol]...
```

Alignment

```
.org new-lc [,fill]
.align abs-expr [,abs-expr] [,abs-expr]
.balign[w|l] abs-expr [,abs-expr] [,abs-expr]
.p2align[w|l] abs-expr [,abs-expr] [,abs-expr]
.falign
```

Data

```
.byte [expr [,expr]...]
.2byte [expr [,expr]...]
.4byte [expr [,expr]...]
.word [expr [,expr]...]
.hword [expr [,expr]...]
.half [expr [,expr]...]
.short [expr [,expr]...]
.int [expr [,expr] ...]
```

```
.long [expr [,expr]...]
.quad [bignum [,bignum]...]
.single [flonum [,flonum]...]
.double [flonum [,flonum]...]
.float [flonum [,flonum]...]
.ascii ["string" [,"string"]...]
.asciz ["string" [,"string"]...]
.fill repeat, size, value
.space size [,fill]
.skip size [,fill]
.string "string" [,"string"]...
```

Conditionals

```
.if abs-expr
.ifdef symbol
.ifndef symbol
.ifnotdef symbol
.else
.elseif
.endif
```

Macros

```
.macro name
.macro name argument...
.endm
.purgem name
.irp symbol, value [,value]...
.irpc symbol, string
.endr
```

Include files

```
.include "filename"
```

Debug

```
.file fileno filename
.loc fileno lineno [column] [options]
.stabs string, type, other, desc, value
.uleb128 expr [,expr]...
.sleb128 expr [,expr]...
```

Assembler control

```
.rept [count]
```

3.5.1 .2byte

```
.2byte [expr [,expr]...]
```

Allocate 2-byte data items at consecutive addresses.

3.5.2 .4byte

```
.4byte [expr [,expr]...]
```

Allocate 4-byte data items at consecutive addresses.

3.5.3 .align

```
.align abs-expr [,abs-expr] [,abs-expr]
```

Pad the location counter in the current subsection, until it reaches an integral multiple of the value in the first argument.

The first argument is the alignment request value (in bytes). It must be an integral multiple of 2, and cannot exceed 0x3ffff.

The second argument specifies the value to be stored in the padding data. If the argument is omitted the value is zero (or in a text section, NOPs).

The third argument specifies the maximum number of bytes to be skipped. If the alignment would require skipping more bytes than the specified value, no alignment is performed.

NOTE The second argument can be omitted by placing two commas between the first and third arguments.

3.5.4 .ascii

```
.ascii ["string" [,"string"]...]
```

Allocate string literals (with no terminating zero bytes) as character data at consecutive addresses.

3.5.5 .asciz

```
.asciz ["string" [,"string"]...]
```

Allocate string literals (with automatic terminating zero bytes) as character data at consecutive addresses.

3.5.6 .balign

```
.balign[w|l] abs-expr [,abs-expr] [,abs-expr]
```

Pad the location counter in the current subsection, until it reaches an integral multiple of the value in the first argument.

The first argument is the alignment request value (in bytes). It must be an integral multiple of 2, and cannot exceed 0x3ffff.

The second argument specifies the value to be stored in the padding bytes. If the argument is omitted these bytes default to zero, or in a text section NOPs.

The third argument specifies the maximum number of bytes to be skipped. If the alignment would require skipping more bytes than the specified value, no alignment is performed.

In .balignw the padding value is a two-byte word value.

In .balignl the padding value is a four-byte longword value.

NOTE The second argument can be omitted by placing two commas between the first and third arguments.

3.5.7 .byte

`.byte [expr [,expr]...]`

Allocate the specified expressions as bytes stored at consecutive addresses.

3.5.8 .common

```
.common symbol, length [,align]  [,access]
.comm symbol, length [,align]  [,access]
```

Declare common data item with the specified symbol name and data size. The length argument specifies the data length (in bytes).

The *align* argument specifies the data alignment (in bytes). The alignment value must be an integral power of two. The default value is the largest power of two less than or equal to the length value.

The *access* argument specifies the size (in bytes) of the smallest memory access that will be made to the data item. The default value is the align value.

3.5.9 .data

`.data [subsection]`

Assign the following data declarations to the specified data subsection. The value must be absolute. The default is zero.

3.5.10 .double

`.double [flonum [,flonum]...]`

Allocate the specified flonums as 64-bit floating point numbers stored at consecutive addresses.

3.5.11 .else

`.else`

Conditional assembly directive.

3.5.12 .elseif

```
.elseif abs-expr
```

Conditional assembly directive.

3.5.13 .endif

```
.endif
```

Conditional assembly directive.

3.5.14 .endm

```
.endm
```

End of macro definition.

3.5.15 .endr

```
.endr
```

Mark the end of a sequence of source lines processed by the .rept directive.

3.5.16 .equ

```
.equ symbol,  expr
```

Set symbol value.

3.5.17 .equiv

```
.equiv symbol,  expr
```

Same as `.equ` and `.set`, except that an error is generated if the symbol is already defined.

3.5.18 .falign

```
.falign
```

Align instruction packet to not cross a cache line boundary (or for processor versions before V60, a 16-byte boundary).

Padding is accomplished by either adding NOPs to any empty slots in prior instruction packets, or by adding a new instruction packet containing NOPs.

This directive is used to reduce the amount of cache activity for time-critical code. falign is short for fetch align.

NOTE A label cannot be assigned to a source line containing an .falign directive.

3.5.19 .file

```
.file fileno filename
```

Assign file names to the file name table `.debug_line`.

The *fileno* argument specifies an index into the file name table. The index value must be unique with respect to any other instances of the file directive in the program.

The *filename* argument is a C string literal.

NOTE This is used when generating DWARF debug information.

3.5.20 .fill

```
.fill repeat, [,size] [,value]
```

Allocate fill data at consecutive addresses.

The `repeat` argument specifies the number of data items to allocate.

The `size` argument specifies the size (in bytes) of each data item. The default is 1.

The optional `value` argument specifies the value assigned to each item. The default is 0.

NOTE The second argument can be omitted by placing two commas between the first and third arguments.

3.5.21 .float

```
.float [flonum [,flonum]...]
```

Allocate the specified flonums as 32-bit floating point numbers stored at consecutive addresses.

3.5.22 .global

```
.global symbol
.globl symbol
```

Declare symbol as global.

3.5.23 .half

```
.half [expr [,expr]...]
```

Allocate the specified expressions as halfwords stored at consecutive addresses.

3.5.24 .hidden

`.hidden` *`symbol`* `[,`*`symbol`*`]...`

Set the symbol visibility of the specified symbols to “hidden.”

3.5.25 .hword

`.hword [`*`expr`* `[,`*`expr`*`]...]`

Allocate the specified expressions as halfwords stored at consecutive addresses.

3.5.26 .if

```
.if abs-expr
.ifdef symbol
.ifndef symbol
.ifnotdef symbol
```

Conditional assembly directive.

The condition can be the value of an expression (`.if`), whether a symbol is already defined (`.ifdef`), or whether a symbol is not defined (`.ifndef` and `.ifnotdef`).

3.5.27 .include

`.include "`*`filename`*`"`

Include the contents of the specified file into the program.

3.5.28 .int

`.int [`*`expr`* `[,`*`expr`*`]...]`

Allocate the specified expressions as 32-bit integers stored at consecutive addresses.

3.5.29 .internal

`.internal` *`symbol`* `[,`*`symbol`*`]...`

Set the symbol visibility of the specified symbols to “internal”.

3.5.30 .irp

`.irp symbol, value [,value]...`

Generate a macro-like sequence of similar statements, assigning a different value to the parameter `symbol` in each statement template.

The statement template is delimited by the `.irp` and `.endr` directives, with the symbol appearing in the template as a parameter prefixed by the backslash character (‘\’).

For example:

```
.irp    arg,1,2,3
        str x\arg, [sp, #-8*(\arg-18)]
.endr
```

This declaration generates the following assembly statements:

```
str x1, [sp, #-8*(1-18)]
str x2, [sp, #-8*(2-18)]
str x3, [sp, #-8*(3-18)]
```

3.5.31 .irpc

```
.irpc symbol, string
```

Generate a macro-like sequence of similar statements, assigning a different character from `string` to the parameter `symbol` in each statement template.

The statement template is delimited by the `.irpc` and `.endr` directives, with the symbol appearing in the template as a parameter prefixed by the backslash character (`\`).

For example:

```
.irpc    arg,123
        str x\arg, [x0, #\arg]
.endr
```

This declaration generates the following assembly statements:

```
str x1, [sp, #1]
str x2, [sp, #2]
str x3, [sp, #4]
```

3.5.32 .lcommon

```
.lcommon symbol, length [,align]  [,access]
.lcomm symbol, length [,align]  [,access]
```

Declare local common data item with the specified symbol name and data size. The length argument specifies the data length (in bytes).

The `align` argument specifies the data alignment (in bytes). The alignment value must be an integral power of two. The default value is the largest power of two less than or equal to the length value.

The `access` argument specifies the size (in bytes) of the smallest memory access that will be made to the data item. The default value is the align value.

3.5.33 .loc

```
.loc fileno lineno [column]  [options...]
```

Add rows to the .debug_line line number matrix.

The `fileno` argument specifies an entry index in the file name table. The `lineno` and `column` arguments specify a file position

The `options` argument specifies one or more instances of the following values: `basic_block`, `prologue_end`, `epilogue_end`, `is_stmt`, `value`, or `isa value`.

3.5.34 .long

`.long [`*`expr`* `[,`*`expr`*`]...]`

Allocate the specified expressions as 32-bit numbers stored in consecutive addresses.

Accepts zero or more expressions separated by commas.

NOTE `.long` is equivalent to `.int`.

3.5.35 .macro

```
.macro name
.macro name argument ...
```

Declare macro.

Macro arguments in the macro declaration must be separated by commas or spaces.

The macro definition is delimited by the `.macro` and `.endm` directives, with the macro parameters appearing in the macro definition as the macro argument name prefixed by the backslash character (`\`).

Macro arguments can be declared with the following options:

Argument	Description
`arg=default_value`	Default value if argument not specified in macro call.
`arg:req`	Argument must always be assigned non-blank value in macro calls.
`arg:vararg`	Argument assigned all remaining arguments in macro call.

In macro calls, arguments can be specified either by name (`add p1=17, p2=9`) or by order in the argument list (`add 17,9`).

3.5.36 .org

`.org new-lc [,fill]`

Advance the location counter of the current section, allocating pad data as necessary.

The `new-lc` argument must be absolute or an expression that evaluates to the same section as the current subsection.

The `fill` argument specifies the value assigned to the pad data. The default is zero.

3.5.37 .p2align

`.p2align[w|l] abs-expr [,abs-expr] [,abs-expr]`

Pad the location counter in the current subsection, until it reaches an integral multiple of 2x, where x is the value of the first argument.

The first argument effectively specifies the number of low-order zero bits that the location counter must contain after being advanced. The value must not exceed 17.

The second argument specifies the value to be stored in the padding bytes. If the argument is omitted these bytes default to zero, or in a text section NOPs.

The third argument specifies the maximum number of bytes to be skipped. If the alignment would require skipping more bytes than the specified value, no alignment is performed.

In `.p2alignw` the padding value is a two-byte word value.

In `.p2alignl` the padding value is a four-byte longword value.

NOTE The second argument can be omitted by placing two commas between the first and third arguments.

3.5.38 .protected

`.protected` *`symbol`* `[,`*`symbol`*`]...`

Set the symbol visibility of the specified symbols to “protected”.

3.5.39 .purgem

`urgem` *`name`*

Delete the macro definition for the specified macro. Once deleted, the name can be used for a new macro declaration.

3.5.40 .quad

`.quad [bignum [,bignum]...]`

Allocate bignums as 64-bit integers stored at consecutive addresses.

3.5.41 .rept

`.rept [`*`count`*`]`

Generate a macro-like sequence of statements, where `count` indicates the number of times the statement sequence is to be repeated in the code.

The statement sequence is delimited by the `.rept` and `.endr` directives.

The `count` argument specifies the number of times the sequence should appear in the code. The default is zero.

3.5.42 .section

`.section` *`name`* `[,"`*`flags`*`" [,`*`type`*`]]`

Specify a new section.

The *`flags`* argument specifies one or more section properties: allocatable (`a`), writable (`w`), or executable (`x`). The default depends on the section name – if it is not recognized, the section will contain data but have none of the indicated properties.

The *`type`* argument (`@progbits`, `@nobits`) specifies whether or not a section contains data. A section with no data only occupies space.

3.5.43 .set

`.set symbol, expr`

Set symbol to the specified value.

If the symbol was previously specified as external, it remains external.

If the symbol is global, the value that gets stored in the object file is the most recent value stored into the symbol.

3.5.44 .short

`.short [`*`expr`* `[,`*`expr`*`]...]`

Allocate the specified expressions as 16-bit numbers stored at consecutive addresses.

3.5.45 .single

`.single [`*`flonum`* `[,`*`flonum`*`]...]`

Allocate flonums as 32-bit floating point numbers stored at consecutive addresses.

3.5.46 .size

`.size symbol, expr`

Set the symbol size to the specified value (in bytes). The value must be absolute, but can be result of label arithmetic.

3.5.47 .sleb128

```
.sleb128 expr [,expr]...
```

Allocate the specified expressions as `sleb128`-format numbers stored at consecutive addresses.

NOTE `sleb128` is an acronym for “signed little-endian base 128.” This format is used when generating DWARF debug information.

3.5.48 .skip

```
.skip size [,fill]
```

Allocate bytes at consecutive addresses, using the specified fill value. The default fill value is zero.

3.5.49 .space

```
.space size [,fill]
.block size [,fill]
```

Allocate bytes at consecutive addresses, using the specified fill value. The default fill value is zero.

3.5.50 .stabs

```
.stabs name, type, other, desc, value
```

Define symbols that are used by symbolic debuggers.

The directive arguments are used to set the following symbol attributes: `name`, `value`, `type`, `descriptor`, and `other`.

NOTE These symbols are used for debugging – they cannot be referenced in the assembly source code.

3.5.51 .string

```
.string "string" ["string"]...
```

Allocate string literals as character data stored at consecutive byte addresses. Each string is terminated by a zero byte.

3.5.52 .symver

```
.symver name, name2@nodename
.symver name, name2@@nodename
ymver name, name2@@@nodename
```

Bind symbols to specific version nodes in a source file. This is done when developing programs that work with shared objects.

The `name2` argument specifies the name used to reference the symbol externally.

The `nodename` argument specifies the name of a node in the linker script that is used when building a shared library.

The single-`@` version of this directive behaves differently depending on whether or not the symbol is defined in the current source file. If it *is* defined, the symbol is aliased with the name `name2@nodename`. This enables multiple versions of the same function to co-exist in a source file, while still allowing the compiler to be able to distinguish the function references. If the symbol is *not* defined, all references to it are changed to `name2@nodename`.

The double-`@` version of this directive specifies that the symbol must be defined in the source file. It is the same as the one-`@` version, except that `name2@@nodename` will also be used to resolve references to `name2`.

The triple-`@` version specifies that, when the symbol is not defined in the source file, it will be treated as `name2@nodename`. Otherwise it will be renamed as `name2@@nodename`.

3.5.53 .text

`.text [subsection]`

Assign the following ARM instructions to the specified code subsection. The value must be absolute. The default is zero.

3.5.54 .type

`.type symbol, type`

Set the symbol type.

The `type` argument can be one of the following values.

- `@function` – Function name
- `@gnu_indirect_function` – Indirect function name
- `@object` – Data object
- `@tls_object` – Thread-local storage data object
- `@common` – Common data object
- `@notype` – Typeless symbol
- `@gnu_unique_object` – Globally unique in process

3.5.55 .uleb128

`.uleb128 expr [,expr]...`

Allocate expressions as `uleb128`-format numbers stored at consecutive addresses.

NOTE `uleb128` is an acronym for “unsigned little-endian base 128.” This format is used when generating DWARF debug information.

3.5.56 .word

`.word [expr [,expr]...]`

Allocate expressions as 32-bit words stored at consecutive addresses.

3.6 ARM-specific directives

3.6.1 .arch

`.arch name`
`Example: .arch armv5`

Select the target architecture. Valid values for `name` are the same as for the `-march` commandline option.

NOTE `.arch` clears previously selected architecture extensions.

3.6.2 .arch_extension

`.arch_extension name`
`Example: .arch_extension crypto`

Add or remove an architecture extension to the target architecture. Valid values for name are the same as those accepted as architectural extensions by the –mcpu commandline option.

This directive can be used multiple times to incrementally add or remove architectural extension.

3.6.3 .arm

`Example: .arm`

This is the same as specifying `.code 32`

3.6.4 .cantunwind

`.cantunwind`

Prevents unwinding through the current function.

3.6.5 .code

`.code [16|32]`
`Example: .code 16`

This directive indicates the instruction set being generated. The value 16 selects Thumb and the value 32 selects ARM.

3.6.6 .cpu

```
.cpu name
Example: .cpu cortex-m3
```

This directive selects the target processor. Valid values for the `name` are the same as for the `-mcpu` commandline option.

Specifying `.cpu` clears any previously selected architecture extensions.

3.6.7 .eabi_attribute

```
.eabi_attribute tag, value
Example: .eabi_attribute Tag_CPU_arch_profile, 'A'
```

This directive sets the EABI object attribute `tag` to `value`. The tag is either an attribute number or one of the following:

```
Tag_CPU_raw_name, Tag_CPU_name, Tag_CPU_arch, Tag_CPU_arch_profile,
Tag_ARM_ISA_use, Tag_THUMB_ISA_use, Tag_FP_arch, Tag_WMMX_arch,
Tag_Advanced_SIMD_arch, Tag_PCS_config, Tag_ABI_PCS_R9_use,
Tag_ABI_PCS_RW_data, Tag_ABI_PCS_RO_data, Tag_ABI_PCS_GOT_use,
Tag_ABI_PCS_wchar_t, Tag_ABI_FP_rounding, Tag_ABI_FP_denormal,
Tag_ABI_FP_exceptions, Tag_ABI_FP_user_exceptions, Tag_ABI_FP_number_model,
Tag_ABI_align_needed, Tag_ABI_align_preserved, Tag_ABI_enum_size,
Tag_ABI_HardFP_use, Tag_ABI_VFP_args, Tag_ABI_WMMX_args,
Tag_ABI_optimization_goals, Tag_ABI_FP_optimization_goals, Tag_compatibility,
Tag_CPU_unaligned_access, Tag_FP_HP_extension, Tag_ABI_FP_16bit_format,
Tag_MPextension_use, Tag_DIV_use, Tag_nodefaults, Tag_also_compatible_with,
Tag_conformance, Tag_T2EE_use, Tag_Virtualization_use
```

The `value` is either a number or string literal depending on the tag.

3.6.8 .even

```
.even
```

This directive aligns to an even-numbered address.

3.6.9 .fnend

```
.fnend
```

This directive marks the end of a function with an unwind table entry.

3.6.10 .fnstart

```
.fnstart
```

This directive marks the start of a function with an unwind table entry.

3.6.11 .fpu

```
.fpu name
Example: .fpu neon-vfpv4
```

This directive selects the floating-pint unit to assemble for. Valid values for `name` are the same as for the –`mfpu` commanline option.

3.6.12 .handlerdata

```
.handlerdata
```

This directive marks the end of the current function, and the start of the exception table entry for that function. Anything between this directive and the `.fnend` directive will ee added to the exception table entry.

3.6.13 .inst

```
.inst opcode [, …]
.inst.n opcode [, …]
.inst.w opcode [, …]
Example: .inst.w 0xf2400000, 0xf2c00000
```

This directive generates the instruction corresponding to the numerical enconding of `opcode`.

`.inst.n` and `.inst.w` allows the Thumb instruction size to be explicitly, overriding the normal encoding rules.

3.6.14 .ltorg

```
.ltorg
```

This directive causes the current contents of the literal pool to be dumped into the current section at the current location.

3.6.15 .movesp

```
.movsp reg
Example: .movsp r7
```

This directive indicates the unwinder that `reg` contants an offset from the current stack pointer.

3.6.16 .object_arch

```
.object name
Example: .object_arch armv7
```

This directive override the architecture recorded in the EABI object attribute section. Valid values for `name` are the same as for the `.arch` directive. This is useful when code uses runtime detection of CPU features.

3.6.17 .personality

```
.personality name
Example: .personality    __gxx_personality_v0
```

This directive sets the personality routine to `name`.

3.6.18 .personalityindex

```
.personalityindex index
Example: .personalityindex    0
```

This directive sets the personality routine to the EABI standard routine number `index`.

3.6.19 .pool

```
.pool
```

This directive is a synonym for `.ltorg`.

3.6.20 .req

```
name .req register name
Example: foo .req    r2
```

This directive creates an alias of `name` to `register name`.

3.6.21 .save

```
.save reglist
Example: .save   {r4, r5, r6, lr}
```

This directive generates unwinder annotations to restore the registers in `reglist`. The format of `reglist` is the same as the corresponding store-multiple instruction.

3.6.22 .setfp

```
.setfp fpreg, spreg [, #offset]
Example: .setfp r11, sp, #8
```

This directive makes all unwinder annotations relative to a frame pointer. Without his the unwinder will use offset from the stack pointer.

3.6.23 .syntax

```
syntax [unified | divided]
Example: .syntax    unified
```

This directive sets the instruction set syntax. The default, `divided`, uses the old sytle where ARM and Thumb instructions have their own, separate syntax. The new, `unified` syntax unifies the syntax for both of the instructions.

3.6.24 .thumb

```
.thumb
```

This directive is a synonym for `.code 16`.

3.6.25 .thumb_set

```
.thumb_set
Example: .thumb_set beta, alpha
```

This directive performs the equivalent of a `.set` directive to create a symbol which is an alias for another symbol. This directive also marks the aliased symbol as being a thumb function entry point, in the same way that the `.thumb_func` directive does.

3.6.26 .unreq

```
.unreq alias-name
Example: .unreq    foo
```

This directive undefines a register alias which was previously defined using the `req` directive.

3.6.27 .unwind_raw

```
.unwid_raw offset, byte1, …
Example: .unwind_raw 0, 0x100
```

This directive inserts one of more arbitrary unwind opcode bytes, which are known to adjust the stack pointer by `offset` bytes.

3.6.28 .vsave

```
.vsave vfp-reglist
Example: .vsave    {d8, d9, d10, d11, d12}
```

This directive generates unwinder annotations to restore the VFP registers in `vfp-reglist` using FLDMD. The format of `vfp-reglist` is the same as the corresponding store-multiple instruction.

3.7 Arch64-specific directives

3.7.1 .bss

`.bss`

This directive switches to the .bss section

3.7.2 .ltorg

`.ltorg`

This directive causes the current contents of the literal pool to be dumped into the current section at the current location.

3.7.3 .pool

`.pool`

This directive is a synonym for `.ltorg`.

3.7.4 .req

name `.req` *register name*
`Example: foo .req x2`

This directive creates an alias of `name` to `register name`.

3.7.5 .unreq

`.unreq` *alias-name*
`Example: .unreq foo`

This directive undefines a register alias which was previously defined using the `req` directive.

4 Archiver

The archiver utility creates and maintains groups of files combined into an archive. Once an archive has been created, new files can be added to it, and existing files can be extracted, deleted, or replaced.

Files are named in the archive by their last file name component, so if a file referenced by a path containing a '/' is archived, it is named by the last component of the path. Similarly when matching paths listed on the command line against file names stored in the archive, only the last component of the path is compared.

The normal use of the archiver is for the creation and maintenance of libraries suitable for use with the linker, although it is not restricted to this purpose. The archiver can create and manage an archive symbol table which is used to speed up linking. If a symbol table is present in an archive, it is kept up-to-date by subsequent operations on the archive.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To invoke the archiver, use the following command format:

```
arm-ar <generic modifiers><operation><operation modifier with positional
member  names  (if  any)> archive members
```

For commands used to accomplish specific archiving tasks, type one of the following commands from a command line:

```
arm-ar -d[Tvz] archive file...
arm-ar -m[Ta|b|i] member archive file...
arm-ar -p[Tv] archive file...
arm-ar -q[TcDFs|Svz] archive file...
arm-ar -r[TcDFs|Suvza|b|i] member archive file...
arm-ar -s[Dz] archive
arm-ar -t[Tv] archive file... arm-ar -x
[CTouv]  archive file... arm-ar -M
arm-ar -V
```

4.1 Options

The archiver utility supports the following options:

- Options for operations – `d, m, p, q, r, t, x`
- Options for generic modifiers – `c, s, S, T, v, V, @file`
- Options for operation-specific modifiers – `a, b, i, D, o, u`

NOTE All the options must be “clubbed” together as shown above (space separation is not allowed). The only options where spaces are permitted are the member names specification for `-i`, `-a`, and `-b` positional options.

Options for operations

`-d`

Delete the archive members specified as `file` arguments on the command line from the archive specified on the command line.

The archive's symbol table, if present, is updated to reflect the new contents of the archive.

`-m`

Move archive members specified as `file` arguments on the command line within the archive specified on the command line.

If a position has been specified by `-a`, `-b`, or `-i`, the members are moved to before or after the specified position. If no position has been specified, the specified members are moved to the end of the archive. If the archive has a symbol table, it is updated to reflect the new contents of the archive.

`-p`

Write to the standard output the contents of the archive members specified as `file` arguments on the command line. If no members were specified, the contents of all the files in the archive are written in the order they appear in the archive.

`-q`

Append the files specified as `file` arguments on the command line to the archive specified on the command line, without checking if the files already exist in the archive. The archive symbol table is updated as necessary. If the file specified by the argument archive does not already exist, a new archive is created.

`-r`

Replace (add) the files specified as `file` arguments on the command line in the archive specified on the command line, creating a new archive if necessary.

Replacing existing members does not change the order of members within the archive. If one of the specified files does not exist, any existing archive members that match the file name are not changed. New files are added to the end of the archive unless one of the positioning options `-a`, `-b`, or `-i` is specified. The archive symbol table, if it exists, is updated to reflect the new state of the archive.

`-t`

List the files specified as `file` arguments on the command line in the order they appear in the archive specified as the `archive` arguments, with one file listed per line. If no files are specified, all the files in the archive are listed.

`-x`

Extract to the current directory all archive members specified as `file` arguments on the command line. If no members are specified, extract all members of the archive. If the file corresponding to an extracted member does not exist, it is created. If the file corresponding to an extracted member does exist, its owner and group are not changed, while its contents are overwritten and its permissions are set to those entered in the archive. The file's access and modification time are set to the time of extraction unless `-o` is specified.

Options for generic modifiers

`-c`

Suppress the informational message printed when a new archive is created using `-r` or `-q`.

`-s`

Add an archive symbol table to the archive specified as the `archive` argument on the command line. Invoking the archiver with `-s` alone is equivalent to invoking the archive indexer utility.

`-S`

Do not generate an archive symbol table.

`-T`

When naming archive members, use only the first fifteen characters of the archive member name or command line `file` argument.

`-v`

Provide verbose output. When used with `-d`, `-m`, `-q`, or `-x`, the archiver generates a file-by-file description of the archive modification being performed, which consists of three white-space separated fields: the option letter, a dash '-', and the file name.

When used with `-r`, the archiver displays the description as above, but the initial letter is an '`a`' if the file is added to the archive, or an '`r`' if the file replaces a file already in the archive.

When used with `-p`, the name of the file enclosed in < and > characters is written to the standard output preceded by a single newline character and followed by two newline characters. The contents of the named file follow the file name.

When used with `-t`, the archiver displays the following fields: the file permissions; the decimal user and group IDs separated by a slash (`/`); the file size in bytes; the file modification time in format `%b %e %H:%M %Y`; and the name of the file.

`-V`

Print the archiver version number and exit.

`-@file`

Read commands from the specified file.

Options for operation-specific modifiers

`-a member`

When used with `-m`, this option causes the archive members specified as `file`

arguments on the command line to be moved `after` the specified archive member.

When used with `-r`, this option causes the files specified as `file` arguments on the command line to be added `after` the specified archive member.

`-b member`

When used with `-m`, this option causes the archive members specified as `file`

arguments on the command line to be moved before the specified archive member.

When used with -r, this option causes the files specified as `file` arguments on the command line to be added before the specified archive member.

`-i archive`

Equivalent to `-b`.

`-D`

When used with `-r` or `-q`, insert zero values instead of the actual mtime, uid, and gid values, and 0644 instead of file mode, for the archive members specified as `file` arguments on the command line.

This ensures that checksums on the resulting archives are reproducible when the member contents are identical.

`-o`

Preserve the original modification times of members when extracting them.

`-u`

Conditionally update the archive or extract members.

When used with `-r`, the files specified as `file` arguments on the command line are replaced in the archive if they are newer than their archived versions.

When used with `-x`, the archive members specified as `file` arguments on the command line are extracted only if they are newer than the corresponding files in the file system.

Other options

`-C`

Prevent extracted files from replacing like-named files in the file system.

`-F flavor`
`--flavor` *flavor*

Create archives with the specified archive format. The possible values are:

```
bsd        Create BSD-format archives
gnu        Alias for svr4
svr4        Create SVR4-format archives (default)
-M
```

Read and execute archiver script commands from the standard input.

`-z`

This option is accepted for compatibility with the FreeBSD archiver, but is ignored.

RELATED INFORMATION

"Examples" on page 46
"Archive indexer" on page 59
"Command scripts" on page 44

4.2 Command scripts

If the `-M` option is specified, the archiver utility can be controlled using script commands read from the standard input.

If the standard input is a terminal, the archiver displays the prompt `AR >` before reading a line, and continues executing even if errors are encountered. If the standard input is not a terminal, the archiver does not display a prompt, and terminates execution on encountering an error.

Each input line contains a single command. The words in an input line must be separated by whitespace characters: the first word of the line is the command, while the remaining words are the arguments to the command. The command word can be specified in lower or upper case. The command arguments can be separated by commas or blanks.

Empty lines are allowed and ignored.

Long lines are continued by ending them with the + character.

The * and ; characters start a comment. Comments extend to the end of the line.

When executing an archiver script, the archiver works on a temporary copy of an archive. Changes to the copy are made permanent using the save command.

Commands

The archiver recognizes the following script commands:

```
addlib archive
addlib archive (member [, member]...)
```

Add the contents of the specified archive to the current archive. If one or more members are additionally specified, they are added to the current archive. If no members are specified, the entire contents of the specified archive are added to the current archive.

```
addmod member [, member]...
```

Add the specified files to the current archive.

```
clear
```

Discard all the contents of the current archive.

```
create archive
```

Create a new archive with the specified name, and make it the current archive. If the named archive already exists, it is overwritten when the save command is invoked.

```
delete member [, member]...
```

Delete the specified modules from the current archive.

```
directory archive (member [, member]...) [outputfile]
```

List the specified members in the archive. The output format depends on the verbosity level set using the verbose command. Output is sent to the standard output, or to the file specified by `outputfile`.

```
end
```

Exit successfully from the archiver. Any unsaved changes to the current archive are discarded.

```
extract member [, member]...
```

Extract the specified members from the current archive.

```
list
```

Display the contents of the current archive in verbose format.

```
open archive
```

Open the specified archive and make it the current archive.

```
replace member [, member]...
```

Replace named members in the current archive with the files specified by arguments member.

The files must be present in the current directory and the specified members must already exist in the current archive.

`save`

Commit all changes to the current archive.

`verbose`

Toggle the verbosity of the directory command.

4.3 Examples

To create a new archive `ex.a` containing three files `ex1.o`, `ex2.o`, and `ex3.o`, use the following command:

```
arm-ar -rc ex.a ex1.o ex2.o ex3.o
```

To add an archive symbol table to an existing archive `ex.a`, use:

```
arm-ar -s ex.a
```

To delete file `ex1.o` from archive `ex.a`, use:

```
arm-ar -d ex.a ex1.o
```

To verbosely list the contents of archive `ex.a`, use:

```
arm-ar -tv ex.a
```

To create a new archive `ex.a` that contains the files `ex1.o` and `ex2.o`, use the command `arm-ar -M < script` with the following archiver script:

```
create ex.a        * specify the output archive addmod
ex1.o ex2.o        * add modules
save            * save pending changes
end               * exit the utility
```

To create an archive named `lib1.a` with member names starting as mem and having a suffix `.o`:, use:

```
arm-ar crsU lib1.a mem*.o
```

To print a list with all member in `lib1.a`, use:

```
arm-ar tv lib1.a
```

To delete `mem2.o` from `lib1.a`, use:

```
arm-ar d lib1.a mem2.o
```

To insert `mem2.o` before `mem6.o` in library `lib1.a`, use:

```
arm-ar rb mem6.o lib1.a mem2.o
```

5 Object file symbols

The object file symbols utility lists the symbols in the specified executable files, object files, or object library files.

If no input files are specified, the utility attempts to read from `a.out`.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the object files symbols utility from a command line, type:

```
arm-nm [--debug-syms] [--defined-only]
   [--dynamic] [--help] [--line-numbers]
   [--no-sort] [--numeric-sort] [--print-armap]
   [--print-file-name] [--print-size] [--radix=format]
   [--reverse-sort] [--size-sort] [--undefined-only] [--version] [-A]
   [-B]  [-D]  [-P]  [-S]  [-V]  [-a]  [-e]  [-g]  [-h]
   [-n]  [-o]  [-p]  [-r]  [-s]  [-t format]  [-u]  [-x]  [file...]
```

5.1 Options

The object file symbols utility supports the following options:

`--debug-syms`

Display all symbols, including debugger-only symbols.

`--defined-only`

Display only defined symbols.

`--d--dynamic`

Only display dynamic symbols. This option is meaningful only for shared libraries.

`--help`

Display the command usage info and exit.

`--format=`*`format`*

Display output in the specified format. The possible values are:

```
bsd    BSD (default)
sysv    System V
posix    POSIX
--no-sort
```

Do not sort symbols.

`--numeric-sort`

Sort symbols numerically by address instead of alphabetically by name.

`--print-armap`

For archives, include the index of the archive's members.

`--print-file-name`

Write the full pathname or library name of an object on each line, before the rest of the information for a symbol. If this option is not specified, the utility identifies an input file only once, before its symbols are listed.

`--print-size`

Print the size of each symbol instead of its value.

`--radix=`*`radix`*

Print numeric values using the specified radix. The possible values are:

- `d` – Decimal (default)
- `o` – Octal
- `x` – Hexadecimal

`--reverse-sort`

Reverse the order of the sort.

`--size-sort`

Sort symbols by size instead of alphabetically by name.

`--undefined-only`

Display only undefined symbols.

`--version`

Display the utility version identifier and exit.

`-A`

Equivalent to `--print-file-name`.

`-B`

Equivalent to `--format=bsd`.

`-D`

Equivalent to `--dynamic`.

`-F format`

Equivalent to `--format=`*`format`*.

`-P`

Equivalent to `--format=posix`.

`-S`

Equivalent to `--print-size`.

`-V`

Equivalent to `--version`.

`-a`

Equivalent to `--debug-syms`.

`-e`

Only display information for global and static symbols.

`-f`

Produce full output (default).

`-g`

Only display information on global (external) symbols.

`-h`

Equivalent to `--help`.

`-n`

Equivalent to `--numeric-sort`.

`-o`

If POSIX output was specified (using `-F` posix or `-P`), this option is equivalent to `--radix=o`. If POSIX output was not specified, it is equivalent to `--print-file-name`.

`-p`

Equivalent to `--no-sort`.

`-v`

Equivalent to `-n`.

`-r`

Equivalent to `--reverse-sort`.

`-s`

Equivalent to `--print-armap`.

`-t radix`

Equivalent to `--radix=`*`radix`*.

`-u`

Equivalent to `--undefined-only`.

`-x`

Write numeric values in hexadecimal (equivalent to `-t x`).

5.2 Output formats

The object file symbols utility can display information in a number of formats, numeric radices, and sort orders.

By default it uses BSD-style output, hexadecimal radix, no alphabetic sorting by name, and no name demangling.

For each symbol listed, the utility presents the following information:

- The library or object name (if `-A` or `--print-file-name` were specified)
- The symbol name
- The symbol type (denoted by a single character):

A – Global absolute symbol

B – Global bss symbol (uninitialized data)

C – Common symbol (representing uninitialized data)

D – Global symbol naming initialized data

N – Debugger symbol

R – Read-only data symbol

T – Global text symbol

U – Undefined symbol

V – Weak object

W – Weak reference

a – Local absolute symbol

b – Local bss symbol (uninitialized data)

d – Local data symbol

t – Local text symbol

v – Weak object that is undefined

w – Weak symbol that is undefined

? – None of the above

- The symbol value
- The symbol size (if applicable)
- Line number information (if available, and if `-l` or `--line-numbers` were specified)

6 Object file copier

The object file copier utility copies the contents of the specified ELF object file `infile` to a new object file named `outfile`, while performing transformations on the copied file that are specified by the command options.

If the `outfile` argument is not specified, the utility names the copied file `infile`, thus overwriting the original file.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the object file copier utility from a command line, type:

```
arm-elfcopy [--add-section sectionname=filename]
    [--adjust-section-vma section(+|-|=)val]
    [--adjust-start=increment] [--adjust-vma=increment]
    [--adjust-warnings] [--change-addresses=increment]
    [--change-section-address section(+|-|=)val]
    [--change-section-lma section(+|-|=)val]
    [--change-section-vma section(+|-|=)val] [--change-start=increment]
    [--change-warnings] [--discard-all] [--discard-locals]
    [--gap-fill=val] [--help] [--input-target=objformat]
    [--keep-symbol=symbolname] [--localize-symbol=symbolname]
    [--no-adjust-warnings] [--no-change-warnings] [--only-keep-debug]
    [--only-section=sectionname] [--output-target=objformat]
    [--pad-to=address] [--prefix-alloc-sections=string]
    [--prefix-sections=string] [--prefix-symbols=string] [--preserve-dates]
    [--remove-section=sectionname]
    [--rename-section oldname=newname[[,flag]...]]
    [--set-section-flags sectionname=flag[[,flag]...]
    [--set-start=address] [--srec-forceS3] [--srec-len=val]
    [--strip-all] [--strip-debug] [--strip-symbol=symbolname]
    [--strip-unneeded] [--version] [--weaken-symbol=symbolname]
    [--wildcard] [-I objformat] [-K symbolname] [-L symbolname]
    [-N symbolname]  [-O objformat]  [-R sectionname]  [-S]  [-V]
    [-W symbolname]  [-X]  [-d]  [-g]  [-h]  [-j sectionname]  [-p]
    [-s objformat]  [-w]  [-x]  infile [outfile]
```

6.1 Options

The object file copier utility supports the following options:

`--add-section sectionname=filename`

Add the specified section to the output file. The contents of the section are obtained from the file filename. The section size is the number of bytes in filename.

`--adjust-section-vma section(+|-|=)val`

Equivalent to --change-section-address.

`--adjust-start=increment`

Equivalent to --change-start.

`--adjust-vma=increment`

Equivalent to --change-addresses.

`--adjust-warnings`

Equivalent to --change-warnings.

`--change-addresses=increment`

Increase the virtual memory address and the load memory address of all sections by the specified value.

`--change-section-address section(+|-|=)val`

Depending on the operator specified, increase, decrease or set both the virtual memory address and the load memory address of the specified section. The specified value indicates the desired increment, decrement, or new value of the address.

`--change-section-lma section(+|-|=)val`

Change or set the load memory address of the specified section. Depending on the operator specified, the specified value is used as an increment, a decrement, or the new value of the load memory address.

`--change-section-vma section(+|-|=)val`

Change or set the virtual memory address of the specified section. Depending on the operator specified, the specified value is used as an increment, a decrement, or the new value of the virtual memory address.

`--change-start=increment`

Increase the entry point address of the output file by the specified amount.

`--change-warnings`

Issue a warning if the section specified by the options `--change-section-address`, `--change-section-lma` or `--change-section-vma` does not exist in the input object. This is the default option setting.

`--discard-all`

Do not copy non-global symbols to the output.

`--discard-locals`

Do not copy compiler-generated local symbols to the output.

`--gap-fill=val`

Fill gaps between sections with the specified byte value.

`--help`

Display the command usage info and exit.

`--input-target=objformat`

This option is accepted for compatibility with the GNU object file copier, but is ignored.

`--keep-symbol=symbolname`

Copy the specified symbol to the output.

`--localize-symbol=symbolname`

Make the specified symbol local to the output file.

`--no-adjust-warnings`

Equivalent to `--no-change-warnings`.

`--no-change-warnings`

Do not issue a warning if the section specified by the options `--change-section-address`, `--change-section-lma`, or `--change- section-vma` is missing in the input object.

`--only-keep-debug`

Copy only debugging information to the output file.

`--only-section=sectionname`

Copy only the specified section to the output.

`--output-target=objformat`

Write output file using the specified object file format. The possible values are:

`ETF_ELF` ELF object (default)

`ETF_BINARY` Raw binary

`ETF_IHEX` Object encoded in Intel hex format

`ETF_NONE` Unknown object format

`ETF_SREC` Object encoded as S-records

`--pad-to=address`

Pad the load memory address of the output object to the specified address value by increasing the size of the section with the highest load memory address.

`--prefix-alloc-sections=string`

Prefix the section names of all allocated sections with the specified string.

`--prefix-sections=string`

Prefix the section names of all sections with the specified string.

`--prefix-symbols=string`

Prefix the symbol names of all symbols with the specified string.

`--preserve-dates`

Set the access and modification times of the output file to the same as those of the input.

`--rename-section oldname=newname[[,flag]...]`

Rename the section specified by oldname to newname, optionally changing the section flags to the flag values specified by flag. The possible flag values are listed in `--set-section-flags` below.

`--remove-section=sectionname`

Remove any section with the specified name from the output file.

`--set-section-flags sectionname=flag[,flag]...`

Set one or more flags for the specified section. The possible values are:

`alloc` Section occupies space in the output file.

`code` Section contains machine instructions.

`contents` Flag accepted but ignored.

`data` Section contains writable data.

`debug` Section holds debugging information.

`load` Section loadable.

`noload` Section should not be loaded into memory.

`readonly` Section not writable.

`rom` Section contains ROM'able contents.

`share` Flag accepted but ignored.

`--set-start=address`

Set the start address of the output ELF object to the specified value.

`--srec-forceS3`

Only generate S-records of type S3. This option is only meaningful when the output target is set to srec.

`--srec-len=val`

Set the maximum length of an S-record line to the specified value. This option is only meaningful when the output target is set to srec.

`--strip-all`

Do not copy symbol and relocation information to the target file.

`--strip-debug`

Do not copy debugging information to the target file.

`--strip-symbol=symbolname`

Do not copy the specified symbol to the output.

`--strip-unneeded`

Do not copy symbols that are not needed for relocation processing.

`--version`

Display the utility version identifier and exit.

`--weaken-symbol=symbolname`

Mark the specified symbol as weak in the output.

`--wildcard`

Use shell-style patterns to specify symbols. The following meta-characters are recognized in patterns:

! If this is the first character of the pattern, invert the sense of the pattern match

* Match any string of characters in a symbol name

? Match zero or one character in a symbol name

[Mark the start of a character class

\ Remove the special meaning of the next character in the pattern

] Mark the end of a character class

`-I objformat`

Equivalent to `--input-target`.

`-K symbolname`

Equivalent to `--keep-symbol`.

`-L symbolname`

Equivalent to `--localize-symbol`.

`-N symbol`

Equivalent to `--strip-symbol`.

`-O objformat`

Equivalent to `--output-target`.

`-R sectionname`

Equivalent to `--remove-section`.

`-S`

Equivalent to `--strip-all`.

`-V`

Equivalent to `--version`.

`-W symbolname`

Equivalent to `--weaken-symbol`.

`-X`

Equivalent to `--discard-locals`.

`-d`

Equivalent to `--strip-debug`.

`-g`

Equivalent to `--strip-debug`.

`-h`

Equivalent to `--help`.

`-j sectionname`

Equivalent to `--only-section`.

`-p`

Equivalent to `--preserve_dates`.

`-s objformat`

Equivalent to `--input-target`.

`-w`

Equivalent to `--wildcard`.

`-x`

Equivalent to `--discard-all`.

7 Object file viewer

The object file viewer utility displays information on the specified object files or archive files.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs. This tool needs a .ARM.attributes ELF section in the binary file. The section is not loaded into memory and is only used to find architectural information of the binary.

Command

To start the object file viewer utility from a command line, type:

```
llvm-objdump [-arch=string] [-cfg] [-disassemble] [-dsym=string] [-g]
   [-help] [-macho] [-mattr=a1,+a2,-a3,...] [-mcpu] [-no-show-raw-insn]
   [-private-headers] [-r] [-s] [-source / -S][-linu-numbers / -l]
   [-section-headers] [-stats] [-t] [-triple=string] [-unwind-info]
   [-version] file...
```

7.1 Options

The object file viewer utility supports the following options:

`-arch=`*`string`*

Architecture version of input files. To list the possible values, run the object file viewer utility with the -version option.

`-cfg`

Create a CFG for every symbol in the object file and write it to a Graphviz file (MachO-only).

`-disassemble`

Display assembler mnemonics for the machine instructions.

`-dsym=`*`string`*

Use the .dSYM file for debug information.

`-g`

Print line information from debug information (if available).

`-help`

Display command usage info and exit (for additional info use `-help-hidden`).

Display command usage info and exit (for additional info use `-help-hidden`).

`-macho`

Use MachO-specific object file parser.

`-mattr=`*`a1`*`,+`*`a2`*`,-`*`a3`*`,...`

This option is accepted by the object file viewer utility, but is ignored.

`-mcpu=cpuname`

Specify CPU for disassembler sub-architecture. Useful when .ARM.attributes are missing

`-no-show-raw-insn`

When disassembling instructions, do not print the instruction bytes.

`-private-headers`

Display format-specific file headers.

`-r`

Display the relocation entries.

`-s`

Display the contents of each section.

`-section-headers`

Display summaries of the headers for each section.

`-source / -S`

Interleave source code lines with disassembly

`-line-numbers / -l`

Interleave source line numbers with disassembly

`-stats`

Enable statistics output from program (available with Asserts).

`-t`

Display symbol table.

`-triple=`*`string`*

Architecture target triple to disassemble for. To list the possible values, run the object file viewer utility with the -version option.

`-unwind-info`

Display unwind information.

`-version`

Display the utility version identifier and exit.

8 Archive indexer

The archive indexer utility updates the archive symbol table in the specified archive file. If the archive file does not contain a symbol table, a new one is added to the file.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the archive indexer utility from a command line, type:

```
arm-ranlib [-D] [-t] archive...
arm-ranlib -V
```

8.1 Options

The archive indexer utility supports the following options:

`-D`

For all archive member headers, assign zeros to the mtime, uid, and gid fields, and use mode 0644 for the file mode field. This ensures that checksums on the resulting archives are reproducible when member contents are identical.

`-t`

This option is accepted, but is ignored.

`-V`

Display utility version identifier and exit.

9 Object file size

The object file size utility lists the sizes of ELF sections (and optionally the total size) in the specified files. The files can be object files, archive files, or core dumps.

If no input files are specified, the utility attempts to read from a.out.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the object file size utility from a command line, type:

```
arm-size [--format=format] [--help] [--radix=radix] [--version]  [-A]
[-B]  [-V]  [-d]  [-h]  [-o]  [-x]  [file...]
```

9.1 Options

The object file size utility supports the following options:

`--format=`*`format`*

Display output using the specified format (System V or Berkeley). The possible values are sysv and berkeley (default). For more information on display formats see the sections below.

`--help`

Display command usage info and exit.

`--radix=`*`radix`*

Display numeric values using the specified radix. The possible values are 8, 10, or

16. The default is 10.

`--version`

Display utility version identifier and exit.

`-A`

Equivalent to `--format=sysv`.

`-B`

Equivalent to `--format=berkeley`.

`-V`

Equivalent to `--version`.

`-d`

Equivalent to `--radix=10`.

`-h`

Equivalent to `--help`.

`-o`

Equivalent to `--radix=8`.

`-x`

Equivalent to `--radix=16`.

9.2 Output formats

The object file size utility can display its output in one of two formats:

- Berkeley-style output
- System V-style output

Berkeley-style output

Berkeley-style output consists of an initial header line (labeling the individual fields), followed by one line of output for each ELF object that is either specified directly on the command line, or contained in a specified archive.

Each line contain the following fields:

- The size of the `text` segment in the object.
- The size of the `data` segment in the object.
- The size of the `bss` segment in the object.
- The total size of the object, in decimal or octal format. Decimal is used if the specified output radix for numeric values is 10 or 16. Octal is used if the radix is 8.
- The total size of the object, in hexadecimal format.
- The file name of the object.

If `--totals` was specified, the output includes an additional line (in the same format as above) which contains the sum of the respective fields. The file name field for this line contains the string `(TOTALS)`.

System V-style output

System V-style output consists of the following information for each ELF object that is either specified directly on the command line, or contained in a specified archive:

- The name of the object followed by a colon character.
- A header line containing the field labels for the subsequent lines.
- One line for each section contained in the object. Each line has three fields:
  - The section name.
  - The section size, in the specified radix.

- The section address, in the selected radix.

An additional line with the section name set to `Total`, and a size field set to the sum of all the reported section sizes.

9.3 Examples

Section sizes for `/bin/ls`:

text	data	bss	dec	hex	filename
20975	540	392	21907	5593	`/bin/ls`

Sizes and total for `/bin/ls` and `/bin/dd` in hexadecimal:

text	data	bss	dec	hex	filename
0x51ef	0x21c	0x188	21907	5593	/bin/ls
0x3df5	0x170	0x200	16741	4165	/bin/dd
0x8fe4	0x38c	0x388	38648	96f8	(TOTALS)

Section sizes for /bin/ls in System V format:

section	size	addr
.interp	21	4194704
.note.ABI-tag	24	4194728
.hash	624	4194752
.dynsym	2088	4195376
.rodata	1472	4216448
.data	80	5267456
.eh_frame	1624	5267536
.dynamic	384	5269160
.ctors	16	5269544
.dtors	16	5269560
.jcr	8	5269576
.got	576	5269584
.bss	528	5270176
.comment	686	0
Total	27110	

10 Object file strings

The object file strings utility displays the printable character strings contained in the specified files. The files can be object files, archive files, or arbitrary data files.

The user can control the utility output by specifying the minimum length of strings to be displayed.

If no input files are specified, the utility attempts to read from the standard input.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the object file strings utility from a command line, type:

```
arm-strings [--all] [--bytes=number] [--encoding=encoding] [--help]
    [--print-file-name] [--radix=radix] [--version] [-number]  [-a]
    [-e encoding]  [-f]  [-h]  [-o]  [-n number] [-t=radix] [-v] [file...]
```

10.1 Options

The object file strings utility supports the following options:

`--all`

For ELF objects, scan the entire file for printable strings.

`--bytes=`*`number`*

Print a contiguous character sequence only if it is at least the specified number of characters long. The default length is 4 characters.

`--encoding=`*`encoding`*

Select the character encoding to be used while searching for strings. The possible values are:

s – Single 7-bit-byte characters (ASCII, ISO 8859) (default)

S – Single 8-bit-byte characters

l – 16-bit little-endian

b – 16-bit big-endian

L – 32-bit little-endian

B – 32-bit big-endian

`--help`

Display command usage info and exit.

`--print-file-name`

Print the name of the file before each string.

`--radix=`*`radix`*

Print the file-relative offset of the string before each string. The offset is displayed in the specified radix.

The possible values are:

d – Decimal

o – Octal

x – Hexadecimal

`--version`

Display utility version identifier and exit.

`-number`

Equivalent to `--bytes=`*`number`*.

`-a`

Equivalent to `--all`.

`-e encoding`

Equivalent to `--encoding`.

`-f`

Equivalent to `--print-file-name`.

`-h`

Equivalent to `--help`.

`-n number`

Equivalent to `--bytes=`*`number`*.

`-o`

Equivalent to `-t o`.

`-t radix`

Equivalent to `-radix`.

`-v`

Equivalent to `--version`.

10.2 Examples

Strings in `/bin/ls`:

```
$ arm-strings /bin/ls
```

Strings in all sections of `/bin/ln`:

```
$ arm-strings -a /bin/ln
```

Strings in all sections of `/bin/cat`, prefixed with filename and file-relative offset:

```
$ arm-strings -a -f -t x /bin/cat
```

11 Object file stripper

The object file stripper utility removes information from the specified files. The files can be object files or archive files.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the object file stripper utility from a command line, type:

```
arm-strip [--discard-all] [--discard-locals] [--help]
    [--input-target=objformat] [--keep-symbol=symbolname]
    [--only-keep-debug] [--output-file=filename] [--output-target=objformat]
    [--preserve-dates] [--remove-section=sectionname] [--strip-all]
    [--strip-debug] [--strip-symbol=symbolname] [--strip-unneeded]
    [--version] [--wildcard] [-I objformat]  [-K symbolname]
    [-N symbolname] [-O objformat]  [-R sectionname]  [-S]  [-V]  [-X]  [-d]
    [-g]  [-h] [-o outputfile]  [-p]  [-s]  [-w]  [-x]  file...
```

11.1 Options

The object file stripper utility supports the following options:

`--discard-all`

Remove all non-global symbols from the object file.

`--discard-locals`

Remove compiler-generated local symbols from the object file.

`--help`

Display command usage info and exit.

`--input-target=`*`objformat`*

This option is accepted for compatibility with the GNU object file stripper, but is ignored.

`--keep-symbol=`*`symbolname`*

Keep the specified symbol even if it would otherwise be stripped. This option can be specified more than once.

`--only-keep-debug`

Remove all information from the object file except for the debugging information.

`--output-file=`*`filename`*

Write the output file to a new file with the specified name. The default is to overwrite the input file.

`--output-target=`*`objformat`*

Specify file format of the stripped object file. The possible values are:

- `ETF_ELF` – ELF object (default)
- `ETF_BINARY` – Raw binary
- `ETF_IHEX` – Object encoded in Intel hex format
- `ETF_NONE` – Unknown object format
- `ETF_SREC` – Object encoded as S-records

`--preserve-dates`

Set the access and modification times of the object file to the same as those of the input.

`--remove-section=`*`sectionname`*

Remove the specified section from the object file.

`--strip-all`

Remove all symbols from the object file.

`--strip-debug`

Remove debugging information from the object file.

`--strip-symbol=`*`symbolname`*

Remove the specified symbol even if it would otherwise be kept. This option can be specified more than once.

`--strip-unneeded`

Remove all symbols that are not needed for relocation processing.

`--version`

Display the utility version identifier and exit.

`--wildcard`

Use shell-style patterns to specify symbols. The following meta-characters are recognized in patterns:
! If this is the first character of the pattern, invert the sense of the pattern match
* Match any string of characters in a symbol name
? Match zero or one character in a symbol name
[Mark the start of a character class
\ Remove the special meaning of the next character in the pattern
] Mark the end of a character class

`-I objformat`

Equivalent to `--input-target`.

`-K symbolname`

Equivalent to `--keep-symbol`.

`-N symbol`

Equivalent to `--strip-symbol`.

`-O objformat`

Equivalent to `--output-target`.

`-R sectionname`

Equivalent to `--remove-section`.

`-S`

Equivalent to `--strip-debug`.

`-V`

Equivalent to `--version`.

`-X`

Equivalent to `--discard-locals`.

`-d`

Equivalent to `--strip-debug`.

`-g`

Equivalent to `--strip-debug`.

`-h`

Equivalent to `--help`.

`-o filename`

Equivalent to `--output-file`.

`-p`

Equivalent to `--preserve_dates`.

`-s`

Equivalent to `--strip-all`.

`-w`

Equivalent to `--wildcard`.

`-x`

Equivalent to `--discard-all`.

12 C++ filter

The C++ filter utility translates encoded C++ symbol names to human-readable form.

This utility can be used in two ways:

- As a filter, reading encoded names from the standard input, and writing the translated names to the standard output.
- As a command-line translator, reading the encoded names that are passed as command arguments, and writing the translated names to the standard output.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the C++ filter utility from a command line, type:

```
arm-c++filt [--format=scheme] [--help] [--no-params]
    [--no-strip-underscores] [--strip-underscores] [--version] [-V]
    [-_]  [-n]  [-p]  [-s scheme]  [encoded-names...]
```

12.1 Options

The C++ filter utility supports the following options:

`--format=`*scheme*

Select encoding scheme to use. The possible values are:

`arm` – Use encoding from *C++ Annotated Reference Manual* (default)

`auto` – Determine encoding from input

`gnu-v2` – Use encoding from GNU C++ compiler, version 2

`gnu-v3` – Use encoding from GNU C++ compiler, version 3

`--help`

Display command usage info and exit.

`--no-params`

This option is recognized but ignored.

13 Address Converter

The address converter utility translates the specified program addresses to their corresponding source file names and line numbers in an object file.

The addresses must be in hexadecimal format. If no addresses are specified, the utility attempts to read them from the standard input.

By default the utility uses a.out as the ELF object file. A different object file can be specified as a command argument.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the address converter utility from a command line, type:

```
arm-addr2line [--demangle] [--exe=pathname] [--functions] [--help]
    [--section=sectionname] [--target=target]  [--version]  [-C]  [-H]
    [-V]  [-b target] [-e pathname] [-f]  [address...]
```

13.1 Options

The address converter utility supports the following options:

`--demangle`

Demangle C++ names.

`--exe=pathname`

Use the specified ELF object to translate addresses. The default is `a.out`.

`--functions`

Display function names in addition to the file and line number information.

`--help`

Display command usage info and exit.

`--section=sectionname`

Treat address values as offsets into the specified section.

`--target=target`

This option is accepted for compatibility with the GNU address converter, but is ignored.

`--version`

Display the utility version identifier and exit.

`-C`

Equivalent to `--demangle`.

`-H`

Equivalent to `--help`.

`-V`

Equivalent to `--version`.

`-b target`

Equivalent to `--target`.

`-e pathname`

Equivalent to `--exe`.

`-f`

Equivalent to `--functions`.

13.2 Output formats

`-f` is not specified, for each address the utility prints the corresponding file name and line number on a single line.

If `-f` is specified, for each address the utility prints the function name corresponding to the address on one line, and the associated file name and line number on the following line.

NOTE The file name and line number are printed as `filename:linenumber`.

If a file or function name cannot be determined, the address converter prints a question mark in its place. If a line number cannot be determined, a zero is printed in its place.

13.3 Examples

Map address 080483c4 in the default executable a.out to a source file name and line number:

```
% addr2line 080483c4
```

Map address 080483c4 in an executable helloworld:

```
% addr2line -e helloworld 080483c4
```

Use the address converter as a filter, reading addresses from the standard input:

```
% addr2line
```

Print the function name for an address along with the source file and line number:

```
% addr2line -f 080483c4
```

14 ELF file viewer

The ELF file viewer utility displays information on the contents of the specified ELF object files.

NOTE This utility returns 0 on success, or greater than 0 if an error occurs.

Command

To start the ELF file viewer utility from a command line, type:

```
arm-readelf [--all] [--debug-dump=info] [--dyn-sims] [--dynamic]
    [--file-header] [--headers] [--help] [--hex-dump=(number|name)]
    [--histogram] [--program-headers] [--relocated-dump=(number|name)]
    [--relocs] [--sections] [--section-groups] [--section-headers]
    [--segments] [--string-dump=(number|name)] [--symbols] [--syms]
    [--use-dynamic] [--version] [--wide]  [-D]  [-I]  [-S]  [-W]  [-a]
    [-d]  [-e]  [-g] [-h]  [-l]  [-p (number|name)]  [-r]  [-s]  [-t]  [-v]
    [-x (number|name)]
```

14.1 Options

The ELF file viewer utility supports the following options:

`--all`

Equivalent to setting the following options: `-h -l -S -s -r -d -V -I`

`--debug-dump=`*`info`*

Display the contents of DWARF2 debug sections. The possible values are:

```
All – Dump all debug sections
abbrev – .debug_abbrev
abbrev-dwo – .debug_abbrev.dwo
aranges – .debug_aranges
info – .debug_info
info.dwo – .debug_info.dwo
types     – .debug_types
types.dwo – .debug_types.dwo
line – .debug_line
line.dwo – .debug_line.dwo
loc – .debug_loc
loc.dwo –     .debug_loc.dwo
frames – .debug_frame
macro     –.debug_macinfo
ranges – .debug_ranges
pubnames – .debug_pubnames
pubtypes – .debug_pubtypes
gnu_pubnames –     .debug_gnu_pubnames
gnu_pubtypes –     .debug_gnu_pubtypes
```

```
str – .debug_str
str.dwo –     .debug_str.dwo
str_offsets.dwo – .debug_str_offsets.dwo
cu_index – .debug_cu_index
tu_index – .debug_tu_index
--dyn-syms
```

Display dynamic symbol table.

`--dynamic`

Display dynamic section (if present).

`--file-header`

Display ELF file header.

`--headers`

Equivalent to setting the following options: `-h -l -S`

`--help`

Display command usage info and exit.

`--hex-dump=(`*`number`*`|`*`name`*`)`

Dump contents of the specified section as bytes.

`--histogram`

Display histogram of bucket list lengths.

`--section-details`

Display section details.

`--program-headers`

Display program headers.

`--relocated-dump=(`*`number`*`|`*`name`*`)`

Dump contents of the specified section as relocated bytes.

`--relocs`

Display relocations (if present).

`--section-groups`

Display section groups.

`--section-headers`

Display section header.

`--sections`

Alias for `--section-headers`.

`--segments`

Alias for `--program-headers`.

`--string-dump=(`*`number`*`|`*`name`*`)`

Dump contents of the specified section as character strings.

`--symbols`

Alias for `--syms`.

`--syms`

Display symbol table.

`--use-dynamic`

Use dynamic section info when displaying symbols.

`--version`

Display the utility version identifier and exit.

`--wide`

Allow output width to exceed 80 characters. Ignored, since wide mode is on by default.

`-D`

Equivalent to `--use-dynamic`.

`-I`

Equivalent to `--histogram`.

`-S`

Equivalent to `--section-headers`.

`-V`

Equivalent to `--version-info`.

`-W`

Equivalent to `--wide`.

`-a`

Equivalent to `--all`.

`-d`

Equivalent to `--dynamic`.

`-e`

Equivalent to `--headers`.

`-g`

Equivalent to `--section-groups`.

`-h`

Equivalent to `--file-header`.

`-p`

Equivalent to `--program-headers`.

`-n`

Equivalent to `-notes`.

`-p (number|name)`

Equivalent to `--string-dump`.

`-r`

Equivalent to `--relocs`.

`-s`

Equivalent to `--syms`.

`-t`

Equivalent to `--section-details`.

`-v`

Equivalent to `--version`.

`-w[FLRafilmoprs]`

Equivalent to `--debug_dump`. The possible values are:

```
a – .debug_abbrev
r – .debug_aranges
i – .debug_info
l – .debug_line
o – .debug_loc
f – .debug_frame
m – .debug_macinfo
R – .debug_ranges
p – .debug_pubnames
t – .debug_pubtypes
s – .debug_str
```