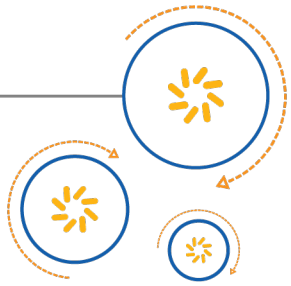
Qualcomm Technologies, Inc.



Qualcomm® Snapdragon™ LLVM ARM Linker

User Guide

80-VB419-102 Rev. G

March 26, 2018

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

Contents

Figures

Tables

1 Introduction

This document describes the Qualcomm® Snapdragon™ ARM LLVM linker, including commands and options as well as linker map and script files.

1.1 Conventions

Square brackets enclose optional items (for example, `[label]`).

The vertical bar character, `|`, indicates a choice of items (for example, `add|del`).

Function declarations, function names, type declarations, attributes, and code samples appear in a different font (for example, `cp armcc armcpp`).

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

1.2 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.

If you have comments or suggestions on how to improve the linker, submit them to developer.qualcomm.com/llvm-forum.

2 Using the linker

The Snapdragon ARM linker merges object and archive files into executable images, relocates program data to its final location in memory, and resolves symbol references within and between files. The linker accepts files built for ARM and Arch64 architecture.

2.1 Linker command

To start the linker from a command line, enter the following:

```
ld.qcld [ option …][ input_object_file…]
```

NOTE The linker is invoked like the GNU linker, and it supports the most commonly used options in the GNU linker.

Arguments that are specified on the command line without an option switch are treated as the names of input object files. If the linker does not recognize an input file as an object file, it treats the file as a linker script.

The linker command can read arguments from a text file rather than from the command line. When starting the linker, use an @ symbol followed by the file name:

```
ld.qcld @file
```

NOTE The `arm-link` command name is a symbolic link to `ld.qcld`, and it can be used when starting the linker.

To specify the target architecture in the command line, use`-march`. For example, to specify the AArch64 architecture (instead of using the default target of ARM):

```
ld.qcld -march=aarch64 other_arguments …
```

NOTE In the Snapdragon LLVM ARM compiler, the Clang driver recognizes the `-fuse-ld=qcld` option as a directive to use `ld.qcld` as the system linker.

2.2 Linker options

The linker accepts command options prefixed with a hyphen,-, on the command line. To list all linker options, enter the following:

```
ld.qcld -help
```

Table 2-1 General options

Function	Description
-build-id=<*value*>	Request the creation of the .note.gnu.build-id ELF note section.
--build-id	Request the creation of the .note.gnu.build-id ELF note section.
-d	Assign space for common symbols.
-dc	Assign space for common symbols. This is an alias of -d.
-dp	Assign space for common symbols. This is an alias of -d.
-o <*value*>	Specify a file path to write output.
--repository-version	Print the linker repository version.
-sysroot <*value*>	Specify the system root directory, overriding the configure-time default. This option is only useful in Linux.
--version	Print the linker version.
--help	Print option help.

Table 2-2 ARM-only linker options

Function	Description
-fix-cortex-a53-843419	Fix the cortex a53 errata 843419.
--fix-cortex-a8	Fix the cortex A8 errata.
-fropi	Enable PC-relative references to address code and read-only data.
-frwpi	Enable read-write data access relative to a static base register.
--no-fix-cortex-a8	Do not fix the cortex A8 bug.

Table 2-3 ARM/AArch64 linker options

Function	Description
-compact	Create a smaller output file by computing the file offset based on the previous section. The physical address of the section should be congruent to the offset from the segment that the section is really in. The loader must be able to support such files.
--disable-bss-conversion	Do not convert a BSS section to a non-BSS section when BSS/non-BSS sections are mixed.

Table 2-3 ARM/AArch64 linker options (cont.)

Function	Description
`--enable-bss-mixing`	Enable the mixing of BSS and non-BSS sections.
`--use-mov-veneer`	Use `movt`/`movw` to load an address in veneers with absolute relocation.

Table 2-4 Ignored options

Function	Description
`--add-needed`	Deprecated.
`--copy-dt-needed-entries`	Add the dynamic libraries mentioned to `DT_NEEDED`.
`--gpsize=<value>`	Set the maximum size of objects to be optimized using `GP`.
`--nmagic`	Turn off the page alignment of sections, and disable linking against shared libraries.
`--no-add-needed`	Deprecated.
`--no-copy-dt-needed-entries`	Turn off the effect of `--copy-dt-needed-entries`.
`--no-omagic`	Negate most of the effects of the `-N` option, and disable linking with shared libraries.
`--omagic`	Set the text and data sections to be readable and writable. Do not page-align the data segment. Disable linking against shared libraries.
`-Qy`	For SVR4 compatibility, this option is ignored.
`-R <value>`	Read symbol names and addresses from the filename.

Table 2-5 Diagnostic options

Function	Description
`-color <value>`	Enable color output for diagnostics.
`-cref`	Print the references for a symbol or section.
`--error-style <value>`	Specify an error style: LLVM/GNU.
`--fatal-warnings`	Enable fatal warnings.
`--no-fatal-warnings`	Disable fatal warnings.
`--gc-cref <value>`	Print references for a symbol or section when garbage collection is enabled.
`--no-warn-mismatch`	Do not warn when incompatible files are passed to the linker.
`-trace <value>`	Allow tracing of files, relocations, symbols, link time optimization (LTO), or threads.
`-t`	Print all files processed by the linker.
`--verbose`	Enable verbose output.

Table 2-5 Diagnostic options (cont.)

Function	Description
`-verify-options <value>`	Verify internal linker computations. This option currently supports relocation.
`--warn-common`	Warn when a common symbol is combined with another common symbol or a symbol definition. **NOTE** `-warn-common` is parsed only, and generates no warning messages.

Table 2-6 Dynamic library options

Function	Description
`-Bgroup`	Enable the runtime linker to handle lookups in the object and its dependencies; to be performed only within the group.
`-Bsymbolic-functions`	Bind references to global functions to the definition within the shared library.
`-Bsymbolic`	Bind references to global symbols to the definition within the shared library.
`-fPIC`	Enable PIC mode.
`-g`	Enable debug output when building shared libraries or executables.
`--hash-size <value>`	Specify a hash size when creating the hash sections for the dynamic loader.
`-hash-style < sysv | gnu | both >`	Specify a hash style when creating the hash sections for the dynamic loader: ■ `sysv` – Classic ELF .hash section ■ `gnu` – New-style GNU .gnu.hash section ■ `both` – ELF and GNU hash tables
`--no-warn-shared-textrel`	Do not warn if the linker adds a `DT_TEXTREL`.
`--warn-shared-textrel`	Warn if the linker adds a `DT_TEXTREL`.
`-soname=<value>`	Set the internal `DT_SONAME` to the specified name.

Table 2-7 Dynamic library/executable options

Function	Description
`-dynamic-linker <value>`	Set the path to the dynamic linker.
`-export-dynamic`	Add all symbols to the dynamic symbol table when creating executables.
`-force-dynamic`	Force the output file to include dynamic sections and force the linker to create dynamic sections. This makes a dynamic executable, `-fPIC`. Set the relocation model to PIC.
`-rpath-link <value>`	Specify the first set of directories to search. This option has no effect and is provided for compatibility with the GNU linker.

Table 2-7 Dynamic library/executable options (cont.)

Function	Description
`-rpath <value>`	Add a directory to the runtime library search path.
`-export-dynamic-symbol <value>`	Export the specified defined symbol to the dynamic symbol table. Garbage collection is disabled for the specified symbol and for any references made by this symbol.

Table 2-8 Executable options

Function	Description
`-emit-relocs`	Generate GNU linker-compatible relocations, which use the relocation target address as the VMA of the executable instead of the offset within sections.
`-e <value>`	Name of the entry point symbol.
`-fini <value>`	Specify the function called when an executable or shared object is unloaded: set `DT_FINI`. The default function name is `_fini`.
`-init <value>`	Specify the function called when an executable or shared object is loaded: set `DT_INIT`. The default function name is `_init`.
`--library-path <value>`	Specify the directory to search for libraries or linker scripts.
`--library=<namespec>`	Specify a library to use.
`-L <dir>`	Directory to search for libraries or linker scripts.
`-l<libName>`	Root name of the library to use.
`-no-whole-archive`	Restore the default behavior of loading archive members.
`--noinhibit-exec`	Retain the executable output file whenever it is still usable.
`-nostdlib`	Disable the default search path for libraries.
`-whole-archive`	Link into the program every object file that is contained in the specified archives. Specify the archives by placing them between the `wholearchive` and `no-whole-archive` options.
`-Y <value>`	Add a path to the default library search path.

Table 2-9 Extended options

Function	Description
`--align-segments`	Align segments to page boundaries.
`-copy-farcalls-from-file <value>`	Copy far calls instead of using trampolines.
`--disable-new-dtags`	Disable new dynamic tags.
`-emit-relocs-llvm`	Emit relocations sections.
`--enable-new-dtags`	Enable new dynamic tags.
`--no-align-segments`	Do not align segments to page boundaries.
`-no-emit-relocs`	Do not emit relocations in the output file. This option also applies to `--emit-relocs-llvm`.

Table 2-9 Extended options (cont.)

Function	Description
`-no-reuse-trampolines-file <value>`	Do not reuse trampolines for symbols specified in a file.
`--no-verify`	Do not verify the link output.
`-rosegment`	Put read-only non-executable sections in their own segment.
`-z <value>`	Specify extended or nonstandard options. Supported values are listed in the `-z` option table.

Table 2-10 Symbol defining options

Function	Description
`--symdef-file <value>`	Emit the SymDef file.
`-symdef`	Output the SymDef file to the console.

Table 2-11 Target options

Function	Description
`-march <arm | aarch64>`	Specify the target architecture. The default is `ARM`.
`-mcpu <mcpu>`	Specify the ARM/AArch64 processor version. When the target is cortex-m0, `-mcpu=cortex-m0` must be specified.
`-mllvm <option>`	Options to pass to the LLVM.
`-mtriple <triple>`	Specify the architecture, ABI, and operating system of the linker output file. For example, `-mtriple=arm-gnu-linux`. When this option is used, the`-march` option must be specified with a NULL value, i.e.,`-march=` **NOTE** We do not recommend using `-mtriple`. Instead, use `-march` to specify the target architecture.
`-m <emulation>`	Select the target emulation.

Table 2-12 LTO options

Function	Description
`-flto-options <value>`	Specify various options with LTO.
`-flto-options=(codegen=option [ ,option …])`	Pass the specified `-mllvm` compiler options to the compiler during LTO.
`-flto-use-as`	Use the standalone assembler instead of the integrated assembler for LTO.
`-flto`	If the embedded bitcode section is present for linking, the linker uses LTO. There is no effect on a simple ELF binary. For options and the method of emitting embedded bitcode sections inside an ELF file, see the *Qualcomm® Snapdragon™ LLVM ARM Compiler User Guide (80-VB419-99)*.

Table 2-12 LTO options (cont.)

Function	Description
`-save-temps`	Save the temporary files produced by LTO. To display the path name of the file, use `-trace=lto`.
`--exclude-lto-filelist <value>` `--include-lto-filelist <value>`	Specify a file that contains a list of files (which can include glob patterns) whose embedded bitcode sections (if any) are to be used with LTO instead of traditional linking with contained ELF sections. **NOTE** These exclude/include options are mutually exclusive.

Table 2-13 Link time speedup options

Function	Description
`-no-threads`	Disable threads at link time (the default behavior).
`-threads`	Enable threads at link time.
`--thread-count <value>`	Specify the number of threads for all linker operations.

Table 2-14 Map options

Function	Description
`--color-map`	Color the map file.
`-MapDetail <value>`	Detail information in the map file. Valid values are `show-strings` or `absolute-path`.
`-MapStyle <value>`	Dump the output layout to the map file in YAML/text form. Valid values are `gnu`, `yaml`, and `llvm`.
`-Map <value>`	Dump the output layout to the map file. The value is the output file name.
`-M`	Emit the map file in the standard output.

Table 2-15 Optimization options

Function	Description
`-best-fit-section <value>`	Reorder the section content to minimize its alignment requirement. This option minimizes the gap created by the alignment requirement so that the section size is reduced. The value is the name of the section that is processed. This option can be used multiple times with different section names.
`--eh-frame-hdr`	Create an EH frame header section for faster exception handling.
`-gc-sections`	Delete all unused input sections from the output file (garbage collection). Display the deleted sections with`--print-gcsections`.
`-no-gc-sections`	Disable garbage collection.
`--no-merge-strings`	Disable string merging.

Table 2-15 Optimization options (cont.)

Function	Description
`--no-trampolines`	Disable trampolines.
`-print-gc-sections`	Print sections that are garbage-collected. Use this option with `--gc-sections`.

Table 2-16 Output control options

Function	Description
`-Bdynamic`	Link against a dynamic library.
`-dynamic`	Create a dynamic executable (default).
`-pie`	Create a PIE executable.
`-r`	Create a relocatable object file.
`-shared`	Create a dynamic library.
`-static`	Create a static executable.

Table 2-17 Script options

Function	Description
`-default-script <value>`	Specify the default linker script. The script is loaded after all other linker options are processed.
`-dynamic-list <value>`	Specify a list of symbols that, if present, are exported.
`--exclude-libs <value>`	Specify a list of archive libraries from which symbols should not be automatically exported.
`-extern-list <value>`	Specify a list of symbols that exist as external dependencies.
`--map-section <value>`	Specify an input section that maps to an output section.
`-print-stats <value>`	Display linker operation statistics.
`-section-start <value>`	Specify a hexadecimal integer as the virtual output section address for a specified section. This option can be repeated multiple times.
`-Tbss <value>`	Specify an address for the `.bss` section.
`-Tdata <value>`	Specify an address for the `.data` section.
`-Ttext-segment <value>`	Specify an address for the `.text-segment` segment.
`-Ttext <value>`	Specify an address for the `.text` section.
`-T <value>`	Specify the linker script file. The order in which the linker script is loaded (relative to when other linker options are processed) is determined by the order of the options specified on the command line.
`-version-script <value>`	Use the linker script as a version script.

Table 2-18 Symbol options

Function	Description
`--demangle`	Demangle C++ symbols.
`--discard-all`	Discard all local symbols.
`--discard-locals`	Discard temporary local symbols.
`--no-demangle`	Do not demangle C++ symbols.
`-portable <value>`	Specify the symbol to be portable.
`--strip-all`	Omit all symbol information from the output.
`--strip-debug`	Omit all debug information from the output.
`-wrap <value>`	Create a wrapper for the specified symbol. Resolve any undefined references to the symbol as references to `__wrap_symbol`. Resolve any undefined references to `__real_symbolas` as references to symbol. This option can be repeated multiple times.

Table 2-19 Symbol resolution options

Function	Description
`--allow-multiple-definition`	Allow multiple definitions of a symbol in the files being linked. By default, this option is disabled. Only the first definition is used; additional definitions are ignored.
`-allow-shlib-undefined`	When creating executables, allow undefined symbols from a dynamic library.
`--as-needed`	By default, a `DT_NEEDED` entry is added for every shared library specified on the command line. This option enables `DT_NEEDED` entries to be added only if the shared library is used to resolve symbols referenced from object files. The`--no-as-needed` option restores the default behavior.
`-defsym symbol=<expression>`	Create a global symbol in the output file that contains the absolute address given by expression. This option can be used multiple times. The expression is limited to a hexadecimal constant or existing symbol name, with the optional addition or subtraction of a second hexadecimal constant or symbol.
`-end-group`	Resolve circular symbol references in the specified files and libraries. Because the linker must repeatedly search the libraries to resolve circular references, this option significantly affects the linker performance. It should only be used when circular symbol references are present. **NOTE** Specify files and libraries by placing them between the `-(` and `-)` options, or the `start-group` and `end-group` options.
`--no-allow-shlib-undefined`	Do not allow undefined symbols from dynamic libraries when creating executables.
`--no-as-needed`	Restore the default behavior of adding `DT_NEEDED` entries.

Table 2-19 Symbol resolution options (cont.)

Function	Description
`-no-undefined`	Report unresolved symbol references from regular object files.
`-start-group`	Start a group.
`--use-shlib-undefines`	Resolve undefined symbols from dynamic libraries.
`-u <value>`	Include the specified symbol in the output file as an undefined symbol. This option forces additional object files to be linked to a program. This option can be repeated multiple times.
`--warn-once`	Warn every undefined reference only once.
`--unresolved-symbols=<value>`	Determine how to handle unresolved symbols. The four possible values are: ■ `ignore-all` – Do not report any unresolved symbols. ■ `report-all` – Report all unresolved symbols (default). ■ `ignore-in-object-files` – Report unresolved symbols contained in shared libraries, but ignore them if they come from regular object files. ■ `ignore-in-shared-library` – Report unresolved symbols from regular object files, but ignore them if they come from shared libraries. This value is useful when creating a dynamic binary and all the shared libraries referenced are included on the linker's command line.

Table 2-20 -z option table

Function	Description
`-z combreloc`	Merge dynamic relocatables into one section and sort them.
`-z nocombreloc`	Do not merge dynamic relocatables into one section.
`-z common-page-size=SIZE`	Set the common page size to *SIZE*.
`-z defs`	Report unresolved symbols in object files.
`-z execstack`	Mark the executable as requiring an executable stack.
`-z global`	Make symbols in DSO available to subsequently loaded objects.
`-z initfirst`	Mark DSO to be initialized first at runtime.
`-z lazy`	Mark an object as lazy runtime binding (default).
`-z now`	Mark an object as non-lazy runtime binding.
`-z max-page-size=SIZE`	Set the maximum page size to *SIZE*.
`-z muldefs`	Allow multiple definitions.
`-z nocopyreloc`	Do not create copy relocatables.
`-z nodelete`	Mark DSO as non-deletable at runtime.
`-z noexecstack`	Mark the executable as not requiring an executable stack.
`-z relro`	Create a `RELRO` program header.
`-z norelro`	Do not create a `RELRO` program header.

Table 2-20 -z option table (cont.)

Function	Description
`-z origin`	Mark an object as requiring immediate `$ORIGIN` processing at runtime.
`-z compactdyn`	Create a dynamic section with a minimal number of entries.

2.3 Link time optimization (LTO)

LTO is a type of code optimization performed by the linker. In LTO, the linker drives the compiler to perform inter-procedure optimizations and apply global program information to improve the code optimizations performed by the compiler.

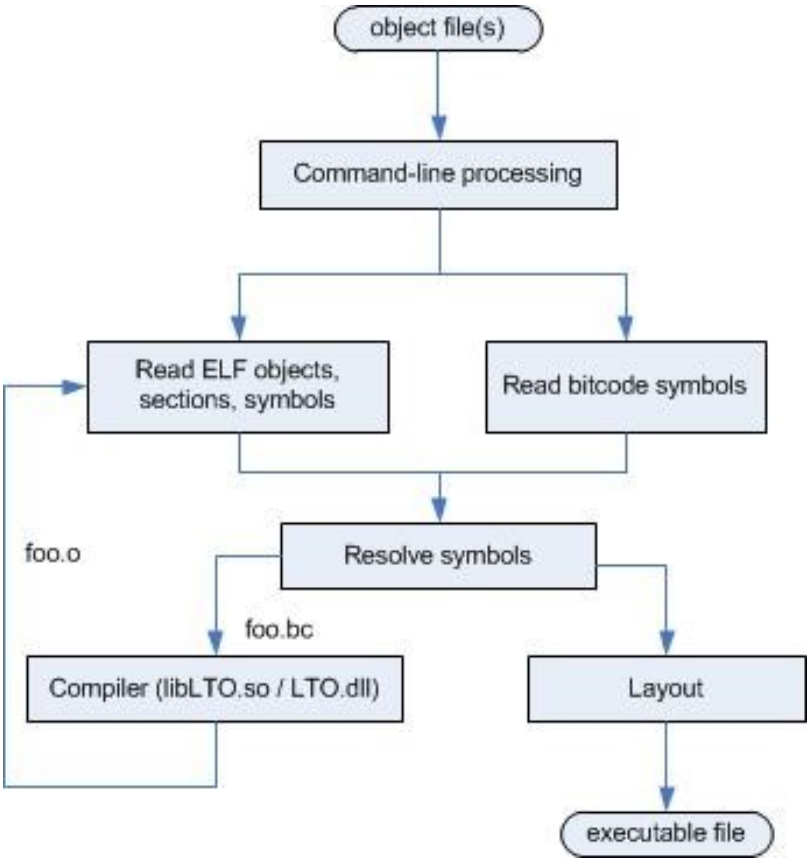


Figure 2-1 LTO flow performed by the linker

If any linker inputs are in LLVM bitcode format, the linker automatically enables LTO. For best LTO results, all the linker input files should be in LLVM bitcode format. This can be done by compiling all the source files with the `-flto` compiler option. For more information on compiling files for LTO, see *Qualcomm® Snapdragon™ LLVM ARM Compiler User Guide (80-VB419-99)*.

Use `-flto-options=codegen` to pass the `-mllvm` options to the compiler as part of LTO.

The linker uses the LLVM integrated assembler to generate bit code directly to a binary object file. To preserve the assembler file and see where it is stored, use `save-temps` and `-t=lto`.

2.4 Link ROPI/RWPI objects

The ARM processor supports the following relocation models:

- Read-only position independence (ROPI) – Code and read-only data is accessed PC-relative. Offsets between code and RO data sections are known at static link time.
- Read-write position independence (RWPI) – Read-write data is accessed relative to a static base register (R9). Offsets between all writable data sections are known at static link time.

These modes are typically used by baremetal systems or small real-time operating systems. They avoid the need for a dynamic linker because the only initialization required is setting the static base register to an appropriate value for RWPI code. They also minimize the size of the writable portion of the executable for systems with limited RAM.

Enable additional Clang flags with`-fropi, -fwrpi`.

Linker behavior

The linker creates an error if the relocation of RWPI data falls into different segments. The *`_RWPI_BASE`* variable is provided at the start of the segment used for SBREL32 relocation. All other errors are captured by the Attribute parser when mixing RWPI and non RWPI objects.

LTO behavior

When`-frwpi` or`-fropi` is passed to the linker, the linker passes the command line option to the LTO backend using the relocation model.

NOTE This relocation is not supported with NDK builds.

2.5 Region tables

The region tables feature removes the restriction that prevents the mixing of `NOBITS` and `PROGBITS` sections in the same segment. This feature is available on the 32-bit ARM processor.

The linker creates a table of mixed `NOBITS` and `PROGBITS` sections in the same segment. The loader, which loads the program, iterates over the table to initialize all `NOBITS` sections to the default initialization value of zero.

The table is only emitted if a loadable segment has a linker script rule for it:

```
*(__region_table__)
```

The text map file produces detailed information of the region table, but the YAML map does not generate the information at this time.

3 Link maps

Link maps, optionally generated by the linker, provide information on how the files are linked. Each link map contains four sections:

- Archive
- Common symbols
- Linker script
- Memory map

The format per section is listed in the following table.

Table 3-1 Link map sections

Section	Section format
Archive	`archive_file (symbol_define_object_file) symbol_reference_object_file (symbol)`
Common symbols	`symbol size archive_file (symbol_define_object_file)`
Linker script	`linker_script_command`
Memory map	`output_section addr size input_section addr size object_file symbol addr ... ...`

A link map is generated as a text file. The file is specified on the command line with `-Map`.

Example link map file

The following link map file was generated by linking the C program for the AArch64 target architecture:

```
#include "stdio.h"
int common_var;
int main() {
   common_var = 1;
   printf("var = %d\n", common_var);
   return 0;
}
```

This map file has been shortened for readability.

```
Archive member included because of file (symbol)
/prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-gnu-4.9.0/
aarch64-linux-gnu/libc/usr/lib64/libc_nonshared.a(elf-init.oS)
    /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
```

```
gnu-4.9.0/aarch64-linux-gnu/libc/usr/lib/../lib64/crt1.o (__libc_csu_init)

Allocating common symbols
Common symbol    size    file

common_var    0x4    /tmp/t-f3306e.o

Linker Script and memory map
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/aarch64-linux-gnu/libc/usr/lib/../lib64/crt1.o[]
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/aarch64-linux-gnu/libc/usr/lib/../lib64/crti.o[]
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/lib/gcc/aarch64-linux-gnu/4.9.0/crtbegin.o[]
LOAD /tmp/t-f3306e.o[]
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/lib/gcc/aarch64-linux-gnu/4.9.0/../../../../aarch64-linux-gnu/
lib/../lib64/libgcc_s.so[]
START GROUP
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/aarch64-linux-gnu/libc/lib64/libc.so.6[]
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/aarch64-linux-gnu/libc/usr/lib64/libc_nonshared.a(elf-init.oS) []
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/aarch64-linux-gnu/libc/lib/ld-linux-aarch64.so.1[]
END GROUP
SKIPPED /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/lib/gcc/aarch64-linux-gnu/4.9.0/../../../../aarch64-linux-gnu/
lib/../lib64/libgcc_s.so (ELF)
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/lib/gcc/aarch64-linux-gnu/4.9.0/crtend.o[]
LOAD /prj/llvm-arm/home/common/build_tools/gcc-fsf-glibc-aarch64-linux-
gnu-4.9.0/aarch64-linux-gnu/libc/usr/lib/../lib64/crtn.o[]

Linker scripts used (including INCLUDE command)/sysroot/aarch64-linux-gnu/
libc/usr/lib/../lib64/libc.so …
interp    0x238    0x1b # Offset: 0x238, LMA: 0x238…
.text    0x460    0x21c # Offset: 0x460, LMA: 0x460.text    0x460    0x48 /
sysroot/aarch64-linux-gnu/libc/usr/lib/../lib64/crt1.o
0x468        .text        0x468        $x        0x468        _start
0x498        $d    ...
.text  0x5c0  0x48    /tmp/t-57e15d.o        0x5c0        $x.0
       0x5c0        .text        0x5c0        main
…
.shstrtab  0x1288    0xf1
.symtab    0x1380    0x7f8
```

3.1 Archive section

The archive section of a link map lists each archive file that was accessed by the linker and the symbol reference that caused the archive file to be accessed. Each entry in the archive section contains the following items:

- The full pathname of the archive file accessed by the linker
- The name of the archived object file that defines the symbol (in parentheses)
- The full pathname of the object file that contains the symbol reference
- The name of the referenced symbol (in parentheses)

In the following example, the `__libc_csu_init` symbol is referenced in the `crt1.o` object file and defined in `_elf-init.oS`, which is stored in the `libc_nonshared.a` archive file:

```
/sysroot/aarch64-linux-gnu/libc/usr/lib64/libc_nonshared.a(elf-init.oS)
        /sysroot/aarch64-linux-gnu/libc/usr/lib/../lib64/crt1.o
(__libc_csu_init)
```

3.2 Common symbols section

The common symbols section of a link map lists the common symbols that were allocated in memory by the linker. Each entry in the common symbols section contains the following items:

- The name of the symbol
- The size of the memory area allocated for the symbol
- The full pathname of the archive file accessed by the linker
- The name of the archived object file that defines the symbol (in parentheses)

In the following example, the `common_var` symbol has size of 0x4 and is defined in the `t-57e15d.o` object file:

```
common_var 0x4 /tmp/t-57e15d.o 3
```

3.3 Linker script section

The linker script section of a link map lists the complete linker script that is specified for the link.

NOTE Linker scripts are optional. If a script is not specified on the linker command line, the link map does not include a linker script.

The following example shows the initial lines of a linker script section:

```
…
START GROUP
LOAD /sysroot/aarch64-linux-gnu/libc/lib64/libc.so.6
LOAD /sysroot/aarch64-linux-gnu/libc/usr/lib64/libc_nonshared.a(elf-init.oS)
LOAD /sysroot/aarch64-linux-gnu/libc/lib/ld-linux-aarch64.so.1
END GROUP
SKIPPED /sysroot/lib/gcc/aarch64-linux-gnu/4.9.0/../../../../aarch64-linux-
```

```
gnu/ lib/../lib64/libgcc_s.so (ELF)
LOAD /sysroot/lib/gcc/aarch64-linux-gnu/4.9.0/crtend.o
LOAD /sysroot/aarch64-linux-gnu/libc/usr/lib/../lib64/crtn.o
Linker scripts used (including INCLUDE command)
/sysroot/aarch64-linux-gnu/libc/usr/lib/../lib64/libc.so
```

RELATED INFORMATION

"Linker scripts" on page 26

3.4 Memory map section

The memory map section of a link map lists how symbols and assembly language sections are assigned to memory in the output file. The section lists one or more output sections, and each output section contains the following items:

- The output section, its start address, and section size
- Each input section that is mapped to the output section, including its start address, section size, and full pathname of the object file containing the section
- Each symbol defined in the input section, including its assigned value

In the following example, the `.text` output section has the start address of 0x468 and size of 0x21c. Multiple input sections (named `.text`) are mapped to this output section. Two sections are shown in this example. The first section has the start address of 0x468 and size of 0x48. The second section has the start address of 0x5c0 and size of 0x48. Following each section descriptor is a list of the symbols in the section.

```
.text   0x468   0x21c
.text   0x468   0x48   /sysroot/aarch64-linux-gnu/libc/usr/lib/../lib64/crt1.o
        0x468          .text
        0x468          $x
        0x468          _start
        0x498          $d
…
.text   0x5c0   0x48   /tmp/t-57e15d.o
         0x5c0          $x.0
        0x5c0          .text
        0x5c0          main
…
```

3.5 Link map in YAML format

If you use the MapStyle option to specify YAML-style output for the map file, the linker generates a YAML file instead of the text file that shows the memory map for the program. To derive statistics from the YAML map file produced by the linker, use the YAMLMapParser.py tool and include any of the following options:

- `-info=architecture` – List the architecture
- `sizes` – List the code and data sizes for the objects in the image
- `summarysizes` – List the code and data sizes of the image
- `totals` – List the total size of all objects in the image
- `unused` – List the sections eliminated from the image
- `unusedsymbols` – List the symbols eliminated from the image
- `--map` – Display the memory map of the images
- `--xref` – List the cross references between input sections
- `list` – Redirect the output to a file

The following table describes the sections of a YAML file.

Table 3-2 Sections of a YAML file

Section	Description
`Header`	Top level information of the program that was built
`Version Information`	Tools version information
`Archive Records`	Archive files pulled in by the linker
`Inputs`	Inputs that were used
`InputInfo`	Inputs that were used and not used
`Linker script used`	Linker script file that was used
`BuildType`	Type of build
`OutputFile`	Name of the output file
`EntryAddress`	Entry address for the image built
`CommandLine`	Command line information on how the linker was called
`CommandLineDefaults`	Various defaults applied in the linker
`OutputSections`	All output sections
`DiscardedComdats`	COMDAT C++ section groups that were discarded
`DiscardedSections`	Sections that were discarded by garbage collection
`DiscardedCommonSymbols`	Common symbols that were discarded
`LoadRegions`	Segments that the program loader will load
`CrossReferences`	Cross-reference table for the program

Example

```
a.c
int discardedcommon;
int main() { foo(); return 0;}
b.c
int foo() { return 0; }
clang -c 1.c 2.c -ffunction-sections -fdata-sections
llvm-ar cr lib2.a 2.o
ld.qcld 1.o lib2.a -MapStyle yaml -Map x.yaml --gc-sections -e main
```

4 Linker scripts

Linker scripts provide detailed specifications of how files are to be linked. They offer greater control over linking than is available using just the linker command options.

NOTE Linker scripts are optional. In most cases, the default behavior of the linker is sufficient.

Linker scripts control the following properties:

- ELF program header
- Program entry point
- Input and output files and searching paths
- Section memory placement and runtime
- Section removal
- Symbol definition

A linker script consists of a sequence of commands stored in a text file. The script file can be specified on the command line either with `-T`, or by specifying the file as an input files. The linker distinguishes between script files and object files and handles each accordingly.

To generate a map file that shows how a linker script controlled linking, use the `M` option.

Table 4-1 Linker script commands

Command	Description
`PHDRS`	
`SECTIONS`	Section mapping and memory placementELF program header definition
`ENTRY`	ELF program headerProgram execution entry point
`OUTPUT_FORMAT`	Parsed, but no effect on linking
`OUTPUT_ARCH`	Parsed, but no effect on linking
`SEARCH_DIR`	Add additional searching directory for libraries
`INCLUDE`	Include linker script file
`OUTPUT`	Define output filename
`GROUP`	Define files that will be searched repeatedly
`ASSERT`	Linker script assertion
`NOCROSSREFS`	Check cross references among a group of sections

4.1 Example script file

```
ENTRY (main)
SEARCH_DIR("./")
PHDRS
{
    CODE PT_LOAD ;
    DATA PT_LOAD ;
}
SECTIONS
{
  .text.qcldfn (0x2000) : { *(.text.qcldfn*) } : CODE
  . = ALIGN(0x1000);
  PROVIDE(__etext = .);
  __text_start = . + 0x1000 - 0x1000;
  .text : { EXCLUDE_FILE(*notused.o*) *(.text.*) } : CODE   .data :
{ *(.data.*) } : DATA
  .init : { KEEP (*(.init)) }
  . = SEGMENT_START(".bss", 0x80000);
  .bss : { *(.bss.*) }   __bss_start = .;
  __bss_end = .;
}
```

The `PHDRS` command defines two loadable ELF segments.

The `SECTIONS` command specifies how input sections are mapped to output sections, and where output sections are located in memory. Wildcard characters in the `SECTIONS` command indicate multiple input sections mapped to a single output section. A period indicates the current location counter and is assigned several different values in the `SECTIONS` command.

.text.qcldfn

All input sections beginning with `.text.qcldfn` are mapped to the `.text.qcldfn` output section. This output section is first in the `CODE` segment. The merged output section is located at virtual memory address 0x2000. The location counter then advances to the next 0x1000 address boundary past the end of the `.text.qcldfn` output section using the `ALIGN` directive.

The `__etext` symbol is defined with the location counter value of any unresolved symbol references for the symbol using the `PROVIDE` command.

The `__text_start` symbol is assigned the current location counter value plus any following arithmetic expression.

.text

All input sections beginning with `.text` are mapped to the `.text` output section. This output section is put into the `CODE` segment. The merged output section is located at the current location counter.

The `.text.qcldfn` section is not affected by this statement even though it matches the `.text.*` section name wildcard because it is already merged in the previous link script statement.

.data

The current location counter is assigned the base address of the `.data` output section with the `SEGMENT_START` directive. If undefined, a default value of 0x50000 is used.

All input sections beginning with `.data` are mapped to the `.data` output section. This section is put into the `DATA` segment. The merged output section is located at the current location counter as specified by the preceding statement in the script.

.init

All input sections beginning with `.init` are mapped to the `.init` output section. Because no segment is defined, available previous segments are used instead and the output is put into the `DATA` segment. The merged output section is located at the current location counter as specified by the preceding statement in the script.

.bss

The current location counter is assigned the base address of the `.bss` output section with the `SEGMENT_START` directive. If undefined, a default value of 0x80000 is used.

All input sections beginning with `.bss` are mapped to the `.bss` output section. The merged output section is located at the current location counter as specified by the preceding statement in the script. If garbage collection is enabled (with the `KEEP` directive), none of the input sections are removed from memory.

The `__bss_start` and `__bss_end` symbols are assigned the current location counter value.

4.2 Basic script syntax

Symbols

Symbol names must begin with a letter, underscore, or period. They can include letters, numbers, underscores, hyphens, or periods.

Comments

Comments can appear in linker scripts:

```
/*comment */
```

Strings

Character strings can be specified as parameters with or without delimiter characters.

Expressions

Expressions are similar to C, and support all C arithmetic operators. They are evaluated as type `long` or `unsigned long`.

Location counter

A period is used as a symbol to indicate the current location counter. It is used in the SECTIONS command only, where it designates locations in the output section:

```
. = ALIGN(0x1000); . = . + 0x1000;
```

Assigning a value to the location counter symbol changes the location counter to the specified value. The location counter can be moved forward by arbitrary amounts to create gaps in an output section. It cannot, however, be moved backwards.

Symbol assignment

Symbols, including the location counter, can be assigned constants or expressions:

```
__text_start = . + 0x1000;
```

Assignment statements are similar to C, and support all C assignment operators. Terminate assignment statements with a semicolon.

RELATED INFORMATION

"Expressions" on page 35

4.3 Script commands

The SECTIONS command must be specified in a linker script. All the other script commands are optional.

RELATED INFORMATION

"Example script file" on page 27
"Expressions" on page 35

4.3.1 PHDRS

```
PHDRS
{ name type [FILEHDR][PHDRS][AT (address)][FLAGS (flags)] }
```

The PHDRS script command sets information in the program header of an ELF output file.

- *name* – Specifies the program header in the SECTIONS command
- *type* – Specifies the program header type
- PT_LOAD – Loadable segment
- PT_NULL – Linker does not include section in a segment. No loadable section should be set to PT_NULL.
- PT_DYNAMIC – Segment where dynamic linking information is stored
- PT_INTERP – Segment where the name of the dynamic linker is stored
- PT_NOTE – Segment where note information is stored
- PT_SHLIB – Reserved program header type
- PT_PHDR – Segment where program headers are stored

The FILEHDR, PHDRS, and AT options are not supported. They are parsed but otherwise have no effect on linking.

The FLAGS option specifies the p_flags field in the ELF header. The following values can be used:

- PF_R – Read
- PF_W – Write
- PF_X – Execute

Multiple values can be specified in p_flags. For example, the value PF_R | PF_W specifies read/write.

NOTE If the sections in an output file have different flag settings than PHDRS, the linker chooses the least-restrictive settings for the output file.

More than one program header specification can be assigned to a given section. The following linker script generates a linker error indicating that the same section cannot be included in two different segments:

```
PHDRS { phdr1 PT_LOAD; phdr2 PT_LOAD;
}
.text : {
*(*.text*)
} : phdr1 :phdr2
```

The PHDRS command overrides the linker's default program header settings. For multiple program headers, only the first can have an LMA or VMA definition.

RELATED INFORMATION

"SECTIONS" on page 30

4.3.2 SECTIONS

```
SECTIONS {
  section_statement   section_statement
      ... }
```

The SECTIONS script command specifies how input sections are mapped to output sections, and where output sections are located in memory. The SECTIONS command must be specified once, and only once, in a linker script.

Section statements

A SECTIONS command contains one or more section statements, each of which can be one of the following:

- An ENTRY command
- A *symbol assignment statement* to set the location counter. The location counter specifies the default address in subsequent section-mapping statements that do not explicitly specify an address.
- An *output section description* to specify one or more input sections in one or more library files, and maps those sections to an output section. The virtual memory address of the output section can be specified using attribute keywords.

Output section description

A `SECTIONS` command can contain one or more *`output section descriptions`*.

```
section-name [virtual_addr][(type)] :
    [AT(load_addr)]
    [ALIGN(section_align)]
    [SUBALIGN(subsection_align)]
    [constraint]
{
        output-section-command         output-section-command
   ...
}[>region][AT>lma_region][ : phdr …][ =fillexp]
```

An output section description has the following syntax:

- `section-name` – Specifies the name of the output section
- `virtual-addr` – Specifies the virtual address of the output section (optional). The address value can be an expression.
- `type` – Specifies the section load property (optional)
- `NOLOAD` – Marks a section as not loadable
- `DSECT` – Parsed only, no effect on linking
- `COPY` – Parsed only, no effect on linking
- `INFO` – Parsed only, no effect on linking
- `OVERLAY` – Parsed only, no effect on linking
- `load-addr` – Specifies the load address of the output section (optional). The address value can be specified as an expression.
- `section-align` – Specifies the section alignment of the output section (optional). The alignment value can be an expression.
- `subsection-align` – Specifies the subsection alignment of the output section (optional). The alignment value can be an expression. *`constraint`* specifies the access type of the input sections (optional).
- `NOLOAD` – All input sections are read-only
- `DSECT` – All input sections are read/write (default)
- `output-section-command` – Specifies an output section command. An output section description contains one or more output section commands.
- `region` – Specifies the region of the output section (optional). The region is expressed as a string. This option is parsed, but has no effect on linking.
- `lma-region` – Specifies the load memory address (LMA) region of the output section (optional). The value can be an expression. This option is parsed, but has no effect on linking.

- `fillexp` – Specifies the fill value of the output section (optional). The value can be an expression. This option is parsed, but has no effect on linking.
- `phdr` – Specifies a program segment for the output section (optional). This option can appear more than once in an output section description to assign multiple program segments to an output section. Setting `phdr` in an output section description will affect subsequent output sections.

Output section commands

An output section description contains one or more *`output section commands`*. An output section command consists of one or more statements separated by a semicolon. The statements can be the following:

- Output section data
- Output section keyword
- A *`symbol assignment statement`* to set the location counter. The location counter specifies the default address in subsequent section-mapping statements that do not explicitly specify an address.
- An *`input section description`* to specify one or more input sections in one or more library files.

Output section data

The `OUTPUT_SECTION_DATA` operator can be used to include specific bytes of data from an expression value. It has the following syntax:

`OUTPUT_SECTION_DATA_`*`keyword`*`(`*`expression`*`)`
`... where` *`keyword`* `can have the literal values BYTE, SHORT, LONG, QUAD, or SQUAD.`

NOTE Output section data is not currently supported by the linker. The keyword is parsed but the data value generated is undefined and the linker does not generate a warning message. Replace all output section data references with fill expressions.

Output section keyword

For compatibility with the GNU linker `CREATE_OBJECT_SYMBOLS`, `CONSTRUCTORS`, and `SORT_BY_NAME(CONSTRUCTORS)` are parsed, but have no effect on linking.

Input section descriptions

An output section command may contain one or more input section descriptions. An input section description specifies the sections to be linked and the files they are contained in. An input section description has the following basic syntax for specifying a section in an object file:

```
file_name(section_name)
```

A single input section description can specify multiple files or sections:

```
file_name[:file_name]... (section_name[section_name[…)
```

Separate multiple file names with colons and multiple section names with spaces.

The wildcard characters, `*` and `?`, can be used in file names and section names. For example, the following input file description specifies all input sections named `.text*` contained in all linker input files:

```
*(.text*)
```

Use the `EXCLUDE_FILE` operator to reduce the number of items matched by a wildcard expression. Excluded items can be files, archives, or archive members. For example:

```
*(EXCLUDE_FILE(*crtend.o) .text .data)
```

This specifies all the linker input files except for any files named `*crtend.o`.

NOTE If an exclusion is used in a list of section names, it applies only to the immediately following section name in the list (`.text` in the example above).

Use the `KEEP` operator to prevent the linker performing garbage collection on unused sections when the linker option `-gc-sections` is used. For example:

```
KEEP(*(.init))
```

This says that the input section `.init` will be retained by the linker even if not referenced in the program being linked.

For compatibility with the GNU linker, the sort operators, `SORT_NONE`, `SORT_BY_NAME`, `SORT_BY_ALIGNMENT`, and `SORT_BY_INIT_PRIORITY`, are all parsed, but have no effect on linking.

Orphan section placement

Orphan sections are sections present in the input files but not explicitly placed into the output file by the linker script. The linker will still copy these sections into the output file but with no guess as to where they should be placed. The linker uses a simple heuristic and attempts to place orphan sections with the following steps:

1. The linker checks the properties of the section that it needs to autoplace. It looks for these properties in the `text`, `data`, and `bss` sections or any predefined section in the linker. If a match is found, the linker places the orphan section at the predefined location.
2. If the orphan section does not match any predefined sections, the linker will place the orphan section after nonorphan sections of the same attribute or permissions.

A special category of orphan sections have an attribute that the compiler tells the linker to place. The compiler communicates this by using an attribute that specifies an address. You can use `__attribute__((at(`*`XXX`*`))` to specify this. The linker uses this information to place the section at the address specified. If there is not enough room, it produces an error.

RELATED INFORMATION

“Example script file” on page 27
“ENTRY” on page 34
“Symbol assignment” on page 36
“Expressions” on page 35
“Linker script examples” on page 38

4.3.3 ENTRY

`ENTRY (`*`symbol`*`)`

The `ENTRY` script command specifies the program execution entry point. The entry point is the first instruction that is executed after a program is loaded.

This command is equivalent to the linker command-line option,`-e`.

4.3.4 OUTPUT_FORMAT

`OUTPUT_FORMAT (`*`string`*`)`

The `OUTPUT_FORMAT` script command specifies the output file properties. For compatibility with the GNU linker, this command is parsed but has no effect on linking.

4.3.5 OUTPUT_ARCH

`OUTPUT_ARCH ("aarch64")`

The `OUTPUT_ARCH` script command specifies the target processor architecture. For compatibility with the GNU linker, this command is parsed but has no effect on linking.

4.3.6 SEARCH_DIR

`SEARCH_DIR (`*`path`*`)`

The `SEARCH_DIR` script command specifies which adds the specified path to the list of paths that the linker uses to search for libraries.

This command is equivalent to the linker command-line option,`-L`.

4.3.7 INCLUDE

`INCLUDE (`*`file`*`)`

The `INCLUDE` script command specifies the contents of the text file at the current location in the linker script. The specified file is searched for in the current directory and any directory that the linker uses to search for libraries.

NOTE Include files can be nested.

4.3.8 OUTPUT

`OUTPUT (`*`file`*`)`

The `OUTPUT` script command defines the location and file where the linker will write output data. Only one output is allowed per linking.

4.3.9 GROUP

```
GROUP (file, file, …)
```

The `GROUP` script command includes a list of achieved file names. The achieved names defined in the list are searched repeatedly until all defined references are resolved.

4.3.10 ASSERT

```
ASSERT(expression, string)
```

The `ASSERT` script command adds an assertion to the linker script.

4.4 Expressions

Expressions in linker scripts are identical to C expressions. They are evaluated in 32-bit for ARM architecture and 64-bit for AArch64 architecture.

In addition to the `SECTION` command operators, the linker defines a number of functions, which can be used in linker script expressions.

Table 4-2 Linker script expression functions

Function	Description
`.`	Return the location counter value representing the current virtual address.
`ABSOLUTE (`*`expression`*`)`	Return the absolute value of the expression.
`ADDR (`*`string`*`)`	Return the virtual address of the symbol or section. Dot (.) is supported.
`ALIGN (`*`expression`*`)`	Return value when the current location counter is aligned to the next expression boundary. The value of the current location counter is not changed.
`ALIGN (`*`expression1`*`, `*`expression 2`*`)`	Return value when the value of expression1 is aligned to the next expression2 boundary.
`ALIGNOF (`*`string`*`)`	Return the align information of the symbol or section.
`ASSERT (`*`expression`*`, `*`string`*`)`	Throw an assertion if the expression result is zero.
`BLOCK (`*`expression`*`)`	Synonym for ALIGN (expression).
`DATA_SEGMENT_ALIGN(`*`maxpagesize`*`, `*`commonpagesize`*`)`	Equivalent to: `(ALIGN(maxpagesize)+(.&(maxpagesize - 1))) or (ALIGN(maxpagesize)+(.&(maxpagesize - commonpagesize)))` The linker computes both of these values and returns the larger one.
`DATA_SEGMENT_END (`*`expression`*`)`	Not used; return the value of the expression.
`DATA_SEGMENT_RELRO_END (`*`expression`*` )`	Not used; return the value of the expression.
`DEFINED (`*`symbol`*`)`	Return 1 if the symbol is defined in the global symbol table of the linker. Otherwise, return 0.

Table 4-2 Linker script expression functions (cont.)

Function	Description
`LOADADDR (string)`	Synonym for `ADDR (string)`.
`MAX (expression, expression)`	Return the maximum value of two expressions.
`MIN (expression1, expression2)`	Return the minimum value of two expressions.
`SEGMENT_START (string, expression)`	If a string matches a known segment, return the start address of that segment. If nothing is found, return the value of the expression.
`SIZEOF (string)`	Return the size of the symbol, section, or segment.
`SIZEOF_HEADERS`	Return the section start file offset.
`CONSTANT (MAXPAGESIZE)`	Return the defined default page size required by ABI.
`CONSTANT (COMMONPAGESIZE)`	Return the defined common page size.

RELATED INFORMATION

"SECTIONS" on page 30

4.5 Symbol assignment

Any symbol defined in a linker script becomes a global symbol. The following C assignment operators are supported to assign a value to a symbol:

- `symbol=expression;`
- `symbol+=expression;`
- `symbol-=expression;`
- `symbol*=expression;`
- `symbol/=expression;`
- `symbol&=expression;`
- `symbol|=expression;`
- `symbol<<=expression;`
- `symbol>>=expression;`

NOTE The first statement above defines `symbol` and assigns it the value of `expression`. In the other statements, `symbol` must already be defined.

All the statements above must be terminated with a semicolon character.

One way to create an empty space in memory is to use the expression `.+=space_size`:

```
BSS1 { . += 0x2000 }
```

This statement generates a section named `BSS1` with size 0x2000.

The symbol assignment functions supported by linker scripts are listed in the following table.

Table 4-3 Linker script symbol assignments

Function	Description
`HIDDEN (symbol = expression)`	Hide the defined symbol so it is not exported.
`FILL (expression)`	Specify the fill value for the current section. The fill length can be 1, 2, 4, or 8. The linker determines the length by selecting the minimum fit length. In the following example, the fill length is 8: `FILL( 0xdeadc0de )` A `FILL` statement covers memory locations from the point at which it occurs to the end of the current section. Multiple `FILL` statements can be used in an output section definition to fill different parts of the section with different patterns.
`ASSERT (expression, string)`	When the specified expression is zero, the linker throws an assertion with the specified message string.
`PROVIDE (symbol = expression)`	Similar to symbol assignment, but does not perform checking for an unresolved reference.
`PROVIDE_HIDDEN (symbol = expression)`	Similar to PROVIDE, but hides the defined symbol so it will not be exported.
`PRINT (symbol = expression)`	Instruct the linker to print symbol name and expression value to standard output during parsing.

4.6 NOCROSSREFS

The NOCROSSREFS command takes a list of space-separated output section names as its arguments. Any cross references among these output sections will result in link editor failure. The list can also contain an orphan section that is not specified in the linker script.

A linker script can contain multiple NOCROSSREFS commands. Each command is treated as an independent set of output sections that are checked for cross references.

4.7 Linker script examples

The following example linker scripts show how to specify input files for linking. The examples use the EXCLUDE_FILE operator, which is defined in the input section descriptions.

RELATED INFORMATION

"SECTIONS" on page 30

4.7.1 Exclude file in archive

To exclude a file in an archive from linking, specify the archive as part of the input expression, and specify the file to be excluded as the parameter of the following the EXCLUDE_FILE operator.

The following example uses four relocatable files: a1.o, a2.o, a3.o, and a4.o. Each file contains functions named foo_*N* and bar_*N*, where *N* indicates the digit in the file name. The linker script excludes foo_2 but not bar_2 from .text1 because EXCLUDE_FILE applies only to the immediately following section name.

```
script.t : SECTIONS {
  .text1 : {
    *lib23.a:(EXCLUDE_FILE(a2.o) .text.foo* .text.bar*)
  }
  .text2 : {
  *(*)
  } }
clang -ffunction-sections -c a1.c a2.c a3.c a4.c
arm-ar cr lib23.a a2.o a3.o
arm-ar cr lib4.a a4.o
arm-link -T script.t -o mcld.out a1.o --whole-archive lib23.a lib4.a --no-
whole-archive
```

Section headers starting at offset 0x22a0 are printed as follows:

```
  [Nr] Name      Type         Addr      Off     Size    ES  Flg  Lk  Inf  Al
  [ 0]           NULL         00000000  000000  000000  00        0    0   0
  [ 1] .text1    PROGBITS     00000000  001000  00002c  00   AX   0    0  16
  [ 2] .text2    PROGBITS     00000030  002030  00009c  00  WAX   0    0  16
```

The symbol table is printed as follows:

```
   Num:    Value   Size  Type    Bind   Vis       Ndx  Name
     8:  00000040    12  FUNC    GLOBAL DEFAULT    2  bar_1
     9:  00000000    12  FUNC    GLOBAL DEFAULT    1  bar_2
    10:  00000020    12  FUNC    GLOBAL DEFAULT    1  bar_3
    11:  000000c0    12  FUNC    GLOBAL DEFAULT    2  bar_4
    12:  00000030    12  FUNC    GLOBAL DEFAULT    2  foo_1
    13:  000000a0    12  FUNC    GLOBAL DEFAULT    2  foo_2
    14:  00000010    12  FUNC    GLOBAL DEFAULT    1  foo_3
    15:  000000b0    12  FUNC    GLOBAL DEFAULT    2  foo_4
```

4.7.2 Exclude all files in archive

To exclude all files in an archive from linking, specify the archive to be excluded as the parameter of the `EXCLUDE_FILE` operator.

The following example uses four relocatable files: `a1.o`, `a2.o`, `a3.o`, and `a4.o`. Each file contains functions named `foo_`*N* and `bar_`*N*, where *N* indicates the digit in the file name. The linker script excludes `foo_2/3` but not `bar_2/3` from `.text1` because `EXCLUDE_FILE` applies only to the immediately following section name.

```
script.t : SECTIONS {
  .text1 : {
    *lib*:(EXCLUDE_FILE(*lib23.a) .text.foo* .text.bar*)   }
  .text2 : {
  *(*)
  } }
clang -ffunction-sections -c a1.c a2.c a3.c a4.c
arm-ar cr lib23.a a2.o a3.o
arm-ar cr lib4.a a4.o
arm-link -T script.t -o mcld.out a1.o --whole-archive lib23.a lib4.a --no-
whole-archive
```

Section headers starting at offset 0x22a0 are printed as follows:

```
  [Nr] Name       Type          Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]            NULL          00000000 000000 000000 00      0   0  0
  [ 1] .text1     PROGBITS      00000000 001000 00003c 00  AX  0   0 16
  [ 2] .text2     PROGBITS      00000040 002040 00008c 00 WAX  0   0 16
```

The symbol table is printed as follows:

```
Num:    Value  Size Type    Bind   Vis       Ndx Name
  8: 00000050    12 FUNC    GLOBAL DEFAULT    2 bar_1
  9: 00000000    12 FUNC    GLOBAL DEFAULT    1 bar_2
 10: 00000010    12 FUNC    GLOBAL DEFAULT    1 bar_3
 11: 00000030    12 FUNC    GLOBAL DEFAULT    1 bar_4
 12: 00000040    12 FUNC    GLOBAL DEFAULT    2 foo_1
 13: 000000b0    12 FUNC    GLOBAL DEFAULT    2 foo_2
 14: 000000c0    12 FUNC    GLOBAL DEFAULT    2 foo_3
 15: 00000020    12 FUNC    GLOBAL DEFAULT    1 foo_4
```

4.7.3 Exclude multiple files

To exclude multiple files from linking, specify the files as parameters of the `EXCLUDE_FILE` operator. `EXCLUDE_FILE`accepts multiple file name parameters.

The following example uses four relocatable files: `a1.o`, `a2.o`, `a3.o`, and `a4.o`. Each file contains functions named `foo_`*`N`* and `bar_`*`N`*, where *`N`* indicates the digit in the file name. The linker script excludes `foo_2/3` but not `bar_2/3` from `.text1` because `EXCLUDE_FILE` applies only to the immediately following section name.

```
script.t : SECTIONS {
  .text1 : {
    *lib*:(EXCLUDE_FILE(a2.0 a3.0) .text.foo* .text.bar*)
  }
  .text2 : {
  *(*)
  } }
clang -ffunction-sections -c a1.c a2.c a3.c a4.c
arm-ar cr lib23.a a2.o a3.o
arm-ar cr lib4.a a4.o
arm-link -T script.t -o mcld.out a1.o --whole-archive lib23.a lib4.a --no-
whole-archive
```

Section headers starting at offset 0x2260 are printed as follows:

```
  [Nr] Name       Type          Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]            NULL          00000000 000000 000000 00      0   0  0
  [ 1] .text1     PROGBITS      00000000 001000 00003c 00  AX  0   0 16
  [ 2] .text2     PROGBITS      00000040 002040 000040 00 WAX  0   0 16
```

The symbol table is printed as follows:

```
Num:    Value  Size Type    Bind   Vis       Ndx Name
  8: 00000050    12 FUNC    GLOBAL DEFAULT    2 bar_1
  9: 00000000    12 FUNC    GLOBAL DEFAULT    1 bar_2
 10: 00000010    12 FUNC    GLOBAL DEFAULT    1 bar_3
 11: 00000030    12 FUNC    GLOBAL DEFAULT    1 bar_4
 12: 00000040    12 FUNC    GLOBAL DEFAULT    2 foo_1
 13: 00000060    12 FUNC    GLOBAL DEFAULT    2 foo_2
 14: 00000070    12 FUNC    GLOBAL DEFAULT    2 foo_3
 15: 00000020    12 FUNC    GLOBAL DEFAULT    1 foo_4
```

4.7.4 Exclude archive and non-archive files

To exclude archive and non-archive files from linking, specify the files as parameters of the `EXCLUDE_FILE` operator. `EXCLUDE_FILE` searches inside and outside archives for files to exclude.

The following example uses four relocatable files: `a1.o`, `a2.o`, `a3.o`, and `a4.o`. Each file contains functions named `foo_N` and `bar_N`, where *N* indicates the digit in the file name. The linker script excludes `foo_1/2` but not `bar_1/2` from `.text1` because `EXCLUDE_FILE` applies only to the immediately following section name.

```
script.t : SECTIONS {
  .text1 : {
    *:(EXCLUDE_FILE(a[12].o) .text.foo* .text.bar*)
  }
  .text2 : {
  *(*)
  } }
clang -ffunction-sections -c a1.c a2.c a3.c a4.c
arm-ar cr lib23.a a2.o a3.o
arm-ar cr lib4.a a4.o
arm-link -T script.t -o mcld.out a1.o --whole-archive lib23.a lib4.a --no-
whole-archive
```

Section headers starting at offset 0x2260 are printed as follows:

```
  [Nr] Name       Type          Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]            NULL          00000000 000000 000000 00      0   0  0
  [ 1] .text1     PROGBITS      00000000 001000 00004c 00  AX  0   0 16
  [ 2] .text2     PROGBITS      00000050 002050 000030 00 WAX  0   0 16
```

The symbol table is printed as follows:

```
Num:    Value  Size Type    Bind   Vis       Ndx Name
  8: 00000060    12 FUNC    GLOBAL DEFAULT    2 bar_1
  9: 00000000    12 FUNC    GLOBAL DEFAULT    1 bar_2
 10: 00000020    12 FUNC    GLOBAL DEFAULT    1 bar_3
 11: 00000040    12 FUNC    GLOBAL DEFAULT    1 bar_4
 12: 00000050    12 FUNC    GLOBAL DEFAULT    2 foo_1
 13: 00000070    12 FUNC    GLOBAL DEFAULT    2 foo_2
 14: 00000010    12 FUNC    GLOBAL DEFAULT    1 foo_3
 15: 00000030    12 FUNC    GLOBAL DEFAULT    1 foo_4
```

4.7.5 Conflicting wildcards

To exclude archive and nonarchive files from linking, specify the files as parameters of the EXCLUDE_FILE operator. EXCLUDE_FILE earches inside and outside archives for files to exclude.

The following example uses four relocatable files: a1.o, a2.o, a3.o, and a4.o. Each file contains functions named foo_*N* and bar_*N*, where *N* indicates the digit in the file name. The linker script excludes foo_1/2 but not bar_1/2 from .text1 because EXCLUDE_FILE applies only to the immediately following section name.

```
script.t : SECTIONS {
  .text1 : {
    *:(EXCLUDE_FILE(a[12].o) .text.foo* .text.bar*)
  }
  .text2 : {
  *(*)
  } }
clang -ffunction-vb419sections -c a1.c a2.c a3.c a4.c
arm-ar cr lib23.a a2.o a3.o
arm-ar cr lib4.a a4.o
arm-link -T script.t -o mcld.out a1.o --whole-archive lib23.a lib4.a --no-
whole-archive
```

Section headers starting at offset 0x2260 are printed as follows:

```
  [Nr] Name       Type          Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]            NULL          00000000 000000 000000 00      0   0  0
  [ 1] .text1     PROGBITS      00000000 001000 00004c 00  AX  0   0 16
  [ 2] .text2     PROGBITS      00000050 002050 000030 00 WAX  0   0 16
```

The symbol table is printed as follows:

```
Num:    Value  Size Type    Bind   Vis       Ndx Name
  8: 00000060    12 FUNC    GLOBAL DEFAULT    2 bar_1
  9: 00000000    12 FUNC    GLOBAL DEFAULT    1 bar_2
 10: 00000020    12 FUNC    GLOBAL DEFAULT    1 bar_3
 11: 00000040    12 FUNC    GLOBAL DEFAULT    1 bar_4
 12: 00000050    12 FUNC    GLOBAL DEFAULT    2 foo_1
 13: 00000070    12 FUNC    GLOBAL DEFAULT    2 foo_2
 14: 00000010    12 FUNC    GLOBAL DEFAULT    1 foo_3
 15: 00000030    12 FUNC    GLOBAL DEFAULT    1 foo_4
```

4.7.6 GNU linker option supporting list

The following table lists all GNU linker options. The designations in the QCLD column are as follows:

- Y – Supported by the linker
- N – Not supported by the linker
- D – Supported by the linker but with different usage

Table 4-4 GNU linker options

GNU option	QCLD	Comment
`-a KEYWORD`	N	Shared library control for HP/UX compatibility.
`-A ARCH`	N	Set the architecture.
`--architecture ARCH`	N	Set the architecture.
`-b TARGET`	N	Specify the target for the input files that follow.
`--format TARGET`	N	Specify the target for the input files that follow.
`-c FILE`	N	Read the MRI format linker script.
`--mri-script FILE`	N	Read the MRI format linker script.
`-d`	Y	Force common symbols to be defined.
`-dc`	Y	Force common symbols to be defined.
`-dp`	Y	Force common symbols to be defined.
`-e ADDRESS`	Y	Set the start address.
`--entry ADDRESS`	Y	Set the start address.
`-E`	Y	Export all dynamic symbols.
`--export-dynamic`	Y	Export all dynamic symbols.
`--no-export-dynamic`	Y	Undo the effect of `--export-dynamic`.
`-EB`	N	Link the big-endian objects.
`-EL`	N	Link the little-endian objects.
`-f SHLIB`	N	Filter for the shared object symbol table.
`--auxiliary SHLIB Auxiliary`	N	Filter for the shared object symbol table.
`-F SHLIB`	N	Filter for the shared object symbol table.
`--filter SHLIB`	N	Filter for the shared object symbol table.
`-g`	Y	Ignored.
`-G SIZE`	Y	Small data size. If no size is specified, the size is the same as `--shared`.
`--gpsize SIZE`	Y	Small data size. If no size is specified, the size is the same as `--shared`.
`-h FILENAME`	Y	Set the internal name of shared library.
`-soname FILENAME`	Y	Set the internal name of shared library.
`-I PROGRAM`	N	Set `PROGRAM` as the dynamic linker to use.
`--dynamic-linker PROGRAM`	Y	Set `PROGRAM` as the dynamic linker to use.

Table 4-4 GNU linker options (cont.)

GNU option	QCLD	Comment
-l LIBNAME	Y	Search for library, LIBNAME.
--library LIBNAME	Y	Search for library, LIBNAME.
-L DIRECTORY	Y	Add DIRECTORY to the library search path.
--library-path DIRECTORY	Y	Add DIRECTORY to the library search path.
--sysroot=<DIRECTORY>	Y	Override the default sysroot location.
-m EMULATION	Y	Set the emulation.
-M	Y	Print the map file on standard output.
--print-map	Y	Print the map file on standard output.
-n	Y	Do not page align the data.
--nmagic	Y	Do not page align the data.
-N	Y	Do not page align the data. Do not make the text read only.
--omagic	Y	Do not page align the data. Do not make the text read only.
--no-omagic	Y	Page the data. Do not make the text read only.
-o FILE	Y	Set the output file name.
--output FILE	Y	Set the output file name.
-O	N	Optimize the output file.
-plugin PLUGIN	N	Load the named plugin.
-plugin-opt ARG	N	Send ARG to the last-loaded plugin.
-flto	Y	Ignored for GCC LTO option compatibility.
-flto-partition=	N	Ignored for GCC LTO option compatibility.
-fuse-ld=	N	Ignored for GCC linker option compatibility.
-Qy	Y	Ignored for SVR4 compatibility.
-q	N	Generate relocations in the final output.
--emit-relocs	Y	Generate relocations in the final output.
-r	Y	Generate the relocatable output.
-i	N	Generate the relocatable output.
--relocatable	N	Generate the relocatable output.
-R FILE	Y	Just link symbols. If a directory, same as --rpath.
--just-symbols FILE	Y	Just link symbols. If a directory, same as --rpath.
-s	Y	Strip all symbols.
--strip-all	Y	Strip all symbols.
-S	Y	Strip all symbols.
--strip-debug	Y	Strip debugging symbols.
--strip-discarded	N	Strip symbols in discarded sections.
--no-strip-discarded	N	Do not strip symbols in discarded sections.

Table 4-4 GNU linker options (cont.)

GNU option	QCLD	Comment
-t	D	Trace file opens.
--trace	D	Trace file opens.
-T FILE	Y	Read the linker script.
--script FILE	Y	Read the linker script.
--default-script FILE	Y	Read the default linker script.
-dT	N	Read the default linker script.
-u SYMBOL	Y	Start with an undefined reference to SYMBOL.
--undefined SYMBOL	Y	Start with an undefined reference to SYMBOL.
--unique [=SECTION]	N	Do not merge input [SECTION | orphan] sections.
-Ur	N	Build global constructor/destructor tables.
-v	N	Print the version information.
--version	Y	Print the version information.
-V	N	Print the version and emulation information.
-x	Y	Discard all local symbols.
--discard-all	Y	Discard all local symbols.
-X	Y	Discard temporary local symbols (default).
--discard-locals	Y	Discard temporary local symbols (default).
--discard-none	N	Do not discard any local symbols.
-y SYMBOL	N	Trace mentions of SYMBOL.
--trace-symbol SYMBOL	N	Trace mentions of SYMBOL.
-Y PATH	Y	Default search path for Solaris compatibility.
-(	N	Start a group.
--start-group	Y	Start a group.
-)	N	End a group.
--end-group	Y	End a group.
--accept-unknown-input-arch	N	Accept input files whose architecture cannot be determined.
--no-accept-unknown-input-arch	N	Reject input files whose architecture is unknown.
--as-needed	Y	Only set DT_NEEDED for following dynamic libs if used.
--no-as-needed	Y	Always set DT_NEEDED for dynamic libraries mentioned on the command line.
-assert KEYWORD	N	Ignored for SunOS compatibility.
-Bdynamic	Y	Link against shared libraries.
-dy	Y	Link against shared libraries.
-call_shared	Y	Link against shared libraries.
-Bstatic	Y	Do not link against shared libraries.
-dn	Y	Do not link against shared libraries.

Table 4-4 GNU linker options (cont.)

GNU option	QCLD	Comment
-non_shared	Y	Do not link against shared libraries.
-static	Y	Do not link against shared libraries.
-Bsymbolic	Y	Bind global references locally.
-Bsymbolic-functions	Y	Bind global function references locally.
--check-sections	N	Check section addresses for overlaps (default).
--no-check-sections	N	Do not check section addresses for overlaps.
--copy-dt-needed-entries	Y	Copy DT_NEEDED links mentioned inside DSOs that follow.
--no-copy-dt-needed-entries	Y	Do not copy DT_NEEDED links mentioned inside DSOs that follow.
--cref	Y	Output a cross-reference table.
--defsym SYMBOL=EXPRESSION	Y	Define a symbol.
--demangle [=STYLE]	N	Demangle symbol names using STYLE.
--embedded-relocs	N	Generate embedded relocations.
--fatal-warnings	Y	Treat warnings as errors.
--no-fatal-warnings	Y	Do not treat warnings as errors (default).
-fini SYMBOL	Y	Call SYMBOL at unload time.
--force-exe-suffix	N	Force generation of a file with the .exe suffix.
--gc-sections	Y	Remove unused sections (on some targets).
--no-gc-sections	Y	Do not remove unused sections (default).
--print-gc-sections	Y	List removed unused sections on stderr.
--no-print-gc-sections	N	Do not list removed unused sections.
--hash-size=<NUMBER>	Y	Set the default hash table size close to <NUMBER>.
--help	Y	Print option help.
-init SYMBOL	Y	Call SYMBOL at load time.
-Map FILE	Y	Write a map file.
--no-define-common	N	Do not define common storage.
--no-demangle	Y	Do not demangle symbol names.
--no-keep-memory	N	Use less memory and more disk I/O.
--no-undefined	Y	Do not allow unresolved references in object files.
--allow-shlib-undefined	Y	Allow unresolved references in shared libraries.
--no-allow-shlib-undefined	Y	Do not allow unresolved references in shared libraries.
--allow-multiple-definition	Y	Allow multiple definitions.
--no-undefined-version	N	Disallow undefined version.
--default-symver	N	Create default symbol version.
--default-imported-symver	N	Create default symbol version for imported symbols.
--no-warn-mismatch	Y	Do not warn about mismatched input files.

Table 4-4 GNU linker options (cont.)

GNU option	QCLD	Comment
--no-warn-search-mismatch	N	Do not warn on finding an incompatible library.
--no-whole-archive	Y	Turn off --whole-archive.
--noinhibit-exec	Y	Create an output file even if errors occur.
-nostdlib	Y	Only use the library directories specified on the command line.
--oformat TARGET	N	Specify the target of output file.
--print-output-format	N	Print the default output format.
--print-sysroot	N	Print the current sysroot.
-qmagic	N	Ignored for Linux compatibility.
--reduce-memory-overheads	N	Reduce memory overheads, possibly taking much longer.
--relax	N	Reduce code size by using target-specific optimizations.
--no-relax	N	Do not use relaxation techniques to reduce code size.
--retain-symbols-file FILE	N	Keep only symbols listed in FILE.
-rpath PATH	N	Set the runtime shared library search path.
-rpath-link PATH	N	Set the link time shared library search path.
-shared	Y	Create a shared library.
-Bshareable	Y	Create a shared library.
-pie	Y	Create a position independent executable.
--pic-executable	Y	Create a position independent executable.
--sort-common	N	Sort common symbols by alignment in a specified order.
--sort-section	N	Sort sections by name or maximum alignment.
--spare-dynamic-tags COUNT	N	Indicate how many tags to reserve in the .dynamic section.
--split-by-file [=SIZE]	N	Split output sections every SIZE octets.
--split-by-reloc [=COUNT]	N	Split output sections every COUNT relocations.
--stats	N	Print memory usage statistics.
--target-help	N	Display target-specific options.
--task-link SYMBOL	N	Do task-level linking.
--traditional-format	N	Use the same format as the native linker.
--section-start SECTION=ADDRESS	Y	Set the address of the named section.
-Tbss ADDRESS	N	Set the address of the .bss section.
-Tdata ADDRESS	N	Set the address of the .data section.
-Ttext ADDRESS	N	Set the address of the .text section.
-Ttext-segment ADDRESS	N	Set the address of the text segment.
-Trodata-segment ADDRESS	N	Set the address of the rodata segment.

Table 4-4 GNU linker options (cont.)

GNU option	QCLD	Comment
-Tldata-segment ADDRESS	N	Set the address of the ldata segment.
--unresolved-symbols=<method>	N	Indicate how to handle unresolved symbols.
--verbose [=NUMBER]	Y	Output lots of information during link.
--version-script FILE	N	Read the version information script.
--version-exports-section SYMBOL	N	Take the export symbols list from .exports, using SYMBOL as the version.
--dynamic-list-data	N	Add data symbols to dynamic list
--dynamic-list-cpp-new	N	Use C++ operator new/delete dynamic list
--dynamic-list-cpp-typeinfo	N	Use C++ typeinfo dynamic list.
--dynamic-list FILE	Y	Read the dynamic list.
--warn-common	Y	Warn about duplicate common symbols.
--warn-constructors	N	Warn if global constructors/destructors are seen.
--warn-multiple-gp	N	Warn if multiple GP values are used.
--warn-once	Y	Warn only once per undefined symbol.
--warn-section-align	N	Warn if the start of section changes due to alignment.
--warn-shared-textrel	Y	Warn if a shared object has DT_TEXTREL.
--warn-alternate-em	N	Warn if an object has alternate ELF machine code.
--warn-unresolved-symbols	N	Report unresolved symbols as warnings.
--error-unresolved-symbols	N	Report unresolved symbols as errors.
--whole-archive	Y	Include all objects from the archives that follow.
--wrap SYMBOL	Y	Use wrapper functions for SYMBOL.
--ignore-unresolved-symbol SYMBOL	N	Unresolved SYMBOL will not cause an error or warning.
--push-state	N	Push the state of flags governing input file handling.
--pop-state	N	Pop the state of flags governing input file handling.
@FILE	Y	Read options from FILE.
--audit=AUDITLIB	N	Specify a library to use for auditing.
-Bgroup	Y	Select the group name lookup rules for DSO.
--build-id[=STYLE]	Y	Generate a build ID note.
-P AUDITLIB	Y	Specify a library to use for auditing dependencies.
--depaudit=AUDITLIB	N	Specify a library to use for auditing dependencies.
--disable-new-dtags	Y	Disable new dynamic tags.
--enable-new-dtags	Y	Enable new dynamic tags.
--eh-frame-hdr	Y	Create a .eh_frame_hdr section.
--exclude-libs=LIBS	Y	Make all symbols in LIBS hidden.
--hash-style=STYLE	Y	Set the hash style to sysv, gnu, or both.
-z combreloc	Y	Merge dynamic relocations into one section and sort.

Table 4-4 GNU linker options (cont.)

GNU option	QCLD	Comment
-z common-page-size=SIZE	Y	Set the common page size to SIZE.
-z defs	Y	Report unresolved symbols in object files.
-z execstack	Y	Mark an executable as requiring an executable stack.
-z global	Y	Make symbols in DSO as available for subsequently loaded objects.
-z initfirst	Y	Mark DSO to be initialized first at runtime.
-z interpose	N	Mark an object to interpose all DSOs but executable.
-z lazy	Y	Mark an object as lazy runtime binding (default).
-z loadfltr	N	Mark an object as requiring immediate processing.
-z max-page-size=SIZE	Y	Set the maximum page size to SIZE.
-z muldefs	Y	Allow multiple definitions.
-z nocombreloc	Y	Do not merge dynamic relocations into one section.
-z nocopyreloc	Y	Do not create copy relocations.
-z nodefaultlib	N	Mark an object to not use default search paths.
-z nodelete	Y	Mark DSO as non-deletable at runtime.
-z nodlopen	N	Mark DSO as not available to dlopen.
-z nodump	N	Mark DSO not available to dldump.
-z noexecstack	Y	Mark an executable as not requiring an executable stack.
-z norelro	Y	Do not create the RELRO program header.
-z now	Y	Mark an object as non-lazy runtime binding.
-z origin	Y	Mark an object as requiring immediate $ORIGIN processing at runtime.
-z relro	Y	Create a RELRO program header.
-z stacksize=SIZE	N	Set the size of stack segment.
--thumb-entry=<sym>	N	Set the entry point to be a Thumb symbol <sym>.
--be8	N	Output the BE8 format image.
--target1-rel	N	Interpre R_ARM_TARGET1 as R_ARM_REL32.
--target1-abs	N	Interpret R_ARM_TARGET1 as R_ARM_ABS32.
--target2=<type>	N	Specify the definition of R_ARM_TARGET2.
--fix-v4bx	N	Rewrite BX rn as MOV pc, rn for the ARMv4 processor.
--fix-v4bx-interworking	N	Rewrite BX rn branch to the ARMv4 interworking veneer.
--use-blx	N	Enable the use of BLX instructions.
--vfp11-denorm-fix	N	Specify how to fix VFP11 denorm erratum.
--no-enum-size-warning	N	Do not warn about objects with incompatible enum sizes.

Table 4-4 GNU linker options (cont.)

GNU option	QCLD	Comment
`--no-wchar-size-warning`	N	Do not warn about objects with incompatible `wchar_t sizes`.
`--pic-veneer`	N	Always generate PIC interworking veneers.
`--long-plt`	N	Generate long `.plt` entries to handle large `.plt/.got` displacements.
`--stub-group-size=N`	N	Maximum size of a group of input sections that can be handled by one stub section.
`--[no-]fix-cortex-a8`	Y	Disable or enable a Cortex-A8 Thumb-2 branch erratum fix.
`--no-merge-exidx-entries`	N	Disable merging exidx entries.
`--[no-]fix-arm1176`	N	Disable or enable ARM1176 BLX immediate erratum fix `armelfb_linux_eabi`.
`--fix-cortex-a53-835769`	Y	Fix erratum `835769`.

A References

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm® Snapdragon™ LLVM ARM Compiler User Guide	80-VB419-99

Glossary

Term	Definition
LTO	Link time optimization