APPENDIX A

Using GCC Compilers

This appendix provides a general discussion of how to use GCC compilers. Originally this information was in the front of the book, but I believe that most readers just want to get started learning about and using specific compilers—not necessarily wading through a generic chapter on using GCC compilers in general. Thus, I have moved this to an appendix that provides a general reference. I have extracted quick "refresher courses" on common options from this appendix and put them in the chapters that discuss each specific compiler. I hope that works for you—if not, my apologies, and please let me know.

The goals of this appendix are to introduce you to how to specify options to a GCC compiler, to introduce shared options that you can use regardless of the specific compiler in the GCC family that you are using, and to explain how to modify the behavior of the GCC compilers using environment variables and spec strings. One of the big advantages of using a family of compilers that are produced from a single base of source code is that they share a large number of options, command-line syntax, and capabilities, regardless of whether you are compiling C, Objective C, C++, Fortran, or Java code. This appendix highlights the GCC command-line options that you can use anywhere, regardless of the language that you are working in. In some cases, an option is shared by most but not all of the GCC compilers—I will still treat these as a shared option and highlight the exception(s).

When you invoke any GCC compiler to compile a source code file, the compilation process passes through up to four stages: preprocessing, compilation, assembly, and linking. The first occurs for any GCC compiler, with the exception of GCC's Java compiler. The next two occur for any input source file in any language, and the fourth is an optional stage that combines code produced by the first three stages into a single, executable file. This appendix explains how to stop the compilation process at any of these stages or specify options that control the behavior of each of these compilation stages. Other options discussed in this appendix enable you to control the names and types of output files produced when compiling your applications and also enable you to exercise greater control over the content and format of any GCC compiler's diagnostic messages.

This appendix also provides a section discussing how to modify the behavior of GCC compilers by setting environment variables or modifying entries in the specification files that tell the GCC compilers what types of files to look for during the compilation process and what to do with them. It concludes with a complete listing of all of the options to GCC compilers that are generic to all compilers, for your reference and reading pleasure.

Using Options with GCC Compilers

All GCC compilers accept both single-letter options, such as `-o`, and multiletter options, such as `-ansi`. The consequence of GCC accepting both types of options is that, unlike many GNU programs, you cannot group multiple single-letter options. For example, the multiletter option `-pg` is not the same as the two single-letter options `-p -g`. The `-pg` option creates extra code in the final binary that outputs profile information for the GNU code profiler, gprof. The combination of the `-p` and –g options,

on the other hand, generates extra code in the resulting binary that outputs profiling information for use by the prof code profiler (`-p`) and causes GCC compilers to generate debugging information using the operating system's normal format (`-g`).

Despite their sensitivity to the grouping of multiple single-letter options, GCC compilers generally enable you to mix the order of options and arguments. For example, invoking GCC's C compiler as

```
gcc -pg -fno-strength-reduce -g myprog.c -o myprog
```

has the same result as

```
gcc myprog.c -o myprog -g -fno-strength-reduce -pg
```

I say that compilers generally enable you to mix options and arguments because, in most cases, the order of options and their arguments does not matter. In some situations order does matter if you use several options of the same kind. For example, the GCC C compiler's `-I` option specifies an extra directory to search for include files. So if you specify `-I` several times, GCC searches the listed directories in the order specified.

Many options have long names starting with `-f` or with `-W`. Examples include `-fforce-mem`, `-fstrength-reduce`, `-Wformat`, and so on. Similarly, most of these long name options have both positive and negative forms. Thus, the negative form of `-fstrength-reduce` would be `-fno-strength-reduce`.

■ **Note** The GCC manual documents only the nondefault version of long name options that have both positive and negative forms. That is, if the GCC manual documents `-mfoo`, the default is `-mno-foo`.

General Information Options

Various GCC command-line options can be used to display basic or extended usage tips and compiler configuration information or control overall behavior. Basically, the options I discuss in this section do not fit neatly into any other category, so I am placing them here.

Table A-1 lists and briefly describes the options that fall into this miscellaneous category.

Table A-1. *General GCC Options*

Option	Description
`-###`	Displays the programs and arguments that would be invoked as the compiler executes with the specified command-line, but does not actually execute them. This is my favorite initial debugging option, especially in cross-platform compilation.
`-dumpmachine`	Displays the compiler's target CPU.
`-dumpspecs`	Displays GCC's default spec strings (see the section "Customizing GCC Compilers with Spec Files and Spec Strings" later in this appendix).
`-dumpversion`	Displays the compiler version number.
`--help`	Displays basic usage information.
`-pass-exit-codes`	Causes GCC to return the highest error code generated by any failed compilation phase.

Table A-1. *General GCC Options*

Option	Description
`-pipe`	Uses pipes to send information between compiler processes rather than intermediate files.
`-print-file-name=`*`lib`*	Displays the path to the library named lib, where lib is a library that is part of the GCC installation.
`-print-libgcc-file-name`	Displays the name of the compiler's companion library.
`-print-multi-directory`	Displays the root directory for all versions of libgcc.
`-print-multi-lib`	Displays the maps between command-line options and multiple library search directories.
`-print-prog-name=`*`prog`*	Displays the path to the program named prog, where prog is an application that is part of the GCC installation.
`-print-search-dirs`	Displays the directory search path.
`-save-temps`	Saves intermediate files created during compilation.
`--target-help`	Displays help for command-line options specific to the compiler's target.
`-time`	Displays the execution time of each compilation subprocess.
`-v`	Displays the programs and arguments invoked as the compiler executes.
`-V` *`ver`*	Invokes version number *ver* of the compiler.
`--version`	Displays the compiler version information and short license.

Spec strings are macrolike constructs that GCC uses to define the paths and default options and arguments for the various components it calls during compilation. I will discuss spec strings in greater detail later in this appendix in the section titled "Customizing GCC Compilers with Spec Files and Spec Strings." The `-dumpversion` and `-dumpmachine` options show the compiler's version number and the processor for which it outputs code, respectively. Only one of these can be used at a time—when `-dumpversion` and `-dumpmachine` are on the same line, only the first argument gets processed.

```
$ gcc -dumpmachine
```

```
x86_64-unknown-linux-gnu
```

```
$ gcc -dumpversion
```

```
4.2.0
```

```
$ gcc --version
```

```
gcc (GCC) 4.2.0 20060508 (experimental)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There
is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

The `--version` argument displays more verbose version information. As you can see from this example, at the time that I captured this output, I was running an experimental version of gcc 4.2.0 on a 64-bit machine. Your output will certainly differ, though the type of information that these options provide will still be the same.

I find that the `-dumpversion` and `-dumpmachine` options are best used in scripts because their output is terse and easily parsed using standard shell utilities. The output from the `-dumpmachine` option may vary between two versions of GCC running on the same system. For example, the default compiler on 64-bit SUSE Linux version 10.0 is GCC 4.0.2. When passed the `–dumpmachine` option, it emits `x86_64-suse-linux`. GCC version 4.1.0, compiled and installed as described in Chapter 11 of this book, emits `x86_64-unknown-linux-gnu` when invoked with the `-dumpmachine` option. The various options listed in Table A-1 that begin with `-print-` enable you to determine the paths to the libraries and programs that GCC compilers use at runtime. This information can be useful when you are trying to track down a problem or want to be sure that a particular library is being used or a specific directory is being searched during the compilation process. The `-###` option can also be extremely useful in identifying path-related problems in libraries or executables.

If a GCC compiler encounters an error during any compilation phase, it exits and returns an error code of 1 to the calling process. Using the `-pass-exit-codes` option instructs GCC compilers to return the highest error code generated by any compilation phase, rather than simply 1. If you want to know how long the compiler runs in each phase of the compilation process, specify the `-time` option. The `-time` option can be especially instructive when compiling GCC compilers. In fact, this option is used as a rough gauge of the overall performance of a GCC compiler by the GCC developers: the faster that GCC can compile itself, the better the developers like it. You might also find it interesting to use `-time` when compiling the Linux kernel or any other large program that really stresses a system.

The `-pipe` option uses pipes rather than temporary files to exchange data between compilation phases. Though it does save some intermediate disk space, the purpose of the `–pipe` option is speed, not disk space conservation. As interprocess communication (IPC) mechanisms, pipes are faster than files because pipes avoid the overhead of file I/O.

Controlling GCC Compiler Output

As mentioned earlier, compiling source code potentially passes through as many as four stages: preprocessing, compilation itself, assembly, and linking. The first three stages, preprocessing, compilation, and assembly, occur at the level of individual source files for all GCC compilers with the exception of gcj, which does not use a preprocessor. The end result of the first three stages is an object file. Linking combines object files into the final executable. GCC compilers evaluate filename suffixes to select the type of compilation they will perform. Table A-2 maps filename suffixes to the type of compilation that the GCC compilers perform.

Table A-2. *GCC Operations by Filename Suffix*

Suffix	Operation
.ada	ADA source code to preprocess
.adb	ADA source code to preprocess
.c	C source code to preprocess
.C	C++ source code to preprocess
.c++	C++ source code to preprocess
.cc	C++ source code to preprocess
.class	Java bytecode, no preprocessing
.cp	C++ source code to preprocess
.cpp	C++ source code to preprocess
.CPP	C++ source code to preprocess
.cxx	C++ source code to preprocess
.f	FORTRAN 77 code to preprocess
.F	FORTRAN 77 code to preprocess
.f90	Fortran 90 code to preprocess
.f95	Fortran 95 code to preprocess
.for	FORTRAN 77 code to preprocess
.FOR	FORTRAN 77 code to preprocess
.fpp	FORTRAN 77 code to preprocess
.FPP	FORTRAN 77 code to preprocess
.i	C source code that should not be preprocessed
.ii	C++ source code that should not be preprocessed
.jar	Jar format archive file of Java source code, no preprocessing
.java	Java source code, no preprocessing
.m	Objective C source code to preprocess
.M	Objective C++ source code to preprocess
.mi	Object C source code that should not be preprocessed
.mm	Objective C++ source code to preprocess
.mii	Objective C++ source code that should not be preprocessed
.h	C header file—included but not compiled or linked
.p	Pascal source code to preprocess
.pas	Pascal source code to preprocess
.r	Rational Fortran (Ratfor) source code to preprocess

Table A-2. *GCC Operations by Filename Suffix (Continued)*

Suffix	Operation
`.s`	Assembly code
`.S`	Assembly code to preprocess
`.zip`	Zip format archive file of Java source code, no preprocessing

A filename with no recognized suffix is considered an object file to be linked. GCC's failure to recognize a particular filename suffix does not mean you are limited to using the suffixes listed previously to identify source or object files. You can use the `-x` *`lang`* option to identify the language used in one or more input files. The `lang` argument tells GCC the input language to expect to encounter in an input file regardless of its name, and is specific to different GCC compilers. For example, GCC's C compiler supports values for `lang` of `c`, `objective-c`, `c++`, `c-header`, `cpp-output`, `c++-cpp-output`, `assembler`, or `assembler-with-cpp`.

■ **Tip** The file suffixes listed for each GCC compiler are listed in the file gcc/language/lang-specs.h in the GCC source code, where *language* is one of `cp`, `ada`, `java`, `objc`, `treelang`, `objcp`, or `fortran`. If you are building your own GCC compilers and use a nonstandard file extension, you can modify this file to add your own extensions. However, using the `–x` *`lang`* option is a better approach if you are writing source code that you expect to share with others.

Table A-3 lists the command-line options you can use to exercise more control over the compilation process.

Table A-3. *GCC Output Options*

Option	Description
`-c`	Stops the compilation process before the link stage
`-E`	Terminates compilation after preprocessing
`-o` *`file`*	Writes output to the file specified by *file*
`-S`	Stops the compilation process after generating assembler code
`-x` *`lang`*	Sets the input language of subsequent files to `lang`
`-x none`	Turns off the definition of a previous `-x` *`lang`* option

When you use `-c`, the output is a link-ready object file, which has an `.o` filename extension. For each input file, GCC compilers generate a corresponding output file. Likewise, if you specify `-E`, the resulting output will be the preprocessed source code, which is sent to standard output. (GCC's Java compiler, which does not perform preprocessing since the concept is irrelevant in Java, generates no output at this point.) If you want to save the preprocessed output, you should redirect it to a file, either with command-line redirection or by using the `-o` option. If you use the `-S` option, each input

file results in an output file of assembly code with an `.s` extension. The `-o` file option enables you to specify the output filename, overriding the default output filename conventions.

Compiling a single source file using a GCC compiler is simple: just invoke the appropriate compiler, specifying the name of the source file as the argument, as in the following example of using the GCC C compiler, gcc, with a source file named myprog.c:

```
$ gcc myprog.c
$ ls -l
```

```
-rwxr-xr-x    1 wvh     users       13644 Oct  5 16:17 a.out
-rw-r--r--    1 wvh     users         220 Oct  5 16:17 myprog.c
```

The result on Linux and Unix systems is an executable file in the current directory named a.out, which you execute by typing **./a.out** in the directory containing the file. The name a.out is a historical artifact dating from C's earliest days. It stands for *assembler output* because, as you might expect, the first C-based executables were the output of assemblers. On Cygwin systems, you will wind up with a file named .a.exe that you can execute by typing either **./a** or **./a.exe** in the directory containing the file.

To define the name of the output file that a GCC compiler uses, use the `-o` option, as illustrated in the following example:

```
$ gcc myprog.c -o runme
$ ls -l
```

```
-rw-r--r--    1 wvh     users         220 Oct  5 16:17 myprog.c
-rwxr-xr-x    1 wvh     users       13644 Oct  5 16:28 runme
```

As you can see, GCC creates an executable file named runme in the current directory. The usual convention when compiling a single source file to executable format is to name the executable by dropping the file extension, so that myprog.c becomes myprog. Naturally, only the simplest programs consist of only a single source code file. More typically, programming projects consist of multiple source code files. In such a situation, you need to use the `-o` option in order to name the resulting binary, unless you intend to stick with the default a.out name. Keeping the name a.out is generally a bad thing because it does not give the user any idea what the binary actually does, and the chance of colliding with some other user's a.out binary is high.

Of course, you will also want to know how to compile multiple source files using GCC compilers. Again, the magic incantation is simple. To illustrate, suppose you are working in the C language and have a source file showdate.c that uses a function that is declared in helper.h and defined in helper.c. The standard way to compile these files, ignoring optimization, debugging, and other special cases, is the following:

```
$ gcc showdate.c helper.c
```

In this example, GCC's C compiler creates the final executable in a file named a.out on Linux and Unix systems (a.exe on Cygwin systems). In the absence of command-line options instructing otherwise, GCC compilers go through the entire compilation process: preprocessing (as appropriate, based on the type of input file), compilation, assembly, and linking. To specify the name of the output file, use the `-o` option, as in the following example:

```
$ gcc showdate.c helper.c -o showdate
```

This invocation, using `-o showdate`, leaves the compiled and linked executable in the file named showdate.

Using some of the other options listed in Table A-3 can be instructive. If you want to stop compilation after preprocessing using the `-E` option, be sure to use `-o` to specify an output filename or use output redirection. For example:

```
$ gcc -E helper.c -o helper.i
$ ls -l helper.*
```

```
-rw-r--r--    1 wvh     users      210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users       45 Oct  5 12:29 helper.h
-rw-r--r--    1 wvh     users    40440 Oct  5 13:08 helper.i
```

The `-o helper.i` option and argument saves the output of the preprocessor in the file helper.i. Notice that the preprocessed file is some 200 times larger than the source file. Also, bear in mind that except for the link stage, GCC compilers work on a file-by-file basis—each input file results in a corresponding output file with a filename extension appropriate to the stage at which compilation is stopped. This latter point is easier to see if you use the `-S` or `-c` options, as the following example illustrates:

```
$ gcc -S showdate.c helper.c
$ ls -l
```

```
total 20
-rw-r--r--    1 wvh     users         210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users          45 Oct  5 12:29 helper.h
-rw-r--r--    1 wvh     users         741 Oct  5 13:18 helper.s
-rw-r--r--    1 wvh     users         208 Oct  5 12:44 showdate.c
-rw-r--r--    1 wvh     users         700 Oct  5 13:18 showdate.s
```

In this case, I used the `-S` option, which stops compilation after the assembly stage and (in this case) leaves the resulting assembly code files, helper.s and showdate.s, which are the assembled versions of the corresponding C source code files. The next example uses `-c` to stop GCC's C compiler after the compilation process itself:

```
$ gcc -c showdate.c helper.c
$ ls -l
```

```
total 20
-rw-r--r--    1 wvh     users         210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users          45 Oct  5 12:29 helper.h
-rw-r--r--    1 wvh     users        1104 Oct  5 13:22 helper.o
-rw-r--r--    1 wvh     users         208 Oct  5 12:44 showdate.c
-rw-r--r--    1 wvh     users        1008 Oct  5 13:22 showdate.o
```

Finally, the following example shows how to use the `-x` option to force GCC to treat input files as source code files of a specific language. First, rename showdate.c to showdate.txt and then attempt to compile and link the program as shown here:

```
$ gcc showdate.txt helper.c -o showdate
```

```
showdate.txt: file not recognized: File format not recognized
collect2: ld returned 1 exit status
```

As you might expect, GCC's C compiler does not know how to "compile" a .txt file and compilation fails. To remedy this situation, use the `-x c` option to tell GCC that the input files following the `-x` option (showdate.txt and helper.c) are C source files, regardless of their output extension:

```
$ gcc –x c showdate.txt helper.c -o showdate
$ ls -l
```

```
total 28
-rw-r--r--    1 wvh     users          210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users           45 Oct  5 12:29 helper.h
-rwxr-xr-x    1 wvh     users        13893 Oct  5 13:38 showdate
-rw-r--r--    1 wvh     users          208 Oct  5 12:44 showdate.txt
```

It worked! Judicious use of the `-x` option with the `-c`, `-E`, and `-S` options enables you to exercise precise control over the compilation process. Although you do not ordinarily need to do so, you can walk through a complete compilation process one step at a time if you want to examine the output of each phase of compilation. The following example uses GCC's C compiler, gcc, but you could do the same thing with any other GCC compiler, with the exception of GCC's Java compiler, gcj, which does not do preprocessing, so you would simply skip the preprocessing step.

```
$ gcc -E helper.c -o helper.pre
$ gcc -E showdate.c -o showdate.pre
$ ls -l
```

```
total 92
-rw-r--r--    1 wvh     users          210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users           45 Oct  5 12:29 helper.h
-rw-r--r--    1 wvh     users        40440 Oct  5 13:44 helper.pre
-rw-r--r--    1 wvh     users          208 Oct  5 12:44 showdate.c
-rw-r--r--    1 wvh     users        37152 Oct  5 13:46 showdate.pre
```

I use the `-o` option to save the output in files with `.pre` filename extensions. Next, run the preprocessed files through the assembler:

```
$ gcc -S -x cpp-output helper.pre -o helper.as
$ gcc -S -x cpp-output showdate.pre -o showdate.as
$ ls -l
```

```
total 100
-rw-r--r--    1 wvh     users          741 Oct  5 13:47 helper.as
-rw-r--r--    1 wvh     users          210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users           45 Oct  5 12:29 helper.h
-rw-r--r--    1 wvh     users        40440 Oct  5 13:44 helper.pre
-rw-r--r--    1 wvh     users          700 Oct  5 13:47 showdate.as
-rw-r--r--    1 wvh     users          208 Oct  5 12:44 showdate.c
-rw-r--r--    1 wvh     users        37152 Oct  5 13:46 showdate.pre
```

This time, I use the `-o` option to save the assembler output using the filename extension `.as`. I use the `-x cpp-output` option because the assembler expects preprocessor output files to have the extension `.i` (for preprocessed C source code). Now, run the assembly code through actual compilation to produce object files:

```
$ gcc -c -x assembler helper.as showdate.as
$ ls -l
```

```
total 108
-rw-r--r--    1 wvh     users            741 Oct  5 13:47 helper.as
-rw-r--r--    1 wvh     users            210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users             45 Oct  5 12:29 helper.h
-rw-r--r--    1 wvh     users           1104 Oct  5 13:50 helper.o
-rw-r--r--    1 wvh     users          40440 Oct  5 13:44 helper.pre
-rw-r--r--    1 wvh     users            700 Oct  5 13:47 showdate.as
-rw-r--r--    1 wvh     users            208 Oct  5 12:44 showdate.c
-rw-r--r--    1 wvh     users           1008 Oct  5 13:50 showdate.o
-rw-r--r--    1 wvh     users          37152 Oct  5 13:46 showdate.pre
```

I use the `-x assembler` option to tell GCC's C compiler that the files helper.as and showdate.as are assembly language files. Finally, link the object files to create the executable, showdate.

```
$ gcc helper.o showdate.o -o showdate
$ ls -l
```

```
total 124
-rw-r--r--    1 wvh     users            741 Oct  5 13:47 helper.ass
-rw-r--r--    1 wvh     users            210 Oct  5 12:42 helper.c
-rw-r--r--    1 wvh     users             45 Oct  5 12:29 helper.h
-rw-r--r--    1 wvh     users           1104 Oct  5 13:50 helper.o
-rw-r--r--    1 wvh     users          40440 Oct  5 13:44 helper.pre
-rwxr-xr-x    1 wvh     users          13891 Oct  5 13:51 showdate
-rw-r--r--    1 wvh     users            700 Oct  5 13:47 showdate.ass
-rw-r--r--    1 wvh     users            208 Oct  5 12:44 showdate.c
-rw-r--r--    1 wvh     users           1008 Oct  5 13:50 showdate.o
-rw-r--r--    1 wvh     users          37152 Oct  5 13:46 showdate.pre
```

It should not take too much imagination to see that a project consisting of more than a few source code files would quickly become exceedingly tedious to compile from the command line, especially after you start adding search directories, optimizations, and other GCC options. The solution to this command-line tedium is the make utility, which is not discussed in this book due to space constraints (although it is touched upon in Chapter 7).

So what was the point of this exercise? First, it illustrates that GCC compilers, in this case GCC's C compiler, performs as advertised. More importantly, the `-E` option can be remarkably useful in C, Objective C, or C++ development when you are trying to track down a problem with a macro that does not behave as you expected. A popular C programming subgenre consists of preprocessor magic, sometimes referred to as *preprocessor abuse* or, as I like to refer to it, *Stupid Preprocessor Tricks* (with apologies to David Letterman). The typical scenario is that the preprocessor does not interpret your macro as you anticipated, causing compilation failure, error or warning messages, or bizarre runtime errors. By halting compilation after preprocessing you can examine the output, determine what you did wrong, and then correct the macro definition.

The value of `-S` becomes apparent if you want to hand-tune the assembly code the compiler generates. You can also use `-S` to see what kind of assembly output the compiler creates for a given block of code, or even a single statement. Being able to examine the compiler's assembly level output is educational in its own right and can help you debug a program that has a subtle bug. Naturally, to get the maximum benefit from GCC's assembly output feature, you have to know the target system's assembly language, or, as is more often the case, have the target CPU's reference manuals close at hand.

Controlling the Preprocessor

The options discussed in this section let you control the preprocessor used by GCC compilers, with the exception of GCC's Java compiler. Skip this section if you are using the GCC gcj compiler.

As mentioned earlier in this appendix, compilation stops after preprocessing if you specify the `-E` option to a GCC compiler. As you know, the preprocessor executes against each source code file before its output is handed off to phases of the other compilation process. Preprocessor options are listed in Table A-4.

Table A-4. *Preprocessor Options*

Option	Description
`-A-QUESTION=ANSWER`	Cancels setting *QUESTION* to *ANSWER*.
`-AQUESTION=ANSWER`	Sets the value of *QUESTION* to *ANSWER*.
`-D`*name*	Defines the preprocessor macro name with a value of 1.
`-D`*name*`=`*def*	Defines the preprocessor macro name with the value specified in *def*.
`-imacros` *file*	Processes *file* but only includes and preprocesses its macro definitions.
`-M`	Causes the preprocessor to output rules suitable for use with the make program rather than traditional preprocessor output. See the section titled "Alphabetical GCC Option Reference" later in this appendix for detailed information.
`-nostdinc`	Tells the compiler not to search the standard system directories for include files. Only the current directory and directories specified with the `-I` option will be searched. This option is only valid for C input files.
`-nostdinc++`	Tells the compiler not to search the standard system directories for include files. Only the current directory and directories specified with the `-I` option will be searched. This option is only valid for C++ input files.
`-std=`*std*	Identifies the standard to which the input file should conform, which can invoke special handling for certain constructs and sets the preprocessor's expectation of valid content. This is only used when preprocessing C and C++ input files. Valid values are `c89`, `c99`, `c9x`, `c++98`, `gnu89`, `gnu99`, `gnu9x`, `gnu++98`, `iso9899:1990`, `iso9899:199409`, `iso9899:1999`, and `iso9899:199x`.
`-U`*name*	Undefines any preprocessor macro name.
`-undef`	Undefines all system-specific macros, leaving common and standard macros defined.
`-w`, `-W`*option*	Control the type of warnings issued by the preprocessor. See the section later in this appendix titled "Enabling and Disabling Warning Messages" for detailed information.
`-Xpreprocessor` *option*	Passes the option specified by *option* to the preprocessor. Any arguments to options passed using the `-Xpreprocessor` option must themselves be preceded by a separate `-Xpreprocessor` option.

The -U*name* option cancels any definition of the macro name that was defined on a GCC compiler's command line using -D or in one of the source code files. Each instance of -D and -U is evaluated in the order specified on the command line. If you use the -imacros *file* option to specify that macros defined in *file* should be included, this inclusion takes place after all -D and -U options have been evaluated.

■ **Caution** Do not put spaces between -D and -U and their arguments or the definition will not work.

Consider Listing A-1. If the preprocessor macro DEUTSCH is defined, the output message will be "Hallo, Welt!" Otherwise, the message will be "Hello, World!"

Listing A-1. *A Sample C File Showing the Use of Preprocessor Macros*

```
#include <stdio.h>

int main (void)
{
#ifdef DEUTSCH
        puts ("Hallo, Welt!");
#else
        puts ("Hello, World!");
#endif
        return 0;
}
```

In the following example, I leave DEUTSCH undefined, so the program outputs the English language greeting:

```
$ gcc hallo.c -o hallo
$ ./hallo
```

```
Hello, World!
```

Specifying -DDEUTSCH on the GCC command line, however, defines the DEUTSCH macro with a value of 1, causing the compiled binary to issue the German version:

```
$ gcc hallo.c -o hallo -DDEUTSCH
$ ./hallo
```

```
Hallo, Welt!
```

With a little command-line or Makefile magic, deftly implemented macros, and the -D and -U options, you can enable and disable program features without having to edit your code simply by enabling and disabling preprocessor macros when you recompile a program. Of course, overuse of #ifdef...#endif blocks in code can make the code unreadable, so use them sparingly.

■ **Tip** Using a construct like `#if 0...#endif` provides a convenient way to comment out huge blocks of code without using the relevant programming language comment characters. Many languages do not support nested comments, most notably C and C++, which can make it tricky to comment out blocks of code that already contain comments. The `#if 0...#endif` construct is a slick way to work around this temporarily. By the way, remember to insert a comment at the beginning of the block explaining what you are doing.

Modifying Directory Search Paths

All of the GCC compilers search directories for various libraries. Some, such as GCC's C and C++ compilers, also search for definition (include) files. The basic GCC compiler framework provides options that enable you to manipulate the list of directories to search and the order in which they are searched. The extent to which this applies to the GCC compiler that you are using depends on that particular compiler. The examples in this section use the GCC C compiler, gcc, because modifying the default system directory search paths is most common when developing C language applications. Table A-5 lists the command-line options for modifying various directory search paths.

Table A-5. *Options for Modifying Directory Search Paths*

Option	Description
`-B` *prefix*	Instructs the compiler to add `prefix` to the names used to invoke its executable subprograms (such as cpp, cc1, as, and ld), libraries, and header files.
`-iquote` *dir*	Adds `dir` to the beginning of the list of directories searched for include files requested with `#include "file"`. Directories added using `-iquote` are not searched for include files specified via `#include <file>` (GCC 4.x and greater only; C, C++, and Objective C only).
`-I` *dir*	Adds `dir` to the list of directories searched for header files.
`-I-`	Limits the type of header files searched for when `-I` *dir* is specified. (Not available in GCC 4.x; replaced by the `-iquote` option's ability to restrict searching include directories to user header files).
`-L` *dir*	Adds `dir` to the list of directories searched for library files.
`-specs=`*file*	Reads the compiler spec file *file* after reading the standard spec file, making it possible to override the default values of arguments passed to GCC component programs.

If you use `-I` *dir* to add `dir` to the include directory search list, GCC compilers insert `dir` at the beginning of the standard include search path. This enables GCC compilers to search directories containing local and custom header files before searching the standard system include directories, enabling you to override system definitions and declarations if you choose. More often, however, you use `-I` to add directories to the header file search path rather than to override already defined functions. For example, if you are compiling a program that uses header files installed in `/usr/local/include/libxml2`, you would specify this extra directory as shown in the following example (the ellipsis indicates other arguments omitted for the sake of brevity):

```
$ gcc -I/usr/local/include/libxml2 [...] resize.c
```

This command causes the GCC C compiler, gcc, to look in `/usr/local/include/libxml2` for any header files included in resize.c before it looks in the standard header file locations.

The C, C++, or Objective C languages can include two different types of header files: *system header files*, which are those included using angle brackets (for example, #include <net/inet.h>), and *user header files*, which are those included using double quotes (for example, #include "log.h"). The default behavior of the `-I` option is to search the specified directories for both user and system header files. However, you can use the `-iquote` option to modify this behavior. All directories specified using the `-iquote` option will only be searched for user header files (those included using double quotes). Any include directories specified with `-I` will still be searched for all header files.

Consider the following #include directives at the top of a sample source code file named resize.c:

```
#include "libxml2/xmlops.h"
#include <netdev/devname.h>
```

If the directories containing the include files for both of these packages are subdirectories of the include directory `/usr/local/include`, you could compile this using the following gcc invocation:

```
$ gcc -I /usr/local/include resize.c
```

This would cause the include files /usr/local/include/netdev/devname.h and /usr/local/include/libxml2/xmlops.h to be used. However, suppose that these are header files that exist in multiple places and you want to use the version of libxml2/xmlops.h located in a custom directory but ignore the version of netdev/devname.h located under that same custom directory. Assuming that this custom directory is named `working`, you could execute the following command:

```
$ gcc –iquote working -I /usr/local/include resize.c
```

Because the directory `working` is specified on the command-line using `–iquote`, the include files working/libxml2/xmlops.h and /usr/local/include/netdev/devname.h will be used. Consider what would happen if the following #include directives were used in resize.c:

```
#include <libxml2/xmlops.h>
#include <netdev/devname.h>
```

In this case, both header files are included as system header files, and gcc would therefore use the versions located under the `/usr/local/include` directory that you specified using `–I`, even if you specified the `working` directory using the `–iquote` option.

■ **Tip** Multiple `-I` *`dir`* options can be specified. They are searched in the order specified, reading left to right.

■ **Note** GCC compilers prior to version 4 used the `–I-` option to differentiate between directories that should be searched for system and user header files. All included directories specified with `-I` before the occurrence of `-I-` would only be searched for user header files (those included using double quotes). Any include directories specified with `-I` after the occurrence of `-I-` would be searched for all header files. This option is deprecated and no longer supported in GCC 4.x compilers.

The `-L` *`dir`* option does for library files what `-I` *`dir`* does for header files: it adds `dir` to the beginning of the library directory search list, so that GCC compilers first search this directory for libraries specified using the `-l` option. The `–l` option is discussed in the section "Controlling the Linker."

The `-specs=file` option will be discussed later in this appendix in the section titled "Customizing GCC Compilers Using Spec Files and Spec Strings." After learning how to use spec strings, you can store multiple spec strings in a file and apply them as a group by replacing *file* with the name of the file containing the updated spec strings.

Passing Options to the Assembler

If you have options that you want GCC compilers to ignore and pass through to the assembler, use the `-Wa,opt` option. Like its sibling option for the linker, `-Wl,opt` (discussed in the next section, "Controlling the Linker"), you can specify multiple `opt` options by separating each option with a comma.

Controlling the Linker

Linking is the last step in the compilation process and refers to merging various object files into a single executable binary. GCC compilers assume that any file that does not end in a recognized suffix is an object file or a library. Refer to the list of recognized filename suffixes listed in the section titled "Controlling GCC Compiler Output" earlier in this appendix if you need a quick refresher. The linker knows how to tell the difference between object files (`.o` files) and library files (`.so` shared libraries or `.a` archive files) by analyzing the file contents. Note that options for controlling the linker will be ignored if you use the `-E`, `-c`, or `-S` options, which terminate the compiler before the link stage begins (after preprocessing, object file generation, and assembling, respectively). Table A-6 lists the options that you can use to control the GCC linker.

Table A-6. *Link Options*

Option	Description
`-lname`	Searches the library named *name* when linking.
`-nodefaultlibs`	Specifies not to use the standard system libraries when linking.
`-nostartfiles`	Ignores the standard system startup files when linking. System libraries are used unless `-nostdlib` is also specified.
`-nostdlib`	Specifies not to use the standard system startup files or libraries when linking (equivalent to specifying `-nostartfiles -nodefaultlibs`).
`-s`	Strips all symbol table and relocation information from the completed binary.
`-shared`	Produces a shared object that can then be linked with other objects to form an executable.
`-shared-libgcc`	Uses the shared libgcc library, if available, on systems that support shared libraries.
`-static`	Forces linking against static libraries on systems that default to linking with shared libraries.
`-static-libgcc`	Uses the statically linked libgcc library, if available, on systems that support shared libraries.
`-u sym`	Behaves as if the symbol `sym` is undefined, which forces the linker to link in the library modules that define it.
`-Wl,opt`	Passes `opt` as an option to the linker.
`-Xlinker opt`	Passes `opt` as an option to the linker.

I have already mentioned the `-L` *`dir`* option for adding directories to the library search patch. The complementary *`-lname`* option enables you to specify additional library files to search for given function definitions. Each library specified with *`-lname`* refers to a file named lib*name*.a or lib*name*.so, which is searched for in the standard library search path, plus any additional directories specified by `-L` options. Most libraries are simple archive files that contain a collection of object files and are produced by the Linux/Unix ar utility. The linker processes the archive file by searching it for members that define symbols (function names) which have been referenced but not yet defined.

The differences between specifying an object filename (such as name.o) and using an *`-lname`* option are that

- Specifying `-l` embeds *name* between `lib` and the library suffix (either `.a` or `.so` if you are using a version of gcc that was built with shared library support).
- The format of library files and object files differs.
- Using `-l` searches several directories for the resulting library, based on your library search path.

For example, provided you have created the archive file libmy.a (using the ar program), the following two gcc invocations are equivalent:

```
$ gcc myprog.o libmy.o -o myprog
$ gcc myprog.o -o myprog -L. –lmy
```

Listings A-2, A-3, and A-4 show some sample code that I will use to verify the equivalence of these two gcc command lines.

Listing A-2. *A Sample C Program, swapme.c*

```
#include <stdio.h>
#include "myfile.h"

int main (void)
{
        int i = 1;
        int j = 2;

        printf ("%d %d\n", i, j);
        swap (&i, &j);
        printf ("%d %d\n", i, j);

        return 0;
}
```

Listing A-3. *A Sample Include File, myfile.h*

```
void swap(int *, int *);
```

Listing A-4. *A Sample C File, myfile.c*

```
#include "myfile.h"
void swap (int *i, int *j)
{
        int t = *i;
        *i = *j;
        *j = t;
}
```

You can demonstrate the equivalence of building and linking the code in Listings A-2, A-3, and A-4 as stand-alone object files and as an object file and a library:

1. Compile myfile.c and swapme.c to object code.

```
$ gcc -c myfile.c
$ gcc -c swapme.c
```

2. Link the resulting object files into the final binary, swapme, and then run the program to demonstrate that it works.

```
$ gcc myfile.o swapme.o -o swapme
$ ./swapme
```

```
1 2
2 1
```

3. Delete the binary.

```
$ rm swapme
```

4. Use the ar command to create an archive file named libmy.a from the myfile.o object file.

```
$ ar rcs libmy.a myfile.o
```

5. Link the object file swapme.o and the archive file as shown in the following command. This command tells the GCC C compiler to search the current directory ('.') for libraries and to load the contents of a library named libmy.a.

```
$ gcc swapme.o -o swapme -L. -lmy
```

6. Run the program again to convince yourself that it still works.

```
$ ./swapme
```

```
1 2
2 1
```

It makes a difference where in the command line you specify *-lname* because the linker searches for and processes library and object files in the order they appear on the command line. Thus, foo.o -lbaz bar.o searches library file libbaz.a after file processing foo.o but before processing bar.o. If bar.o refers to functions in libbaz.a, those functions may not be loaded.

■ **Tip** If you are specifying your own libraries and not trying to override standard libraries used by your GCC compiler, a good general rule is to add library references at the end of a GCC compiler command line to ensure that they are searched for references after all source files have been compiled or examined for external references.

If for some reason you do not want the linker to use the standard libraries, specify `-nodefaultlibs` and specify the `-L` and `-l` options to point at your replacements for functions that are traditionally defined in the standard libraries. The linker will disregard the standard libraries and use only the libraries you specify. The standard startup files will still be used, though, unless you also use `-nostartfiles`.

As noted in Table A-6, `-nostdlib` is the equivalent of specifying both `-nostartfiles` and `-nodefaultlibs`; the linker will disregard the standard startup files and only the libraries you specified with `-L` and `-l` will be passed to the linker.

■ **Note** When you specify `-nodefaultlib`, `-nostartfiles`, or `-nostdlib`, GCC still might generate internal calls to `memcmp()`, `memset()`, and `memcpy()` in System V Unix and ISO C environments, or calls to `bcopy()` and `bzero()` in BSD Unix environments. These entries are usually resolved by entries in the standard C library (libc). You should supply entry points for `memcmp()`, `memset()`, `memcpy()`, `bcopy()`, and `bzero()` when using one of these three linker options.

One of the standard libraries bypassed by `-nostdlib` and `-nodefaultlibs` is libgcc.a. The libgcc.a library contains internal subroutines that GCC compilers use to compensate for shortcomings of particular systems and to provide for special needs required by some languages. As a result, you need the functions defined in libgcc.a even when you want to avoid other standard libraries. Thus, if you specify `-nostdlib` or `-nodefaultlibs`, make sure you also specify `-lgcc` as well to avoid unresolved references to internal GCC library subroutines.

The `-static` and `-shared` options are only used on systems that support shared libraries. GCC supports static libraries on all systems that I have ever encountered. Not all systems, however, support shared libraries, often because the underlying operating system does not support dynamic loading.

On systems that provide libgcc as a shared library, you can specify `-static-libgcc` or `-shared-libgcc` to force the use of either the static or the shared version of libgcc, respectively. There are several situations in which an application should use the shared libgcc instead of the static version. The most common case occurs when compiling C++ and Java programs that may throw and catch exceptions across different shared libraries. In this case, all libraries as well as the application itself should use the shared libgcc. Accordingly, the g++ (c++) and gcj compiler drivers automatically add `-shared-libgcc` whenever you build a shared library or a main executable and would ordinarily be using the static version of libgcc by default.

On the other hand, the driver for the GCC C Compiler, gcc, does not always link against the shared libgcc, especially when creating shared libraries. The issue is one of efficiency. If gcc determines that you have a GNU linker that does not support the link option `--eh-frame-hdr`, gcc links the shared libgcc into shared libraries. If the GNU linker does support the `--eh-frame-hdr` option, gcc links with the static version of libgcc. The static version allows exceptions to propagate properly through such shared libraries, without incurring relocation costs at library load time.

In short, if a library or the primary binary will throw or catch exceptions, you should link it using the g++ (c++) compiler driver (if you are using C++) or the gcj compiler driver (if you are using Java). Otherwise, use the option `-shared-libgcc` so that the library or main program is linked with the shared libgcc.

The final option controlling the linker is the rather odd looking `-Wl,`*`opt`*. This tells GCC to pass the linker option `opt` through directly to the linker. You can specify multiple `opt` options by separating each one with a comma (`-Wl,`*`opt1,op2,opt3`*).

Enabling and Disabling Warning Messages

A warning is a diagnostic message that identifies a code construct that might potentially be an error. GCC also emits diagnostic messages when it encounters code or usage that looks questionable or ambiguous. For the sake of discussion, I divide the GCC compilers' handling of warning messages into two groups: general options that control the number and types of warnings that a compiler emits, and options that affect language features or that are language-specific. I will start with the

options that control the overall handling of warnings by GCC compilers. Table A-7 shows the most commonly used options that control warnings (see the "Alphabetical GCC Option Reference" section of this appendix for the complete mind-numbing list). Some of these options are specific to certain GCC compilers, most commonly GCC's C compiler—the explanation of the warning should make it clear whether an option applies to the GCC compiler that you are using. Specifying warning options that do not apply to your compiler is not flagged as an error by the GCC compilers.

Table A-7. *General GCC Warning Options*

Option	Description
`-fsyntax-only`	Performs a syntax check but does not compile the code.
`-pedantic`	Issues all warnings required by ISO standards and rejects GNU extensions, traditional C constructs, and C++ features used in C code.
`-pedantic-errors`	Converts warnings issued by `-pedantic` into errors that halt compilation.
`-w`	Disables all warnings, including those issued by the GNU preprocessor.
`-W`	Displays extra warning messages for certain situations.
`-Wall`	Enables all of the warnings about code constructions that most users consider questionable, dangerous, or easy to eliminate with code modifications. It was originally intended to cause the compiler to display all warnings, but there are now so many types of possible warnings that there are exceptions to this rule. For example, this option does not activate some of the more granular formatting warnings such as `–Wformat=2`, `-Wformat-nonliteral`, `Wformat-security`, and `-Wformat-y2k`, the additional warning cases specified by `–Wextra`, and many stylistic C++ warnings.
`-Wbad-function-cast`	Emits a warning if a function call is cast to an incompatible type (C only).
`-Wcast-qual`	Displays a warning when a typecast removes a type qualifier.
`-Wchar-subscripts`	Emits a warning when a `char` variable is used as a subscript.
`-Wcomment`	Emits a warning when nested comments are detected.
`-Wconversion`	Emits a warning if a negative integer constant is assigned to an unsigned type.
`-Wdisabled-optimization`	Displays a warning when a requested optimization is not performed.
`-Werror`	Converts all warnings into hard errors that halt compilation of the indicated translation unit.

Table A-7. *General GCC Warning Options (Continued)*

Option	Description
`-Werror-implicit-sfunction-declaration`	Emits an error when a function is not explicitly declared before first use.
`-Wextra`	Emits the same warnings as `-W`, but provides a more memorable option name.
`-Wfloat-equal`	Emits a warning when floating-point values are compared for equality.
`-Wformat`	Issues a warning when arguments supplied to `printf()` and friends do not match the specified format string.
`-Wformat=2`	The same as specifying `-Wformat -Wformat -nonliteral Wformat-security -Wformat-y2k`.
`-Wformat-nonliteral`	Issues a warning, if `-Wformat` is also specified, about any formatting strings that are not literal strings and therefore cannot be checked.
`-Wformat-security`	Issues a warning, if `-Wformat` is also specified, about arguments to `printf()` and friends that pose potential security problems.
`-Wformat-y2k`	Issues a warning, if `-Wformat` is also specified, if any formatting strings in calls to `printf()` and friends would display a non-Y2K-compliant (i.e., two-digit) year.
`-Wimplicit`	Combines `-Wimplicit-int` and `-Wimplicit-function-declaration`.
`-Wimplicit-int`	Emits a warning when a declaration does not specify a type.
`-Wimplicit-function-declaration`	Emits a warning when a function is not explicitly declared before first use.
`-Winline`	Issues a warning when functions declared inline are not inlined.
`-Wlarger-than-`*n*	Emits a warning when an object larger than *n* bytes is defined.
`-Wmain`	Emits a warning if `main()`'s return type or declaration is malformed.
`-Wmissing-braces`	Displays a warning when aggregate or union initializers are improperly bracketed.
`-Wmissing-declarations`	Displays a warning if a global function is defined without being declared.
`-Wnested-externs`	Issues a warning if an extern declaration occurs in a function definition.
`-Wno-deprecated-declarations`	Disables warnings about use of features that are marked as deprecated.

Table A-7. *General GCC Warning Options (Continued)*

Option	Description
`-Wno-div-by-zero`	Disables warnings issued if (integer) division by zero is detected.
`-Wno-format-y2k`	Disables warnings about `strftime()` formats that result in two-digit years.
`-Wno-format-extra-args`	Disables warnings about extra arguments to `printf()` and friends.
`-Wno-long-long`	Disables warnings about using the `long long` type.
`-Wno-multichar`	Disables warnings issued if multibyte characters are used.
`-Wpadded`	Issues a warning when a structure is padded for alignment purposes.
`-Wparentheses`	Issues a warning about ambiguous or potentially confusing use (or misuse or disuse) of parentheses.
`-Wpoint-arith`	Issues a warning when a code operation or structure depends on the size of a function type or a `void *` pointer.
`-Wredundant-decls`	Displays a warning when an object is multiply-declared in the same scope, even in contexts in which multiple declaration is permitted and valid.
`-Wreturn-type`	Issues a warning if a function's return type is not specified or if it returns a value but is declared void.
`-Wsequence-point`	Flags code that violates C sequence point rules.
`-Wshadow`	Displays a warning if a locally declared variable shadows another local or global variable, parameter, or built-in function.
`-Wsign-compare`	Issues a warning when comparisons between signed and unsigned values might produce incorrect results because of type conversions.
`-Wstrict-protoypes`	Displays a warning if a function is defined with specifying argument types (C only).
`-Wswitch`	Emits a warning about switch statements with enumerated values for unhandled cases.
`-Wsystem-headers`	Issues warnings for code in system headers as if they occurred in your own code.
`-Wtraditional`	Emits a warning about code constructs that behave differently between ISO and traditional C and about ISO C features with no parallel in traditional C (C only).
`-Wtrigraphs`	Emits warnings if any trigraphs are encountered outside of comment blocks.
`-Wundef`	Issues a warning if an undefined identifier is used in a `#if...#endif` construct.

Table A-7. *General GCC Warning Options (Continued)*

Option	Description
`-Wuninitialized`	Emits a warning when automatic variables are used without being initialized.
`-Wunknown-pragmas`	Displays a warning if a `#pragma` is used that GCC does not recognize.
`-Wunreachable-code`	Emits a warning about code that never executes.
`-Wunused`	Combines all of the listed `-Wunused` options.
`-Wunused-function`	Issues a warning about declared functions that are never defined.
`-Wunused-label`	Issues a warning about declared labels that are never used.
`-Wunused-parameter`	Issues a warning about declared function parameters that are never used.
`-Wunused-value`	Issues a warning about computed results that are never used.
`-Wunused-variable`	Issues a warning about declared variables that are never used.

The chapters of this book that discuss using each of the compilers that are part of GCC explain the options that are related to those compilers in more detail.

In general, the group of options that fall under the `-Wunused` category are particularly helpful. In optimization passes, GCC compilers do a good job of optimizing away unused objects; but if you disable optimization, the unused cruft bloats the code. More generally, unused objects and unreachable code (detected with `-Wunreachable-code`) are often signs of sloppy coding or faulty design. My own preference is to use the plain `-Wunused` option, which catches all unused objects. If you prefer otherwise, you can use any combination of the five options that begin with `-Wunused-`.

Listing A-5 is a short example of a C program with an unused variable.

Listing A-5. *A Sample C File, unused.c, with an Unused Variable*

```
int main (void)
{
        int i = 10;
        return 0;
}
```

As you can see, the program defines the `int` variable `i`, but never does anything with it. Here is the output from the gcc compiler when compiling unused.c with no options:

```
$ gcc unused.c
```

Well, perhaps I really should have written "here is the lack of output from the gcc compiler when compiling the file unused.c with no options." Even adding the `-ansi` and `-pedantic` options does not change the compiler's output. However, here is gcc's output when I add the `–Wunused` option:

```
$ gcc -Wunused unused.c -ansi -pedantic
unused.c: In function 'main':
unused.c:3: warning: unused variable 'i'
```

Each of the warning options discussed in this section results in similar output that identifies the translation unit and line number in which a problem was detected and a brief message describing the problem. I encourage you to experiment with the warning options. Given the rich set of choices, you can debug and improve the overall quality and readability of your code just by compiling with a judiciously chosen set of warning options. Many companies specify that code must compile cleanly with `-Wall` and `-Werror` in order to be considered code complete.

Adding Debugging Information

Referring to a program he used to illustrate a point, Donald Knuth once wrote "Beware of bugs in the above code; we have only proved it correct, not tried it." Bugs are an inescapable reality of coding. Accordingly, the GCC compilers support a number of options to help you debug code. If you have used GCC compilers before, you are no doubt familiar with the `-g` and `-ggdb` options, which instruct the GCC compilers to embed debugging information in executables in order to facilitate debugging. These two options hardly exhaust GCC's repertoire of debugging support. In this section I will show you not only the `-g` and `-ggdb` switches, but also other lesser-known GCC options that augment the debugging process. Or, to put it more whimsically, this section helps you debug debugging.

I will start with Table A-8, which lists and briefly describes the most commonly used debugging options supported by the GCC compilers.

Table A-8. *Debugging Options*

Option	Description
`-d[`*mod*`]`	Generates debugging dumps at compilation points specified by *mod*. See Table A-9 for a list of valid values for *mod*.
`-fdump-class-hierarchy`, `-fdump-class-hierarchy-`*option*	Dumps the class hierarchy and vtable information, subject to the control expressed by *option*, if specified.
`-fdump-ipa-`*switch*	Dumps the tree structure for various stages of interprocedural analysis to a file, subject to the value of *switch*, which is also used to determine the file extension used for the dump files. Possible values for *switch* are `all` and `cgraph`. Specifying `all` currently only dumps the same call graph information as specifying `cgraph`, but this may change in the future.
`-fdump-translation-unit`, `-fdump-translation-unit-`*option*	Dump the tree structure for an entire unit, subject to the control expressed by *option*, if specified.
`-fdump-tree-`*switch*, `-fdump-tree-`*option*	Dump the intermediate representation of the language tree structure, subject to the controls expressed by *switch* and *option*, if specified.
`-fdump-unnumbered`	Inhibits dumping line and instruction numbers in debugging dumps (see `-d[mod]`).
`-fmem-report`	Displays memory allocation statistics for each phase of the compiler. Note that this information is about the compiler itself, not the code that is being compiled.

Table A-8. *Debugging Options (Continued)*

Option	Description
`-fpretend-float`	Pretends that the target system has the same floating-point format as the host system.
`-fprofile-arcs`	Instruments code paths, creating program dumps showing how often each path is taken during program execution.
`-ftest-coverage`	Generates data for the gcov test coverage utility.
`-ftime-report`	Displays performance statistics for each compiler phase.
`-g[`*n*`]`	Generates level *n* debugging information in the system's native debugging format (*n* defaults to 2).
`-gcoff[`*n*`]`	Generates level *n* debugging information in the COFF format (Common Object File Format) (*n* defaults to 2).
`-gdwarf`	Generates debugging information in the DWARF format (Debug With Arbitrary Record Format).
`-gdwarf+`	Generates debugging information in the DWARF format, using extensions specific to GDB, the GNU debugger.
`-gdwarf-2`	Generates debugging information in the DWARF version 2 format.
`-ggdb[`*n*`]`	Generates level *n* debugging information that only GDB can fully exploit.
`-gstabs[`*n*`]`	Generates level *n* debugging information in the STABS format (*n* defaults to 2).
`-gstabs+`	Generates debugging information in the STABS format (using extensions specific to GDB).
`-gvms[`*n*`]`	Generates level *n* debugging information in the VMS format (*n* defaults to 2).
`-gxcoff[`*n*`]`	Generates level *n* debugging information in the XCOFF format (*n* defaults to 2).
`-gxcoff+`	Generates debugging information in the XCOFF format, using extensions specific to GDB.
`-p`	Generates code that dumps profiling information used by the prof program.
`-pg`	Generates code that dumps profiling information used by the gprof program.
`-Q`	Displays the name of each function compiled and how long each compiler phase takes.
`-time`	Displays the CPU time consumed by each phase of the compilation sequence.

■ **Caution** The `-a` and `-ax` options for displaying the profiling information on basic blocks in GCC prior to version 3.0 have been removed, though they still appear in some later versions of the GCC documentation.

Quite a few options, eh? Fortunately, and as is often the case with GCC, you only have to concern yourself with a small subset of the available options and capabilities at any given time or for any given project. In fact, most of the time, you can accomplish a great deal just using `-g` or `-ggdb`. Before getting started, though, I will make short work of a few options that I will discuss elsewhere: `-fprofile-arcs`, `-ftest-coverage`, `-p`, and `-pg`. Although listed as some of GCC's debugging options, they are best and most often used to profile code or to debug the compiler itself. In connection with their use in code profiling and code analysis, I discuss these options at length in Chapter 6.

The option `-d`*mod*, referred to as the *dump option* in this section, tells GCC to emit debugging dumps during compilation at the times specified by *mod*, which can be one of almost every letter in the alphabet (see Table A-9). These options are almost exclusively used for debugging the compiler itself, but you might find the debugging dumps informative or instructional if you want to learn the deep voodoo of GCC compilers. Each dump is left in a file, usually named by appending the compiler's pass number and a phrase indicating the type of dump to the source file's name (for example, myprog.c.14.bbro). Table A-9 lists the possible values for *mod*, briefly describes the corresponding compiler pass, and includes the name of the dump file, when applicable.

Table A-9. *Arguments for the Dump Option*

Argument	Description	File
`a`	Produces all dumps except those produced by `m`, `p`, `P`, `v`, `x`, and `y`	N/A
`A`	Annotates the assembler output with miscellaneous debugging information	N/A
`b`	Dumps after computing the branch probabilities pass	*file*.14.bp
`B`	Dumps after the block reordering pass	*file*.29.bbro
`c`	Dumps after the instruction combination	*file*.16.combine
`C`	Dumps after the first if-conversion	*file*.17.ce
`d`	Dumps after delayed branch scheduling	*file*.31.dbr
`D`	Dumps all macro definitions after preprocessing	N/A
`e`	Dumps after performing SSA (static single assignment) optimizations	*file*.04.ssa, *file*.07.ussa
`E`	Dumps after the second if-conversion pass	*file*.26.ce2
`f`	Dumps after life analysis	*file*.15.life
`F`	Dumps after purging ADDRESSOF codes	*file*.09.addressof
`g`	Dumps after global register allocation	*file*.21.greg
`G`	Dumps after global common subexpression elimination (GCSE)	*file*.10.gcse
`h`	Dumps after finalizing of EH handling code	*file*.02.eh

Table A-9. *Arguments for the Dump Option (Continued)*

Argument	Description	File
`I`	Dumps after the sibling call optimization pass	*file*.01.sibling
`j`	Dumps after the first jump optimization pass	*file*.03.jump
`k`	Dumps after the register to stack conversion pass	*file*.28.stack
`l`	Dumps after local register allocation	*file*.20.lreg
`L`	Dumps after the loop optimization pass	*file*.11.loop
`M`	Dumps after the machine-dependent reorganization pass	*file*.30.mach
`m`	Prints statistics on memory usage at the end of the run	Standard error
`n`	Dumps after register renumbering	*file*.25.rnreg
`N`	Dumps after the register move pass	*file*.18.regmove
`o`	Dumps after the post-reload optimization pass	*file*.22.postreload
`p`	Annotates the assembler output with a comment indicating which pattern and alternative was used	N/A
`P`	Dumps the assembler output with the equivalent RTL (register transfer level) code as a comment before each instruction and enables `-dp` annotations	N/A
`r`	Dumps after RTL generation	*file*.00.rtl
`R`	Dumps after the second scheduling pass	*file*.27.sched2
`s`	Dumps after the first common subexpression elimination (CSE) and jump optimization pass	*file*.08.cse
`S`	Dumps after the first scheduling pass	*file*.19.sched
`t`	Dumps after the second CSE pass and post-CSE jump optimization	*file*.12.cse2
`v`	Dumps a representation of the control flow graph for each requested dump file (except file.00.rtl)	*file.pass*.vcg
`w`	Dumps after the second flow analysis pass	*file*.23.flow2
`x`	Generates RTL for a function instead of compiling it (used with `-dr`)	N/A
`X`	Dumps after the SSA dead-code elimination pass	*file*.06.ssadce
`y`	Dumps debugging information during parsing	Standard error
`z`	Dumps after the peephole pass	*file*.24.peephole2

■ **Caution** The GCC info documentation lists the `-dk` option twice, once outputting the file file.28.stack and once outputting the file file.32.stack. My testing indicates that only the file.28.stack is generated.

Although the options listed in Table A-9 are most often used for debugging the compiler itself, they can also be used as an analytic aid for any program. The `-dv` option is particularly useful because its output can be used with a third-party program to generate a graphical display that corresponds to the dump option with which it was used. For example, if you compile the program myprog.c using the option `-dCv`, you will wind up with the files myprog.c.17.ce and myprog.c.17.ce.vcg, in addition to the normal compiler output.

```
$ gcc -O -dCv myprog.c -o myprog
$ ls -l myprog*
```

```
-rw-r--r--    1 wvh     users      220 Oct  5 16:17 myprog.c
-rw-r--r--    1 wvh     users     4233 Oct 16 23:04 myprog.c.17.ce
-rw-r--r--    1 wvh     users     4875 Oct 16 23:04 myprog.c.17.ce.vcg
```

The `C` argument to `-d` dumps a flow control graph after the first if-conversion pass, which means that you need to enable optimization (see Chapter 5). The `v` argument creates a second dump file that contains the same information in a format that the Visualization of Compiler Graphs (VCG) tool can read and convert into a pretty graph. VCG is available for Unix and Windows systems. More information and downloadable versions in source and binary format are available from the VCG Web site at `http://rw4.cs.uni-sb.de/users/sander/html/gsvcg1.html`. VCG is developed and maintained independently of GCC.

Customizing GCC Compilers

The default behavior of the GCC compilers is usually what most people want—if that wasn't the case, the compilers would probably behave differently. GCC usually "does the right thing" and, at least in my experience, rarely violates the principle of least surprise. As you have no doubt begun to appreciate by now, if you want GCC to do something, chances are pretty good that it has a command-line option that will make it do so. Nevertheless, what should you do if you dislike GCC's default behavior, get tired of typing the same command-line option, and do not know enough to modify GCC's code? Naturally, you customize GCC using environment variables (as explained in the next section, "Customizing GCC Compilers Using Environment Variables") or specification (spec) strings, as you will learn in the section titled "Customizing GCC Compilers Using Spec Files and Spec Strings."

Customizing GCC Compilers Using Environment Variables

This section describes environment variables that affect how the GCC compilers operate. Some of these environment variables define directories or prefixes to use when searching for various kinds of files, while others control other features of the runtime environment used by the GCC compilers. In all cases, options specified on the command line always override options specified via environment values. Similarly, GCC compiler options defined in the environment always override the options built into a specific compiler, that is, those constituting each compiler's default configuration.

Some of the options in this section are specific to certain GCC compilers—these will, of course, have no effect if you are using a GCC compiler that does not reference them. If an environment variable is specific to a certain compiler or input language, the discussion of that environment variable identifies that fact.

The first two environment variables, `COMPILER_PATH` and `GCC_EXEC_PREFIX`, control modification of GCC's search path. `COMPILER_PATH` contains a colon-delimited list of directories, comparable to the familiar Unix `PATH` environment variable. GCC uses the directories specified in `COMPILER_PATH`, if any, to find subprograms, such as cc1, collect2, and cpp0, in its default search path or subprograms with the prefix specified by `GCC_EXEC_PREFIX`.

GCC_EXEC_PREFIX, if set, specifies the prefix to use in the names of the subprograms executed by the compiler. This variable does not necessarily specify a directory name because GCC does not add a slash (/) to the front of the prefix when it combines GCC_EXEC_PREFIX with the name of a subprogram. So, if GCC_EXEC_PREFIX is /usr/lib/gcc-lib/i586-suse-linux/4.1/, GCC attempts to invoke the subprogram cc1 as /usr/lib/gcc-lib/i586-suse-linux/4.1/cc1. If GCC_EXEC_PREFIX is custom-, GCC compilers attempt to invoke the subprogram cc1 as custom-cc1.

■ **Note** The GCC_EXEC_PREFIX environment variable does not seem to work under Cygwin. I am unclear if this is a function of Cygwin's emulation environment or a problem in a specific version of GCC.

If GCC_EXEC_PREFIX is unset, GCC uses a heuristic to derive the proper prefix based on the path by which you invoked GCC. The default value is PREFIX/lib/gcc-lib/, where PREFIX is the value of --prefix you specified when you configured the GCC script. If you specify -B on the command line, as noted earlier, the command-line specification overrides GCC_EXEC_PREFIX.

■ **Note** GCC_EXEC_PREFIX also enables GCC to find object files, such as crt0.o and crtbegin.o, used during the link phase.

C_INCLUDE_PATH, used by the C preprocessor, specifies a list of directories that will be searched when preprocessing a C file. This option is equivalent to the -isystem option, except that paths specified in C_INCLUDE_PATH are searched after directories specified with -isystem.

Another environment variable, CPATH, used by the C preprocessor, specifies a list of directories to search for included files when preprocessing C programs. This option is equivalent to the -I option, except that directories specified in CPATH are searched after directories specified using -I. CPATH is always used if defined, and used before C_INCLUDE_PATH, CPLUS_INCLUDE_PATH, and OBJC_INCLUDE_PATH if they are defined.

CPLUS_INCLUDE_PATH functions comparably to C_INCLUDE_PATH, but applies to C++ compilations rather than C compilations. CPLUS_INCLUDE_PATH specifies a list of directories that will be searched during preprocessing of a C++ file. This option is equivalent to the -isystem option, except that paths specified in CPLUS_INCLUDE_PATH are searched after directories specified with -isystem.

Finally, OBJC_INCLUDE_PATH contains a list of directories to search when preprocessing Objective C source files. This option is equivalent to the -isystem option, except that paths specified in OBJC_INCLUDE_PATH are searched after directories specified with -isystem.

As discussed later in this appendix in the section titled "Alphabetical GCC Option Reference," GCC's -M options can be used to tell GCC's C compiler to generate dependency output for all nonsystem header files (system header files are ignored) for the make utility. The environment variable DEPENDENCIES_OUTPUT customizes this behavior if set. If DEPENDENCIES_OUTPUT is a filename, the generated make rules are written to that file, and GCC guesses the target name based on the name of the source file. If DEPENDENCIES_OUTPUT is specified as *FILE TARGET*, the rules are written to the file *FILE* using *TARGET* as the target name. Using DEPENDENCIES_OUTPUT is equivalent to combining the options -MM and -MF, and optionally -MT, except that using DEPENDENCIES_OUTPUT enables compilation to proceed while still saving the text-format make rules in a file.

LIBRARY_PATH is a colon-delimited list in which GCC searches for special linker files. LIBRARY_PATH is searched after GCC_EXEC_PREFIX. If you use GCC for linking, that is, if you do not invoke the linker directly, GCC uses the directories specified in LIBRARY_PATH, if any, to find libraries specified with -L and -l (in that order).

GCC tries to play nice with language and locale information, so it is sensitive to the presence of the LANG, LC_ALL, LC_CTYPE, and LC_MESSAGES environment variables. GCC uses these variables to select the character set to use when the C and C++ compilers parse character literals, string literals, and comments. If configured for multibyte characters, GCC understands JIS, SJIS, and EUCJP characters if LANG has the value C-JIS, C-SJIS, or C-EUCJP, respectively. Otherwise, GCC uses the locale functions mblen() and mbtowc() to process multibyte characters.

Whereas LANG determines how GCC compilers process character sets, LC_ALL, LC_CTYPE, and LC_MESSAGES control how GCC uses locale information to interpret and process other elements of locale customization. GCC uses LC_CTYPE, for example, to determine the character boundaries in a string—some multibyte character encodings contain quote and escape characters that are ordinarily interpreted to denote an escape character or the end of a string. LC_MESSAGES informs GCC of the language to use when emitting diagnostic messages.

If LC_CTYPE or LC_MESSAGES are not set, they default to the value of the LANG environment variable. If set, the LC_ALL environment variable overrides the values of the LC_CTYPE and LC_MESSAGES environment variables. If none of these variables are set, GCC defaults to the traditional behavior of a C compiler in a U.S. English setting.

Finally, TMPDIR defines the directory to use for temporary files, which is normally /tmp on Unix systems. GCC uses temporary files to hold the output of one compilation stage, which will be used as input to the next stage. For example, the output of the preprocessor is the input to the compiler proper. One way to speed up any GCC compiler is to define TMPDIR to point to a RAM disk or a Linux tmpfs filesystem, which bypasses the overhead and limitations of physical disk reads and writes during compilation. If TMPDIR is not set, GCC uses the default temporary directory, which varies according to system.

Customizing GCC Compilers with Spec Files and Spec Strings

One of the points I have tried to emphasize throughout this book is that all GCC compilers are actually a sequence of programs that are executed in order. The idea of writing smaller programs that do one thing and do it well, and then chaining them together, is the classic Unix programming paradigm. For example, gcc is a driver program that does its job by invoking a sequence of other programs to do the heavy lifting of preprocessing, compiling, assembling, and linking. In general, the command-line parameters and filename arguments that you pass to any GCC compiler help it decide which helper and back-end programs to invoke and how to invoke them. Spec strings control the invocation of the helper programs.

A *spec string* is a list of options passed to a program. Spec strings can contain variable text substituted into a command line, or text that might or might not be inserted into a command line based on a conditional statement (in the spec string). Using the flexible constructs provided by spec strings, it is possible to generate quite complex command lines. A *spec file* is a plain text file that contains spec strings. Spec files contain multiple directives separated by blank lines. The type of directive is determined by the first nonwhitespace character on the line. In most cases there is one spec string for each program that a GCC compiler can invoke, but a few programs have multiple spec strings to control their behavior.

■ **Note** Prior to version 4.0 of GCC, the spec strings used by default by all of the GCC compilers were generated and stored in a single text file. It was common practice to query and potentially modify aspects of the behavior of various GCC compilers by examining and modifying this file. This file was typically installed as *prefix*/lib/gcc/*system-type*/*gcc-version*/specs. For example, on a 64-bit AMD system, the specs file for a set of GCC 3.4.4 compilers installed into `/usr/local/gcc3.4.4` would have been the file /usr/local/gcc-3.4.4/lib/gcc/x86_64-unknown-linux/gnu/3.4.4/specs. With GCC 4.0 and greater, the spec strings used by the GCC compilers are no longer stored in a single text file. If you want to modify spec strings, you must first use the `-dumpspecs` option to dump the internal set of spec strings used by your compiler, redirecting the output of this command into a file. You can then modify this file to your heart's content and then load it using the `-specs=`*`file`* option.

To override the spec strings built into the GCC compilers (and after generating the specs file, if necessary), use the `-specs=`*`file`* option to specify an alternative or additional spec file named *file*. The GCC compiler driver reads the specified file after reading the defaults from the standard specs file. If you specify multiple spec files, the compiler driver processes them in order from left to right. Table A-10 lists the possible spec file directives.

Table A-10. *Spec File Directives*

Directive	Description
`%command`	Executes the spec file command specified by *command* (see Table A-11)
`*spec_name`	Creates, deletes, or overrides the spec string specified by *spec_name*
`suffix`	Creates a new suffix rule specified by *suffix*

The next few sections explain these different types of spec file directives.

Spec File Commands

Before you start to panic at the prospect of facing yet another command language to learn, fear not: GCC's spec file command language has only three very simple and easy-to-remember commands: `include`, `include_noerr`, and `rename`. Table A-11 describes what these spec file commands do.

Table A-11. *Spec File Commands*

Command	Description
`%include` *`file`*	Searches for and inserts the contents of *file* at the current point in the spec file
`%include_noerr` *`file`*	Works like `%include`, without generating an error message if *file* is not found
`%rename` *`old new`*	Renames the spec string `old` to `new`

Creating, Deleting, or Redefining Spec Strings

The `*spec_name` directive creates, deletes, or redefines the spec string `spec_name`, whose name is separated from the new definition by a colon. The spec string definition or redefinition continues to the next directive or blank line. To delete a spec string, add a blank line or another directive immediately following `*spec_name`. To append text to an existing spec, use + as the first character of the text defining the spec.

Creating, Deleting, or Modifying Suffix Rules

Spec file suffix rules create, delete, or modify a spec pair, which is comparable to a suffix rule for the make utility. Suffix rules exist primarily to simplify extending the GCC compiler driver program to handle new back-end compilers and new file types. As with spec string definitions following the * directive, the suffix in the suffix rule must be separated from the actual rule directives by a colon, with the spec string for *suffix* being defined by the text following *suffix* up to the next directive or a blank line. When the compiler encounters an input file with the named suffix, it uses the spec string to determine how to compile that file. For example, the following suffix directive creates a spec string that says any input file ending with the characters `.d` should be passed to the program dcc, with an argument of `-j` and the results of the substitution for `%i` (I discuss substitution in the section titled "Modifying and Processing Spec Strings").

```
.d:
dcc -j %i
```

The spec string `@language` following a suffix definition tells GCC that the given suffix definition is really another valid suffix for a predefined language. For example, the following suffix rule tells the GCC compilers that files whose names end in `.d` should be treated exactly as files with the `.c` suffix traditionally used by GCC's C compiler and friends:

```
.d:
@c
```

The spec string *#name* following a suffix specification instructs the GCC compiler to emit an error message that says "name compiler not installed on this system." The actual text of the error message appears to differ depending on the version of GCC you are using, the compiler that you are using, and the platform (operating system) on which you are working. For example, consider the following suffix spec string:

```
.d:
#dcc
```

In this case, if the gcc binary encounters a file whose name ends in `.d`, the GCC compiler driver will emit the error message "dcc compiler not installed on this system."

■ **Tip** If you choose to work through the examples in this section, you might find it easier to see what is happening if you invoke the compiler with `-###`. This option makes it easier to distinguish the output.

Now that you know how to create and modify spec strings, take a look at Table A-12, which lists GCC's built-in spec strings. You can use this table to learn how GCC's designers have configured GCC and, more importantly, to identify spec strings you should **not** idly redefine unless you want to break the GCC compilers in really interesting ways.

Table A-12. *Built-in Spec Strings*

Spec String	Description
`asm`	Specifies the options passed to the assembler
`asm_final`	Specifies the options passed to the assembler postprocessor
`cc1`	Specifies the options passed to the C compiler
`cc1plus`	Specifies the options passed to the C++ compiler
`cpp`	Specifies the options passed to the C preprocessor
`endfile`	Specifies the object files to link at the end of the link phase
`lib`	Specifies the libraries to pass to the linker
`libgcc`	Specifies the GCC support library (libgcc) to pass to the linker
`link`	Specifies the options passed to the linker
`linker`	Specifies the name of the linker
`predefines`	Specifies the `#defines` passed to the C preprocessor
`signed_char`	Specifies the `#defines` passed to the C preprocessor indicating if a `char` is signed by default
`startfile`	Specifies the object files to link at the beginning of the link phase

Modifying and Processing Spec Strings

In addition to extending the GCC compilers to support new suffixes, another common reason for modifying spec file strings is to change how the compiler handles files with new suffixes or to ensure that GCC uses a specific program, such as a certain cross-compiler or part of a cross-compilation chain, to process files with existing suffixes. As such, spec files support a fairly rich set of built-in strings that can be used when modifying or working with spec strings. Table A-13 shows many of GCC's predefined substitution specs.

Table A-13. *Predefined Substitution Specs*

Spec String	Description
`%%`	Substitutes a `%` into the program name or argument.
`%b`	Substitutes the currently processing input file's basename (as the `basename` shell command might generate) without the suffix.
`%B`	Substitutes the currently processing input file's basename, like `%b`, but includes the file suffix.
`%d`	Denotes the argument following `%d` as a temporary filename, which results in the file's automatic deletion upon gcc's normal termination.
`%estr`	Designates `str` as a newline-terminated error message to display.

Table A-13. *Predefined Substitution Specs*

Spec String	Description
`%gsuffix`	Substitutes a filename with a suffix matching `suffix` (chosen once per compilation), marking the argument for automatic deletion (as with `%d`).
`%i`	Substitutes the name of the currently processing input file.
`%jsuffix`	Substitutes the name of the host's null device (such as `/dev/null` on Unix systems), if one exists, if it is writable, and if `-save-temps` has not been specified. On systems that do not have a null device or some other type of bit bucket, `%jsuffix` substitutes the name of a temporary file, which is treated as if specified with the `%usuffix` spec. Such a temporary file should not be used by other spec strings because it is intended as a way to get rid of temporary or intermediate data automatically, not as a means for two compiler processes to communicate.
`%o`	Substitutes the names of all the output files, delimiting each name with spaces.
`%O`	Substitutes the suffix for object files.
`%(name)`	Inserts the contents of spec string `name`.
`%usuffix`	Substitutes a filename with a suffix matching `suffix`, like `%gsuffix`, but generates a new temporary filename even if `%usuffix` has already been specified.
`%v1`	Substitutes GCC's major version number.
`%v2`	Substitutes GCC's minor version number.
`%v3`	Substitutes GCC's patch-level number.
`%w`	Defines the argument following `%w` as the current compilation's output file, which is later used by the `%o` spec (see the entry for `%o`).

Here is a small example of a spec string that changes the linker definition from, say, `ld`, to `my_new_ld`:

```
%rename linker old_linker

*linker:
my_new_%(old_linker)
```

This example renames the spec string named `linker` to `old_linker`. The blank line ends the `%rename` command. The next line, `*linker`, redefines the `linker` spec string, replacing it with `my_new_%(old_linker)`. The syntax `%(old_linker)`, known as a *substitution*, appends the previous definition of `linker` to the new definition, much as one would use the Bourne shell command `PATH=/usr/local/gcc/bin:$PATH` to insert `/usr/local/gcc/bin` at the beginning of the existing directory search path.

Table A-14 lists the available spec processing instructions.

Good examples of files that use and modify spec files can be found in the test suites that accompany the various GCC compilers and in many of the wrapper compilation scripts that accompany alternate C libraries such as uClibc or Newlib.

Table A-14. *Spec Processing Instructions*

Spec String	Description
%1	Processes the cc1 spec, which selects the options to pass to the C compiler (cc1).
%2	Processes the cc1plus spec, which builds the option list to pass to the C++ compiler (cc1plus).
%a	Processes the asm spec, which selects the switches passed to the assembler.
%c	Processes the signed_char spec, which enables cpp (the C preprocessor) to decide if a char is signed or unsigned.
%C	Processes the cpp spec, which builds the argument list to pass to cpp.
%E	Processes the endfile spec, which determines the final libraries to pass to the linker.
%G	Processes the libgcc spec, which determines the correct GCC support library against which to link.
%l	Processes the link spec, which constructs the command line that invokes the linker.
%L	Processes the lib spec, which selects the library names to pass to the linker.
%S	Processes the startfile spec, which determines the startup files (such as the C runtime library) that must be passed to the linker first. For C programs, this is the file crt0.o.

Alphabetical GCC Option Reference

As you would expect from the world's most popular (and freely available) compiler, the compilers in the GNU Compiler Collection have a tremendous number of potential command-line options. This is partially due to the flexibility that arises from having thousands of users and contributors to GCC, each of whom wants the compiler to behave in a specific way, but even more so due to the tremendous number of platforms and architectures on which GCC is used.

GCC's online help in info format is a great source of reference and usage information for GCC's command-line options. However, option information in GCC info is organized into logical groups of options, rather than providing a simple, alphabetical list that you can quickly scan to obtain information about using a specific option. It is also sometimes somewhat out of date—we all know that the documentation is typically the last thing to be updated.

This section provides a single monolithic list of all of the machine-independent GCC command-line options, organized alphabetically for easy reference. GCC also provides hundreds of machine-specific options that you will rarely need to use unless you are using a specific platform and need to take advantage of some of its unique characteristics. Because machine-specific options are both the largest set of GCC options and the set that you will be using least frequently if you tend to work on a more standard hardware platform or operating system, I have grouped these options together in Appendix B.

Only those well-versed in character and string comparisons know offhand whether *A* is less than *a*, and so on, and it is a pain to type **man ascii** each time you need to remember how to sort a specific letter. Since not everyone may be one with the ASCII chart (and may not even be using a Linux, *BSD, or Un*x machine, for that matter), the options described in this section are listed more or less alphabetically, with single-letter uppercase options preceding single-letter lowercase options involving the same letter of the alphabet. Options whose names begin with symbols are listed before

the alphabetic options, followed by options beginning with numerals, and concluding with alphabetic options. The number of dashes preceding any given argument is shown, but is ignored for sorting purposes.

■ **Note** Language-specific options in this section identify the appropriate language whenever possible. This is intended as much for completeness' sake as to help you identify options that are irrelevant for the language that you are compiling.

`-###`: This output option causes the GCC commands relevant to your command line to be displayed in quoted form but not executed. This option is typically used to identify mode-specific commands that you can subsequently incorporate into shell scripts, or to verify exactly what GCC is attempting to execute in response to specific command-line options.

`-A-`: This option cancels all predefined assertions and all assertions that precede it on the command line, and undefines all predefined macros and all macros that precede it on the command line.

`-A-QUESTION=ANSWER`: This option cancels setting the value of `QUESTION` to `ANSWER`. This option is typically used to cancel assertions that had previously been made for use by the preprocessor.

`-AQUESTION=ANSWER`: This option sets the value of `QUESTION` to `ANSWER`. This option is typically used to make assertions for use by the preprocessor. For example, `-A system=gnu` would tell the preprocessor that the value of the system predicate should be asserted as `gnu`.

`-ansi`: This option, for C and C++ programs, enforces compliance with ANSI C (ISO C89) or standard C++, disabling features such as the `asm` and `typeof` keywords; predefined platform-specific macros such as `unix` or `vax`; the use of C++ `//` comments in C programs; and so on. When using this option, non-ISO programs can still be successfully compiled—to actively reject programs that attempt to use non-ANSI features, you must also specify the `-pedantic` option.

`-aux-info filename`: This option, for C programs, causes prototyped declarations for all referenced functions to be dumped to the specified output file filename, including those defined in header files. This option is silently ignored in any language other than C. The output file contains comments that identify the source file and line number for each declared function, and letters indicating whether the declaration was implicit (I), prototyped (N), or unprototyped (O), and whether the function was declared (C) or defined (F) there. Function definitions are followed by a list of arguments and their declarations.

`-B`*`prefix`*: This option, when using any GCC compiler as a cross-compiler, enables you to specify a prefix that should be used to try to find the executables, libraries, and include and data files for the compiler. For each of the subprograms (cpp, cc1, as, and ld) run by the compiler, using this option causes the compiler driver to try to use *`prefix`* to locate each subprogram, both with and without any values specified with the `-b` (`machine`) and `-V` (`version`) options. If binaries with the specified prefix are not found in the directories listed in your `PATH` environment variable, the GCC compiler's driver also looks in the directories `/usr/lib/gcc` and `/usr/local/lib/gcc-lib` for both binaries and subdirectories with relevant names. Using this option also causes the GCC compiler's driver to attempt to locate and use include files and libraries with the specified prefix. This command-line option is equivalent to setting the `GCC_EXEC_PREFIX` environment variable before compilation.

`-b machine`: This option enables you, when using gcc as a cross-compiler, to identify the target machine, and therefore the associated compiler. `machine` should be the same value specified when you executed the configure script to build the cross-compiler.

`-C`: This option causes the preprocessor to retain comments from the input files, with the exception of comments within preprocessor directives, which are deleted along with the directive. This option can be useful if you want to examine the output of the preprocessor (preserved by the `-save-temps` option) and need a convenient way for delimiting regions of code with comments so that you can see exactly what the preprocessor is doing with your input code.

`-CC`: This option does not discard comments when processing the input file, including those contained in macros that are inserted as a result of macro expansion.

`-c`: This output option causes the GCC compiler driver to compile or assemble the source files without linking them, producing separate object files. This option is typically used to minimize recompilation when compiling and debugging multimodule applications.

`-combine`: This option tells the GCC compiler driver to pass all input source code files to the compiler at once, and is only currently supported by the gcc C compiler. If used with the `-save-temps` option, one preprocessed file will be produced for each input file, but only one final assembler output and object code file will be produced.

`-DMACRO[=`*`DEFINITION`*`]`: This preprocessor option sets the value of *`MACRO`* to 1 if no *`DEFINITION`* is specified, or to *`DEFINITION`* if specified.

`-d`*`letters`*: This internal compiler debugging option (rather than an application debugging option) causes the GCC compiler driver to generate debugging output files during compilation at times specified by *letters*. The names of the debugging files are created by appending a pass number (pass) to a word identifying the phase of compilation to the name of the source file (file), separated by a period, and then adding an extension that reflects the phase of compilation at which the file was generated. Values for letters and the names of the output files are as follows:

- `A`: Annotates the assembler output with miscellaneous debugging information
- `a`: Produces the rtl, flow2, addressof, stack, postreload, greg, lreg, life, cfg, and jump debugging output files
- `B`: Dumps after block reordering (file.pass.bbro)
- `b`: Dumps after computing branch probabilities (file.pass.bp)
- `C`: Dumps after the first if-conversion (file.pass.ce)
- `c`: Dumps after instruction combination (file.pass.combine)
- `D`: Dumps all macro definitions at the end of preprocessing
- `d`: Dumps after delayed branch scheduling (file.pass.dbr)
- `E`: Dumps after the second if-conversion (file.pass.ce2)
- `e`: Dumps after static single assignment (SSA) optimizations (file.pass.ssa and file.pass.ussa)
- `F`: Dumps after purging `ADDRESSOF` codes (file.pass.addressof)
- `f`: Dumps after life analysis (file.pass.life)
- `G`: Dumps after global common subexpression elimination (GCSE) (file.pass.gcse)
- `g`: Dumps after global register allocation (file.pass.greg)
- `h`: Dumps after finalization of exception handling (EH) code (file.pass.eh)
- `i`: Dumps after sibling call optimizations (file.pass.sibling)
- `j`: Dumps after the first jump optimization (file.pass.jump)
- `k`: Dumps after conversion from registers to stack (file.pass.stack)
- `L`: Dumps after loop optimization (file.pass.loop)

- `l`: Dumps after local register allocation (file.pass.lreg)
- `M`: Dumps after performing the machine-dependent reorganization (file.pass.mach)
- `m`: Prints statistics on memory usage at the end of the run
- `N`: Dumps after the register move pass (file.pass.regmove)
- `n`: Dumps after register renumbering (file.pass.rnreg)
- `o`: Dumps after post-reload optimizations (file.pass.postreload)
- `P`: Dumps the register transfer language (RTL) in the assembler output as a comment before each instruction
- `p`: Annotates assembler output with a comment identifying the pattern, alternative, and length of each instruction
- `R`: Dumps after the second scheduling pass (file.pass.sched2)
- `r`: Dumps after RTL generation (file.pass.rtl)
- `S`: Dumps after the first scheduling pass (file.pass.sched)
- `s`: Dumps after first common subexpression elimination (CSE) and associated jump optimization pass (file.pass.cse)
- `t`: Dumps after the second CSE and associated jump optimization pass (file.pass.cse2)
- `v`: Dumps a representation of the control flow graph for each dump file. This representation is suitable for viewing with VCG (file.pass.vcg)
- `w`: Dumps after the second flow pass (file.23.flow2)
- `X`: Dumps after SSA dead-code elimination pass (file.pass.ssadce)
- `x`: Only generates RTL for each function instead of compiling it
- `y`: Dumps debugging information during parsing
- `z`: Dumps after the peephole pass (file.pass.peephole2)

`-dumpmachine`: This debugging option displays the compiler's target machine and then exits.

`-dumpspecs`: This debugging option displays the compiler's built-in specs and then exits. These are specifications stored in a file named specs, located in the `lib/gcc-lib` directory of the directory hierarchy in which you installed GCC. GCC's specifications tell GCC where to find various mandatory files and libraries, which tools to use at each phase of compilation, and how to invoke them.

`-dumpversion`: This debugging option displays the compiler's version and then exits.

`-E`: This output option causes the GCC framework to define the macros `__GNUC__`, `__GNUC_MINOR__`, and `__GNUC_PATCHLEVEL__`, and to stop after the preprocessing stage without running the compiler itself. Files that do not require preprocessing are ignored.

`-fabi-version=`*n*: This option specifies the version of the C++ ABI (application binary interface) that the g++ compiler should use. Version 2 is the default and is the version of the ABI introduced with g++ version 3.4. Version 1 is the version of the ABI introduced with g++ version 3.2. Version 0 is the version of the ABI that conforms most closely to the actual C++ ABI specification.

`-falign-functions` | `-falign-functions=`*n*: These optimization options cause GCC to align the start of functions to a machine-specific value (when *n* is not specified) or the next power-of-two boundary greater than *n*, skipping up to *n* bytes. Specifying `-falign-functions=1` is equivalent to specifying `-fno-align-functions`, meaning that functions will not be aligned.

`-falign-jumps` | `-falign-jumps=`*n*: These optimization options cause GCC to align branch targets to a machine-specific value (when *n* is not specified) or to the next power-of-two boundary, skipping up to *n* bytes. Specifying `-falign-jump=1` is equivalent to specifying `-fno-align-jumps`, meaning that jumps will not be aligned.

`-falign-labels` | `-falign-labels=`*n*: These optimization options cause GCC to align all branch targets to a machine-specific value (when *n* is not specified) or to the next power-of-two boundary, skipping up to *n* bytes. Specifying this option causes GCC to insert dummy options in output code to cause the requested alignment. Specifying `-falign-labels=1` is equivalent to specifying `-fno-align-labels`, meaning that labels will not be aligned. If `-falign-loops` or `-falign-jumps` are given and their machine-specific or specified values are greater than the value requested by this option, their values are used instead.

`-falign-loops` | `-falign-loops=`*n*: These optimization options cause GCC to align loop targets to a machine-specific value (when *n* is not specified) or to the next power-of-two boundary, skipping up to *n* bytes. Specifying `-falign-loops=1` is equivalent to specifying `-fno-align-loops`, meaning that loops will not be aligned.

`-fallow-single-precision`: This option, when compiling C applications, specifies not to promote single precision math operations to double precision (the K&R C default), even if `-traditional` is specified. Using this option may provide performance optimizations on certain architectures. This option has no effect when compiling with ISO or GNU C conventions.

`-falt-external-templates`: This deprecated option for C++ compilation generates template instances based on where they are first instantiated. This option is similar to `-fexternal-templates`.

`-fargument-alias` | `-fargument-noalias` | `-fargument-noalias-global`: These code generation options specify the possible relationships among parameters and between parameters and global data. These options rarely need to be used—in most cases, the GCC framework uses the options appropriate to the language that is being compiled. The `-fargument-alias` option specifies that arguments (parameters) can alias each other and global storage. The `-fargument-noalias` option specifies that arguments do not alias each other, but can alias global storage. The `-fargument-noalias-global` option specifies that arguments do not alias each other or global storage.

`-fasynchronous-unwind-tables`: This code generation option causes GCC to generate a loop unwind table in DWARF2 format (if DWARF2 is supported by the target machine) that can be used in stack unwinding by asynchronous external events such as a debugger or a garbage collector.

`-fbounds-check`: This optimization option, in GCC's Java and FORTRAN 77 front ends, automatically generates additional code that checks whether all array indices are within the declared size of the appropriate array. This option is on by default in Java, and false by default for FORTRAN 77. This option is ignored when compiling code in languages other than Java and FORTRAN 77.

`-fbranch-probabilities`: This optimization option uses the file.da files produced by a run of GCC with the `-fprofile-arcs` option to further optimize code based on the number of times each branch is taken. The information in the .da files is closely linked to the GCC options used during a specific run of GCC, so you must use the same source code and the same optimization options for both compilations.

`-fbranch-target-load-optimize`: This optimization option tells the GCC compilers to perform branch target register load optimization before prologue/epilogue threading. This hoists loads out of loops and enables interblock scheduling.

`-fbranch-target-load-optimize2`: This optimization option tells the GCC compilers to perform branch target register load optimization after prologue/epilogue threading.

`-fbtr-bb-exclusive`: This optimization option tells GCC not to reuse branch target registers within any basic block when doing branch target register load optimization.

`-fcall-saved-reg`: This code generation option tells GCC to treat the register named reg as an allocable register whose contents are saved by functions and will therefore persist across function calls. Functions compiled with this option save and restore the register reg if they use it. This option should not be used with registers such as the frame pointer or the stack pointer that are used internally by the compiler, or in which function values are returned. Registers are machine-specific—those valid for each specific GCC output target are defined in the `REGISTER_NAMES` macro in the machine description macro file.

`-fcall-used-reg`: This code generation option tells GCC to treat the register named reg as an allocable register whose contents may be overwritten by function calls. Functions compiled with this option will not save and restore the register reg. This option should not be used with registers such as the frame pointer or the stack pointer that are used internally by the compiler. Registers are machine-specific—those valid for each specific GCC output target are defined in the `REGISTER_NAMES` macro in the machine description macro files.

`-fcaller-saves`: This optimization option tells GCC to automatically add extra instructions that save and restore the contents of each register across function calls. This enables compiled code to make global use of registers that may also be used as scratch pads within various functions. This option is active by default for systems that have no call-preserved registers. This option is automatically enabled when using optimization level 2 and higher.

`-fcheck-new`: This option, when compiling C++ programs, causes GCC to check that the pointer returned by the operator `new` is nonnull before attempting to use the storage that the pointer refers to. This should be unnecessary, since `new` should never return a null pointer. If you declare your operator `new` as `throw()`, G++ will automatically check the return value.

`-fcond-mismatch`: This option, when compiling C programs, allows the successful compilation of conditional expressions with mismatched types in the second and third arguments. The value of such expressions is `void`. This option is not supported for C++.

`-fconserve-space`: This option, when compiling C++ programs, causes GCC to put uninitialized or runtime-initialized global variables into the common segment, as is done when compiling C programs. This saves space in the executable but obscures duplicate definitions and may cause problems if these variables are accidentally destroyed multiple times. This option is no longer useful on most targets because newer releases of GCC typically put variables into the BSS (the Block Started by Symbol section of an object file, which I always think of as the *Bullshit Section*) without making them common.

`-fconstant-string-class=`*`classname`*: This option, when compiling Objective C programs, tells GCC to use *`classname`* as the name of the class to instantiate for each literal string specified with the syntax `@"..."`. The default *`classname`* is `NXConstantString`.

`-fcprop-registers`: This optimization option tells GCC to perform a copy propagation pass after post-allocation instruction splitting and register allocation to try to reduce scheduling dependencies and eliminate extraneous copies. This option is disabled by default at optimization levels 2 and 3, and when optimizing for size.

`-fcross-jumping`: This optimization option tells GCC to perform a cross-jumping transformation that tries to unify equivalent code and thereby reduce code size. This option is enabled by default at optimization levels 2 and 3, and when optimizing for size.

`-fcse-follow-jumps`: This optimization option, used during CSE, tells GCC to scan through jump instructions when the target of the jump is not reached by any other path. For example, when CSE encounters an `if` statement with an `else` clause, CSE will follow the jump when the condition tested is false.

`-fcse-skip-blocks`: This optimization option, used during common subexpression elimination, is similar to `-fcse-follow-jumps`, but causes CSE to follow jumps that conditionally skip over blocks. When CSE encounters a simple `if` statement with no `else` clause, this option causes CSE to follow the jump around the body of the `if` statement.

`-fcx-limited-range`: This optimization option specifies that a range reduction step should be used when performing complex division. Though this option sets the C99 `CX_LIMITED_RANGE` pragma, it can be used with all GCC compilers.

`-fdata-sections`: This optimization option, for output targets that support arbitrary code sections, causes GCC to place each data item into its own section in the output file. The name of the data item determines the name of the section in the output file. This option is typically used on systems with HP/PA or SPARC processors (under HPUX or Solaris 2, respectively), whose linkers can perform optimizations that improve the locality of reference in the instruction space. These optimizations may also be available on AIX and systems using the ELF object format. Using this option causes the assembler and linker to create larger object and executable files that may therefore be slower on some systems. Using this option also prevents the use of gprof on some systems and may cause problems if you are also compiling with the `-g` option.

`-fdelayed-branch`: This optimization option, if supported by the target machine, causes GCC to attempt to reorder instructions in order to exploit instruction slots that are available after delayed branch instructions.

`-fdelete-null-pointer-checks`: This optimization option tells GCC to use global dataflow analysis to identify and eliminate useless checks for null pointers. The compiler assumes that dereferencing a null pointer would have halted the program, so that pointers that are checked after being dereferenced cannot be null. On systems that can safely dereference null pointers, you can use `-fno-delete-null-pointer-checks` to disable this optimization.

`-fdiagnostics-show-location=[once|every-line]`: This option tells GCC's diagnostic message formatter how frequently diagnostic messages should display source information when diagnostic messages are wrapped across multiple lines. The default value is `once`; specifying *`every-line`* causes the diagnostic formatter to specify the source location on every output line, even if it is just a continuation of a previous message.

`-fdiagnostics-show-options`: This debugging option tells the GCC diagnostics engine to identify the command-line option associated with each diagnostic message that is displayed.

`-fdollars-in-identifiers`: This C++ compilation option tells GCC to accept the dollar sign symbol ($) in identifiers. You can also explicitly prohibit use of dollar signs in identifiers by using the `-fno-dollars-in-identifiers` option. This is a backward-compatibility option; K&R C allows the character $ in identifiers, but ISO C and C++ forbid the use of this character.

`-fdump-class-hierarchy` | `-fdump-class-hierarchy-`*`options`*: These debugging options, when compiling C++ programs, cause GCC to dump a representation of each class's hierarchy and virtual function table layout to a file. The filename is constructed by appending `.class` to the name of each source file. If the *`options`* form is used, the values specified in *`options`* control the details of the dump as described in the definition for the `-fdump-tree` option.

`-fdump-ipa-`*`switch`*: This debugging option controls dumping the language tree to a file at various points of interprocedural analysis, as identified by *switch*. When *switch* is `cgraph`, information about call-graph optimization, unused function removal, and inlining is dumped. Setting *switch* to `all` enables all interprocedural language dumps.

`-fdump-translation-unit` | `-fdump-translation-unit-`*`options`*: These debugging options, when compiling C++ programs, cause GCC to dump a representation of the tree structure for the entire translation unit to a file. The filename is constructed by appending `.tu` to the name of each source file. If the *`options`* form is used, the values specified in *`options`* control the details of the dump as described in the description of the `-fdump-tree` options.

`-fdump-tree-`*`switch`* | `-fdump-tree-`*`switch-options`*: These debugging options, when compiling C++ programs, cause GCC to dump the intermediate language tree to a file at various stages of processing. The filename is constructed by appending a *switch*-specific suffix to the source filename, preceded by a `.t`*`XX`*`.` prefix, where *XX* identifies the numeric sequence of the pass made when processing the language tree. The following tree dumps are possible:

- `alias`: Dumps aliasing information for each function.
- `all`: Enables all the available tree dumps with the flags provided in this option. With no flags, this option enables the `cfg`, `generic`, `gimple`, `original`, and `vcg` dumps.
- `ccp`: Dumps each function after CCP (conditional constant propagation).
- `cfg`: Dumps the control flow graph of each function to a file.
- `ch`: Dumps each function after copying loop headers.
- `copyprop`: Dumps trees after copy propagation.
- `copyrename`: Dumps each function after applying the copy rename optimization.
- `dce`: Dumps each function after dead-code elimination.
- `dom`: Dumps each function after applying dominator tree optimizations.
- `dse`: Dumps each function after applying dead-store elimination.
- `forwprop`: Dumps each function after forward propagating single-use variables.
- `fre`: Dumps trees after full redundancy elimination.
- `gimple`: Dumps each function to a file before and after the gimplification pass.
- `inlined`: Dumps after function inlining.
- `mudflap`: Dumps each function after adding mudflap instrumentation.
- `nrv`: Dumps each function after applying the named return value optimization on generic trees.
- `optimized`: Dumps after all tree-based optimizations.
- `original`: Dumps before any tree-based optimization.
- `phiopt`: Dumps each function after optimizing the `phi` operator nodes into straightline code. The `phi` operator is a special operator used to assign different values based on how a block is reached.
- `pre`: Dumps trees after partial redundancy elimination.
- `salias`: Dumps structure aliasing variable information to a file.
- `sink`: Dumps each function after performing code sinking.
- `sra`: Dumps each function after performing scalar replacement of aggregates.

- `ssa`: Dumps SSA-related information to a file.
- `store_copyprop`: Dumps trees after store copy propagation.
- `storeccp`: Dumps each function after store conditional constant propagation.
- `vcg`: Dumps the control flow graph of each function to a file in VCG format.
- `vect`: Dumps each function after applying vectorization of loops.
- `vrp`: Dumps each function after value range propagation.

If the *`options`* form is used, *`options`* is a list of hyphen-separated options that control the details of the dump. Any options that are irrelevant to a specific dump are ignored. The following options are available:

- `address`: Specifying this dump-tree option prints the address of each node and is primarily used to associate a specific dump-tree file with a specific debugging environment.
- `all`: Specifying this dump-tree option turns on all options except `raw`, `slim`, and `lineno`.
- `blocks`: Specifying this dump-tree option prints basic block boundaries and is disabled in raw dumps.
- `details`: Specifying this dump-tree option increases the level of information provided in the dump files where possible.
- `lineno`: Specifying this dump-tree option displays line numbers for statements.
- `raw`: Specifying this dump-tree option prints a raw representation of the dump tree. By default, trees are pretty printed with a C-like representation.
- `slim`: Specifying this dump option prevents dumping members of a scope or function body simply because the end of that scope has been reached and only dumps such items when they are directly reachable by some other path. When dumping pretty-printed trees, this option prevents dumping the bodies of control structures.
- `stats`: Specifying this option dumps various statistics about each available pass.
- `uid`: Specifying this dump option displays the unique ID (DECL_UID) for each variable.
- `vops`: Specifying this option displays the virtual operands for every statement.

`-fdump-unnumbered`: This option, when doing debugging dumps due to the use of the `-d` option, causes GCC to suppress instruction and line numbers. This makes it easier to use `diff` to compare debugging dumps from multiple runs of GCC with different compiler options, most specifically with and without the `-g` option.

`-fearly-inlining`: This debugging option tells GCC to inline specific types of functions early—functions that are marked by `always_inline` and functions whose body seems smaller than the overhead required for a function call. This is done before profiling the code and before the actual inlining pass, and simplifies profiling. This option is enabled by default when doing optimization.

`-feliminate-dwarf2-dups`: This debugging option causes GCC to compress DWARF2 debugging information by eliminating any duplicate information about each symbol but is only meaningful when generating DWARF2 debugging information.

`-feliminate-unused-debug-symbols`: This debugging option causes GCC to only generate debugging information (in STABS format) for symbols that are actually used in the source file(s) that are being compiled.

`-feliminate-unused-debug-types`: This debugging option causes GCC to only generate debugging information for types that are actually used in the source file(s) that are being compiled.

`-femit-class-debug-always`: This debugging option causes GCC to generate debugging information for a C++ class in all object files that use that class. The default is to only generate this information in the object file where the class is first encountered. This option is provided to support debuggers that require that this information be present in each object file.

`-fexceptions`: This code generation option tells GCC to enable exception handling and generates any extra code needed to propagate exceptions. This option is on by default when compiling applications written in languages such as C++ that require exception handling. It is primarily provided for use when compiling code written in languages that do not natively use exception handling, but that must interoperate with C++ exception handlers. For some targets, using this option also causes GCC to generate frame unwind information for all functions, which can substantially increase application size although it does not affect execution.

■ **Note** As an optimization, you can use the `-fno-exceptions` option to disable C++ support for exception handling in older C++ applications that do not use exception handling.

`-fexpensive-optimizations`: This optimization option tells GCC to perform a number of minor optimizations that may require a substantial amount of processing time.

`-fexternal-templates`: This option, when compiling C++ applications, causes GCC to apply `#pragma` interface and `#pragma` implementation to template instantiation. Template instances are emitted or suppressed according to the location of the template definition. The use of this option is deprecated.

`-ffast-math`: This optimization option causes GCC to define the preprocessor macro `__FAST_MATH__` and perform internal math optimizations. This option implies the use of the `-fno-math-errno`, `-funsafe-math-optimizations`, and `-fno-trapping-math` options and activates them if they are not specified. This option should never be used in conjunction with GCC's standard `-O` optimization options, because this can result in incorrect output for programs that depend on the exact implementation of IEEE or ISO rules and specifications for math functions.

`-ffinite-math-only`: Specifying this optimization option causes GCC to allow floating-point optimizations to make the assumption that no checks need to be done to verify that arguments and results are NaN (Not-a-Number) or have infinite values.

`-ffixed-`*reg*: This code generation option tells GCC to treat the register identified by *reg* as a fixed register that is never referred to by generated code other than compiler internals. Registers are machine-specific—those valid for each specific GCC output target are defined in the `REGISTER_NAMES` macro in the machine description macro file.

`-ffloat-store`: This optimization option tells GCC not to store floating-point variables in registers and to inhibit other options that might change whether a floating-point value is taken from a register or memory. Using this option prevents excess precision on machines where floating registers keep more precision than a double floating-point value requires, such as the 68000 (with 68881) and x86 architectures. Additional precision may be undesirable in programs that rely on the precise definition of IEEE floating point. You can compile such programs with this option only if you modify them to store relevant intermediate computations in variables rather than in registers.

`-ffor-scope`: When compiling C++ programs, specifying this option tells GCC to limit the scope of variables declared in a `for-init` statement to the `for` loop, as specified by the C++ standard. If the opposite `-fno-for-scope` option is used, the scope of variables declared in a `for-init` statement extends to the end of the enclosing scope. This was the default behavior of older versions of GNU C++ and many traditional implementations of C++. If neither option is specified, GCC follows the C++ standard.

`-fforce-addr`: This optimization option causes GCC to force memory address constants to be copied into registers before doing arithmetic on them.

`-fforce-mem`: This optimization option causes GCC to force memory operands to be copied into registers before doing arithmetic on them. This may produce better code because it makes all memory references potential common subexpressions that can be reduced. When they are not common subexpressions, instruction combination should eliminate the additional overhead of separate register load. This option is automatically turned on when using the `-O2` optimization option.

`-ffreestanding`: This option, when compiling C applications, tells GCC that compilation takes place in a freestanding environment where the standard library may not be available or exist, such as when compiling an operating system kernel. Using this option implies the use of the `-fno-builtin` option, and is equivalent to using the `-fno-hosted` option.

`-ffriend-injection`: This C++ option tells GCC to inject friend functions into the enclosing namespace such that they are visible outside the scope of the class in which they were declared. This follows the conventions documented in the Annotated C++ Reference Manual and used by versions of GCC prior to 4.1, and is provided for compatibility.

`-ffunction-sections`: This optimization option, for output targets that support arbitrary code sections, causes GCC to place each function into its own section in the output file. The name of the function determines the name of the section in the output file. This option is typically used on systems with HP/PA or SPARC processors (under HPUX or Solaris 2, respectively), whose linkers can perform optimizations that improve the locality of reference in the instruction space. These optimizations may also be available on AIX and systems using the ELF object format. Using this option causes the assembler and linker to create larger object and executable files that may therefore be slower on some systems. Using this option also prevents the use of gprof on some systems and may cause problems if you are also compiling with the `-g` option.

`-fgcse`: This optimization option tells GCC to perform a global common subexpression elimination pass, also performing global constant and copy propagation.

■ **Note** When compiling a program using GCC's computed `gotos` extension, you may get better runtime performance if you disable the global common subexpression elimination pass by specifying the `-fno-gcse` option.

`-fgcse-after-reload`: This optimization option causes GCC to perform an additional load elimination pass after reload in order to clean up redundant spilling.

`-fgcse-las`: This optimization option tells GCC to eliminate redundant loads that come after stores to the same memory location during the global common subexpression elimination pass.

`-fgcse-lm`: This optimization option causes GCC to attempt to optimize load operations during global command subexpression elimination. If a load within a loop is subsequently overwritten by a store operation, GCC will attempt to move the load outside the loop and to only use faster copy/store operations within the loop.

`-fgcse-sm`: This optimization causes GCC to attempt to optimize store operations after global common subexpression elimination. When used in conjunction with the `-fgcse-lm` option, loops containing a load/store sequence will be changed to a load before the loop, and a store after the loop whenever possible.

`-fgnu-runtime`: This option, when compiling Objective C programs, causes GCC to generate object code that is compatible with the standard GNU Objective C runtime. This is the default on most systems.

`-fhosted`: This option, when compiling C programs, tells GCC that the entire standard C library is available during compilation, which is known as a *hosted environment*. Specifying this option implies the `-fbuiltin` option. This option is usually appropriate when compiling any application other than a kernel, unless you want to compile statically linked applications. Using this option is equivalent to using the `-fno-freestanding` option.

`-fif-conversion`: This optimization option tells GCC to use conditional execution to convert conditional jumps into nonbranching equivalents, wherever possible.

`-fif-conversion2`: This optimization option tells GCC to attempt to convert conditional jumps into nonbranching equivalents.

`-finhibit-size-directive`: This code generation option tells GCC not to output a `.size` assembler directive or any other internal information that could cause trouble if the function is split and the two portions subsequently relocated to different locations in memory. This option is typically used only when compiling the `crtstuff.c` character handling routine and should not be necessary in any other case.

`-finline-functions`: This optimization option tells GCC to integrate simple functions into the routines that call them if they are simple enough to do so, based on heuristics. If all calls to a given function are integrated and the function is declared static, no independent assembler code for the function is generated independent of that in the routines that call it.

`-finline-functions-called-once`: This optimization option tells GCC to evaluate all static functions that are only called once for integration into the routine from which they are called, even if they are not marked as inline. When such functions are integrated into the calling routines, no stand-alone assembler code for the function is generated.

`-finline-limit=`*n*: This optimization option tells GCC to modify its internal limit on the size of functions that are explicitly marked as inline with the `inline` keyword or that are defined within the class definition in C++. *N* represents the number of pseudoinstructions (an internal GCC calculation) in such functions, excluding instructions related to parameter handling. The default value of *n* is 600. Increasing this value can result in more inlined code, which may cause increased compilation time and memory consumption. Decreasing this value can improve compilation faster, but may result in slower programs because less code will be inlined. This option can be quite useful for programs such as C++ programs that use recursive templates and therefore can substantially benefit from inlining.

`-finstrument-functions`: This code generation option tells GCC to insert instrumentation calls for function entry and exit. The following profiling functions will be called with the address of the current function and the address from which it was called immediately after entering and immediately before exiting from each function:

```
void __cyg_profile_func_enter (void *this_fn, void *call_site);
void __cyg_profile_func_exit (void *this_fn, void *call_site);
```

The first argument is the address of the start of the current function, which can be looked up in the symbol table. You can specify the `no_instrument_function` attribute for specific functions in order to avoid making profiling calls from them. For more information about profiling and for examples of using profiling functions and attributes, see Chapter 6.

`-fipa-pta`: This optimization option tells GCC to perform interprocedural pointer analysis.

`-fivopts`: This optimization option tells GCC to perform induction variable optimizations (strength reduction, induction variable merging, and induction variable elimination) on trees.

`-fkeep-inline-functions`: This optimization option tells GCC to generate a separate, callable version of each function marked as `inline`, even if all calls to a given function have resulted in inline code. This does not apply to functions marked as `extern inline`.

`-fkeep-static-consts`: This optimization option causes GCC to create and allocate all variables declared `static const`, even if the variables are not referenced. This option is active by default. To force GCC to check whether variables are referenced and to only create them if this is the case, use the `-fno-keep-static-consts` option.

`-fleading-underscore`: This code generation option is provided for compatibility with legacy assembly code, and forces C symbols to be created with leading underscores in object files. This is not completely supported on all GCC output targets.

`-fmath-errno`: This optimization option tells GCC to set the value of `ERRNO` after calling math functions that are executed with a single instruction, such as the `sqrt` function. Programs that rely on IEEE exceptions for math error handling may want to use the `-fno-math-errno` option to provide performance improvements while still maintaining IEEE arithmetic compatibility.

`-fmem-report`: This debugging option causes GCC to display statistics about permanent memory allocation at the end of compilation.

`-fmerge-all-constants`: This optimization option causes GCC to attempt to merge identical constants and variables. This increases the scope of the `-fmerge-constants` option and also tries to merge arrays and variables that are initialized with string and floating-point constants. This may generate nonconformant C and C++ code because these languages require all nonautomatic variables to have distinct locations.

`-fmerge-constants`: This optimization option causes GCC to attempt to merge identical string and floating-point constants across compilation units. This option is on by default during optimization on output targets where the assembler and linker support it.

`-fmessage-length=`*n*: This diagnostic option causes GCC to format diagnostic messages so that they fit on lines of *n* characters or less, inducing line wrapping in messages that are greater than *n* characters. The default value for *n* is 72 characters for C++ messages and 0 for all other front ends supported by GCC. If *n* is 0, no line wrapping will be done.

`-fmodulo-sched`: This optimization option tells GCC to perform swing modulo scheduling, where the instructions in innermost loops can be reordered to improve scheduling performance before the first elimination pass.

`-fmove-loop-invariants`: This optimization option causes GCC to move all invariant computations in loops outside the loop. This option was formerly known as `-fmove-all-movables`.

`-fms-extensions`: This option when compiling C programs tells GCC to accept some nonstandard constructs used in Microsoft Foundation Class (MFC) header files. Specifying this option when compiling C++ programs tells GCC to disable pedantic warnings about MFC constructs such as implicit integers and nonstandard syntax for getting pointers to member functions.

`-fmudflap`: This optimization option when compiling C and C++ applications tells GCC to instrument pointer/array dereferencing operations, standard library string/heap functions, and associated constructs with range and validity tests. The instrumented functions and constructs are assumed to be valid. This instrumentation relies on the libmudflap library, which was first introduced in GCC 4.0, and which is linked with the application at link time if this option is specified.

`-fmudflapir`: This optimization option when compiling C and C++ applications tells GCC to perform the same instrumentation as the `-fmudflap` option, but uses a different version of the libmudflap library in which instrumentation ignores pointer reads. This executes more quickly but can enable erroneous reads to propagate within a program.

`-fmudflapth`: This optimization option when compiling multithreaded C and C++ applications tells GCC to perform the same instrumentation as the `-fmudflap` option but uses a different multithreaded version of the libmudflap library.

`-fnext-runtime`: This option, when compiling Objective C programs, causes GCC to generate output that is compatible with the former NeXT computer system runtime that is used on Darwin and Mac OS X systems.

`-fno-access-control`: This option when compiling C++ programs disables access checking. This option is rarely used and is primarily provided to work around problems in the access control code.

`-fno-asm`: This option affects the keywords that are recognized when compiling C and C++ programs. When compiling ISO C99 programs, this option disables the `asm` and `typeof` keywords. When compiling other C programs, this option also disables the `inline` keyword. When compiling C++ programs, this option only disables the `typeof` keyword. You can still use the keywords `__asm__`, `__inline__`, and `__typeof__` in any C or C++ application. This option is automatically enabled when using the `-ansi` option.

`-fno-branch-count-reg`: This optimization option causes GCC to not use decrement and branch instructions on a count register. Instead, GCC generates a sequence of instructions that decrement a register, compare it against zero, and then branch, based upon the result. This option is only meaningful on architectures such as x86, PowerPC, IA-64, and S/390 that provide decrement and branch instructions.

`-fno-builtin` | `-fno-builtin-`*`FUNCTION`*: These C and Objective C options cause GCC not to recognize built-in functions that do not begin with `__builtin_` as a prefix, which guarantees that you will be able to set breakpoints on function calls and replace the functionality of `FUNCTION` by linking with a different library. This rule is always the case in C++ applications compiled with GCC; to specifically invoke GCC's internal functions, you must call them with the `__builtin_` prefix.

> GCC normally generates special code to handle its built-in functions more efficiently. In some cases, built-in functions may be replaced with inline code that makes it difficult to set breakpoints during debugging or to link with external libraries that provide equivalent functions.

In C and Objective C applications, you can use the less restrictive `-fno-builtin-FUNCTION` option to selectively disable specific built-in functions. This option is ignored if no such built-in function is present. Similarly, you can use the `-fno-builtin` option to disable all built-in functions and then modify your applications to selectively map function calls to the appropriate built-ins, as in the following example:

```
#define strcpy(d, s)    __builtin_strcpy ((d), (s))
```

`-fno-common`: This code generation option when compiling a C language application causes GCC to allocate all global variables in the data section of the object file, even if uninitialized, rather than generating them as common blocks. This option is provided for compatibility with existing systems but has the side effect that if the same variable is declared in multiple separately compiled source modules and is not explicitly declared as `extern`, GCC will display an error message during linking.

`-fno-const-strings`: This C++ option causes GCC to assign the type `char *` to string constants rather than `const char *` as specified in the C++ standard. Specifying this option does not allow you to write to string constants unless you also use the `-fwritable-strings` option. Using this option is deprecated in the GCC 3.x compilers and has been removed in GCC 4.x.

`-fno-cprop-registers`: This optimization option causes GCC to perform an additional copy propagation pass to try to improve scheduling and, where possible, eliminate the register copy entirely. This is done after register allocation and post–register allocation instruction splitting.

`-fno-default-inline`: This C++ optimization option causes GCC not to assume that functions declared within a class scope should be inlined. If you do not specify this option, GCC will automatically inline the functions if you use any optimization level (`-O`, `-O2`, or `-O3`) when compiling C++ applications.

`-fno-defer-pop`: This optimization option causes GCC to pop the arguments to each function call as soon as that function returns. GCC normally lets arguments accumulate on the stack for several function calls and pops them all at once on systems where a pop is necessary after a function call return.

`-fno-elide-constructors`: This option when compiling C++ options causes GCC not to omit creating temporary objects when initializing objects of the same type, as permitted by the C++ standard. Specifying this option causes GCC to explicitly call the copy constructor in all cases.

`-fno-enforce-eh-specs`: This option when compiling C++ applications causes GCC to skip checking for violations of exception specifications at runtime. This option violates the C++ standard but may be useful for reducing code size. The compiler will still optimize based on the exception specifications.

`-fno-for-scope`: This option when compiling C++ applications causes GCC to extend the scope of variables declared inside a `for` loop to the end of the enclosing scope. This is contrary to the C++ standard but was the default in older versions of GCC and most other traditional C++ compilers. If neither this option nor the opposite `-ffor-scope` option is specified, GCC adheres to the standard but displays a warning message for code that might be invalid or assumes the older behavior.

`-fno-function-cse`: This optimization option causes GCC not to put function addresses in registers but to instead make each instruction that calls a constant function contain the explicit address of that function. You may need to use this option if you receive assembly errors when using other optimization options.

`-fno-gnu-keywords`: This option affects the keywords that are recognized when compiling C and C++ programs. When compiling ISO C99 programs, this option disables the `asm` and `typeof` keywords. When compiling other C programs, this option also disables the `inline` keyword. When compiling C++ programs, this option only disables the `typeof` keyword. You can still use the keywords `__asm__`, `__inline__`, and `__typeof__` in any C or C++ application. This option is automatically enabled when using the `-ansi` option.

`-fno-gnu-linker`: This code generation option is used when you are compiling code for linking with non-GNU linkers and suppresses generating global initializations (such as C++ constructors and destructors) in the form used by the GNU linker. Using a non-GNU linker also requires that you use the collect2 program during compilation to make sure that the system linker includes constructors and destructors, which should be done automatically when configuring the GCC distribution.

`-fno-guess-branch-probability`: This optimization option tells GCC to use an empirical model to predict branch probabilities, which may prevent some optimizations. Normally, GCC may use a randomized model to guess branch probabilities when none are available from either profiling feedback (`-fprofile-arcs`) or `__builtin_expect`. This may cause different runs of the compiler on the same program to produce different object code, which may not be desirable for applications such as real-time systems.

`-fno-ident`: This code generation option causes GCC to ignore the `#ident` directive.

`-fno-implement-inlines`: This option when compiling C++ programs tells GCC not to generate out-of-line copies of inline functions based on the `#pragma` implementation. This saves space but will cause linker errors if the code for these functions is not generated inline everywhere these functions are called.

`-fno-implicit-inline-templates`: This option when compiling C++ programs tells GCC not to generate code for implicit instantiations of inline templates. This option is typically used in combination with optimization options to minimize duplicated code.

`-fno-implicit-templates`: This option when compiling C++ programs tells GCC to only emit code for explicit instantiations and not to generate code for noninline templates that are instantiated by use.

`-fno-inline`: This optimization option prevents GCC from expanding any function inline, effectively ignoring the `inline` keyword. During normal optimization, the code for functions identified as inline is automatically inserted at each function call.

`-fno-jump-tables`: This code generation option tells GCC not to use jump tables for handling switch statements and is typically used when generating code for use in a dynamic linker that therefore cannot reference the address of a jump or global offset table.

`-fno-math-errno`: This optimization option prevents GCC from setting `ERRNO` after calling math functions that can be executed as a single instruction, such as `sqrt`. Not setting `ERRNO` reduces the overhead of calling such functions, but `ERRNO` should be set in programs that depend on exact IEEE or ISO compliance for math functions. The `-fno-math-errno` option is therefore not turned on by any of the generic GCC optimization options and must be specified manually.

`-fno-nil-receivers`: This Objective C and Objective C++ language option tells GCC to assume that the receiver of any message dispatched in the current translation unit is not nil. This option is only supported in the NeXT runtime used on Mac OS X 10.3 and greater.

`-fno-nonansi-builtins`: This option when compiling C++ programs disables the use of built-in GCC functions that are not specifically part of the ANSI/ISO C specifications. Such functions include `exit()`, `alloca()`, `bzero()`, `conjf()`, `ffs()`, `index()`, and so on.

`-fno-operator-names`: This option when compiling C++ programs prevents GCC from recognizing keywords such as `and`, `bitand`, `bitor`, `compl`, `not`, `or`, and `xor` as synonyms for the C++ operators that represent those operations.

`-fno-optional-diags`: This option when compiling C++ programs disables optional diagnostics that are not mandated by the C++ standard. In version 3.2.2 of GCC, the only such diagnostic is one generated when a name has multiple meanings within a class. Other such diagnostics may be added in future releases of GCC.

`-fno-peephole` | `-fno-peephole2`: These optimization options disable different types of machine-specific optimizations. Whether one or both of these options are useful on your system depends on how or if they are implemented in the GCC implementation for your system.

`-fno-rtti`: This option, when compiling C++ programs, tells GCC not to generate information about every class with virtual functions. This information is typically used by C++ runtime type identification features such as `dynamic_cast` and `typeid`. Using this option can save space if you do not explicitly use those aspects of C++—they will still be used by internals such as exception handling, but the appropriate information will only be generated when needed.

`-fno-sched-interblock`: This optimization option tells GCC not to schedule instructions across basic blocks, which is normally done by default when using the `-fschedule-insns` option or optimization options `-O2` or higher.

`-fno-sched-spec`: This optimization option tells GCC not to move nonload instructions, which is normally done by default when using the `-fschedule-insns` option or optimization options `-O2` or higher.

`-fno-signed-bitfields`: This option, when compiling C programs, tells GCC that bit fields are unsigned by default (in other words, when neither `signed` or `unsigned` is present in their declaration). Without specifying this option, bit fields are assumed to be signed in order to be consistent with other basic datatypes. This option is redundant when used in conjunction with the `-traditional` option, which causes all bit fields to be unsigned by default.

`-fno-stack-limit`: This code generation option causes GCC to generate code without an explicit limit on stack size.

`-fno-threadsafe-statics`: This C++ language option tells GCC not to generate the extra code required to use the C++ ABI's thread-safe initialization routines for local statics. This can reduce code size when the generated code does not have to be thread-safe.

`-fno-toplevel-reorder`: This optimization option tells GCC not to reorder any top-level functions, `asm` statements, or variable declarations, and is designed to support existing code that depends on its current ordering.

`-fno-trapping-math`: This optimization option causes GCC to generate code that assumes that floating-point operations cannot generate user-visible traps, which may result in faster operation due to reduced overhead when returning from such functions. If you want to experiment with or perform this type of optimization, this option must be explicitly specified because its use may result in code that is not completely IEEE or ISO compliant. This option is never automatically turned on by any standard GCC optimization option.

`-fno-unsigned-bitfields`: This option, when compiling C programs, tells GCC that bit fields are signed by default (in other words, when neither the `signed` or `unsigned` keyword is present in their declaration). This option should not be used in conjunction with the `-traditional` option, which causes all bit fields to be unsigned by default.

`-fno-verbose-asm`: This code generation option minimizes the number of comments inserted into generated assembly code, and is the GCC default.

`-fno-weak`: This option when compiling C++ programs tells GCC not to use weak symbol support, even if it is provided by the linker—GCC's default action is to use weak symbols when they are available. This option is primarily used for testing at this point and generally should not be used.

`-fno-zero-initialized-in-bss`: This optimization option tells GCC to put variables that are initialized to zero in the data section rather than the BSS. Using the BSS for such variables is GCC's default behavior because this can reduce the size of generated code.

`-fnon-call-exceptions`: This option when compiling languages such as C++ that support exceptions and when runtime support for exception handling is present causes GCC to generate code that allows trap instructions to throw exceptions. This option does not enable exceptions to be thrown from arbitrary signal handlers.

`-fobjc-call-cxx-cdtors`: This Objective C and Objective C++ language option tells GCC to check if the instance variables for any Objective C class are C++ objects with a nontrivial constructor or destructor. If so, GCC synthesizes special instance methods that run nontrivial constructors in the right order (returning self) or nontrivial destructors in the opposite order.

`-fobjc-direct-dispatch`: This Objective C and Objective C++ language option tells GCC to enable fast jumps to the message dispatcher.

`-fobjc-exceptions`: This Objective C and Objective C++ language option tells GCC to enable syntactic support for structure exceptions handling, similar to that provided by C++ and Java.

`-fobjc-gc`: This Objective C and Objective C++ language option enables garbage collection in Objective C and Objective C++ programs.

`-fomit-frame-pointer`: This optimization option tells GCC not to keep the frame pointer in a register for functions that do not require a frame pointer. Besides making an additional register available for other code, this option reduces both code size and the execution path by eliminating the instructions required to save, set up, and restore frame pointers. This option will have no effect on systems where the standard function calling sequence always includes frame pointer allocation and setup. When building GCC for a specific platform, the `FRAME_POINTER_REQUIRED` macro determines whether this option is meaningful on a specific target system.

`-fopenmp`: This code generation option tells GCC to support handling the OpenMP directives `#pragma omp` in C/C++ applications, and `!$omp` in Fortran applications. See the OpenMP API Specification at `http://www.openmp.org` for more information about OpenMP and associated directives.

`-foptimize-register-move`: This optimization option tells GCC to reassign register numbers in simple operations in an attempt to maximize register tying. This option is synonymous with the `-fregmove` option and is automatically enabled when using GCC optimization levels 2 and higher.

`-foptimize-sibling-calls`: This optimization option attempts to optimize sibling and tail recursive calls.

`-fpack-struct`: This code generation option tells GCC to attempt to pack all structure members together without holes, reducing memory use in applications that allocate significant numbers of in-memory data structures. This option is rarely used in applications that make extensive use of system functions that may employ offsets based on the default offsets of fields in data structures. Using this option produces code that is not binary-compatible with code generated without this option.

`-fpcc-struct-return`: This code generation option causes GCC to return short (integer-sized) `struct` and `union` values in memory rather than in registers. This option is typically used when linking object code compiled with GCC with object code produced by other compilers that use this convention.

`-fpeel-loops`: This optimization option tells GCC to simplify loops for which enough information is available to extract the first (or first few) problematic iterations of the loop so that the rest of the loop can be simplified.

`-fpermissive`: This option, when compiling C++ code, causes GCC to downgrade the severity of messages about nonconformant code to warnings, rather than treating them as actual errors. This option has no effect if used with the `-pedantic` option.

`-fPIC`: This code generation option tells GCC to emit position-independent code (PIC) that is suitable for dynamic linking but eliminates limitations on the size of the global offset table. PIC uses a global offset table to hold constant addresses that are resolved when an application is executed. This option is therefore only meaningful on platforms that support dynamic linking and is used when generating code for the 680x0, PowerPC, and SPARC processors. If you are interested in potentially reducing the size of the global offset table, you should use the `-fpic` option instead of this one.

`-fpic`: This code generation option tells GCC to generate position-independent code that is suitable for use in a shared library. This option is only meaningful on target platforms that support dynamic linking and use a machine-specific value (typically 16K or 32K) for the maximum size of the global allocation table for an application. If you see an error message indicating that this option does not work, you should use the `-fPIC` option instead of this one.

■ **Note** Code generated for the IBM RS/6000 is always position independent.

`-fPIE`: This code generation option produces code that is the same as that produced when specifying `-fPIC` but which can only be linked into executables. See the `-pie` option for more information.

`-fpie`: Specifying this code generation option produces code that is the same as that produced when specifying `-fpic`, but which can only be linked into executables. See the `-pie` option for more information.

`-fprefetch-loop-arrays`: This optimization option tells GCC to generate instructions to prefetch memory on platforms that support this. Prefetching memory improves the performance of loops that access large arrays.

`-fpretend-float`: This debugging option is often used when cross-compiling applications and tells GCC to compile code, assuming that both the host and target systems use the same floating-point format. Though this option can cause actual floating constants to be displayed correctly, the instruction sequence will probably still be the same as the one GCC would make when actually running on the target machine.

`-fprofile-arcs`: This debugging and optimization option tells GCC to instrument arcs (potential code paths from one function or procedure to another) during compilation to generate coverage data. This coverage information can subsequently be used by gcov or to enable GCC to perform profile-directed block ordering. Arc coverage information is saved in files with the `.da` (directed arc) extension after each run of an instrumented application. To enable profile-directed block-ordering optimizations, you must compile your application with this option, execute it with a representative data set, and then compile the program again with the same command-line options, adding the `-fbranch-probabilities` option. To use this option during code coverage analysis, you must also use the `-ftest-coverage` option.

`-fprofile-generate`: This optimization option tells GCC to instrument applications such that they produce profiling information that can be used for profile feedback optimization. When used, this option must be specified when compiling and also when linking your code.

`-fprofile-use`: This optimization option tells GCC to enable profile feedback optimization. Specifying this option enables the `-fbranch-probabilities`, `-fpeel-loops`, `-ftracer`, `-funroll-loops`, and `-fvpt` options.

`-fprofile-values`: This optimization option tells GCC to add code to generate data about expression values. When used with the `-fbranch-probabilities` option, this data is used to generate `REG_VALUE_PROFILE` notes that can be used in subsequent optimizations.

`-frandom-seed=`*`STRING`*: This debugging option specifies a seed value that GCC should use when it would otherwise use random numbers, such as when generating symbol names and uniquifiers in the data files used by gcov and the object code that produces them. If you use this option, you must use a different `STRING` value for each file that you compile.

`-freduce-all-givs`: This optimization option tells GCC to strength reduce all general-induction variables used in loops. Strength reduction is an optimization that uses previous calculations or values to eliminate more expensive calls or calculations. This option is activated by default when any GCC optimization level is used.

`-freg-struct-return`: This option when compiling C or C++ applications causes GCC to generate code that returns `struct` and `union` values in registers whenever possible. By default, GCC uses whichever of the `-fpcc-struct-return` or `-freg-struct-return` options is appropriate for the target system.

`-fregmove`: This optimization option tells GCC to reassign register numbers in order to maximize the amount of register tying, and is synonymous with the `-foptimize-register-move` option. This option is active by default when using GCC optimization level 2 or higher.

`-frename-registers`: This optimization option tells GCC to make use of any unallocated registers in order to attempt to avoid false dependencies in scheduled code. This option is therefore most frequently used on systems with large numbers of registers.

`-freorder-blocks`: This optimization option tells GCC to reorder basic blocks in compiled functions in order to improve code locality and minimize the number of branches taken. This option is enabled at optimization levels 2 and 3.

`-freorder-blocks-and-partition`: This optimization option tells GCC to perform the same reordering as specified by the `-freorder-blocks` option, but to also partition hot and cold basic blocks into separate sections of the assembly and object files in order to improve paging and cache locality.

`-freorder-functions`: This optimization option causes GCC to optimize function placement using profile feedback.

`-freplace-objc-classes`: This Objective C and Objective C++ language option tells GCC to emit a special marker that tells the linker not to statically link the resulting object file, enabling the Mac OS X dynamic loader, dylib, to load the class at runtime. This option is only useful on Mac OS X 10.3 or later, as it is associated with the NeXT runtime's fix-and-continue functionality.

`-frepo`: This option when compiling C++ applications enables automatic template instantiation at link time. Using this option also implies the `-fno-implicit-templates` option.

`-frerun-cse-after-loop`: This optimization option tells GCC to re-run common subexpression elimination after loop optimization has been performed.

`-frerun-loop-opt`: This optimization option tells GCC to run the loop optimizer twice, attempting to immediately capitalize on the results of the first pass.

`-freschedule-modulo-scheduled-loops`: This optimization option tells GCC that it can reschedule loops that have already been modulo scheduled. The `-fno-reschedule-modulo-scheduled-loops` option is used to prevent this rescheduling.

`-frounding-math`: This optimization option tells GCC to disable transformations and optimizations that assume default floating-point rounding behavior, which is round-to-zero for all floating-point-to-integer conversions, and round-to-nearest for all other arithmetic truncations.

`-frtl-abstract-sequences`: This size optimization option tells GCC to look for identical sequences of RTL code that it can turn into pseudoprocedures, replacing the original code with calls to the pseudoprocedure.

`-fsched-spec-load`: This optimization option tells GCC to move some load instructions where this is predicted to improve performance, reduce the execution path, or enable subsequent optimizations. This option is typically used only when you are also using the `-O2`, `-O3`, or `-fschedule-insns` options to schedule before register allocation.

`-fsched-spec-load-dangerous`: This optimization option is slightly more aggressive than the `-fsched-spec-load` option and tells GCC to be even more aggressive in moving load instructions in order to improve performance, reduce the execution path, or enable subsequent optimizations. This option is typically only used when you are also using the `-O2`, `-O3`, or `-fschedule-insns` options to schedule before register allocation.

`-fsched-stalled-insns=`*n*: This optimization option identifies the maximum number (*n*) of instructions (if any) that can be moved from the queue of stalled instructions into the ready list during the second scheduling pass.

`-fsched-stalled-insns-dep=`*n*: This optimization option identifies the number (*n*) of instruction groups (cycles) that will be examined for dependency on a stalled instruction that is a candidate for premature removal from the stalled instruction queue. This option is only meaningful if the `-fsched-stalled-insns` option is specified with a nonzero value.

`-fsched-verbose=`*n*: This debugging option tells GCC the amount of output to print to stderr (or specified dump listing file) during instruction scheduling. When $n > 0$, this option displays the same information as `-dRS`. When $n > 1$, this option also displays basic block probabilities, a detailed ready list, and unit/instruction information. When $n > 2$, this option also displays RTL information about abort points, control flow, and regions. When $n > 4$, this option also displays dependency information.

`-fsched2-use-superblocks`: This optimization option tells GCC to use a superblock scheduling algorithm when scheduling after register allocation. This option should only be used when the `-fschedule-isns2` option is specified, or with optimization level 2 or higher.

`-fsched2-use-traces`: This optimization option tells GCC to use the same algorithm as `-fsched2-use-superblocks`, but to also do code duplication as needed to produce faster, if larger, superblocks and resulting binaries.

`-fschedule-insns`: This optimization option, if supported on the target machine, tells GCC to attempt to reorder instructions to eliminate execution stalls that occur when the data required for an operation is unavailable. This option can be quite useful on systems with slow floating-point or memory load instructions by enabling other instructions to execute until the result of the other instructions are available.

`-fschedule-insns2`: This optimization option is similar to `-fschedule-ins` but tells GCC to perform an additional pass to further optimize instruction scheduling after register allocation has been performed. This option can further improve performance on systems with a relatively small number of registers or where memory load instructions take more than one cycle.

`-fsection-anchors`: This optimization option tells GCC to try to reduce the number of symbolic address calculations by using shared *anchor* symbols to address new entries, reducing the number of GOT (global offset table) entries and accesses.

`-fshared-data`: This code generation option causes GCC to locate data and nonconstant variables in the code that is currently being compiled in shared, rather than private, data. This may be useful on operating systems where shared data is literally sharable between processes running the same program.

`-fshort-double`: This code generation option tells GCC to use the same size when storing double and float data.

`-fshort-enums`: This code generation option tells GCC to minimize the amount of storage allocated to enumerated datatypes, only allocating as many bytes as necessary for the complete range of possible values. In other words, the amount of storage associated to `enum` datatypes will be the smallest integer datatype that provides sufficient space.

`-fshort-wchar`: This C programming language option causes GCC to override the underlying datatype used for `wchar_t` so that it is hardwired to be a short unsigned `int` instead of whatever the default datatype is for the target hardware.

`-fsignaling-nans`: This optimization option tells GCC to compile code that assumes that IEEE Not-a-Number signaling may generate user-visible traps during floating-point operations. This option assumes the `-ftrapping-math` option, and causes the `__SUPPORT_SNAN__` preprocessor macro to be defined.

`-fsigned-bitfields`: This C programming language option controls whether a bit field is signed, or unsigned when this option is unspecified. By default, bit fields are typically signed because it is consistent with basic integer types such as `int`, which are also signed unless the `-traditional` option is also specified on the command line.

`-fsigned-char`: This C programming language option causes GCC to define the `char` datatype as signed, requiring the same amount of storage as `signed char`. This option is equivalent to specifying the `-fno-unsigned-char` command-line option.

`-fsingle-precision-constant`: This optimization option tells GCC to handle floating-point constants as single-precision constants instead of implicitly converting them to double-precision constants.

`-fsplit-ivs-in-unroller`: This optimization option tells GCC that it can express the values of induction variables in later iterations of unrolled loops using the value from the first iteration, which breaks long dependency chains and improves scheduling efficiency.

`-fssa`: This optimization option tells GCC to perform its optimizations using static single assignment (SSA) form. The flow graph for each function is first translated into SSA form, optimizations are done while in that form, and the SSA form is then translated back into a flow graph. This option is not available in GCC 4.x compilers.

`-fssa-ccp`: This optimization option tells GCC to do sparse conditional constant propagation in SSA form. The `-fssa` option must also be specified in order to use this option and, like that option, this option is not available in GCC 4.x compilers.

`-fssa-dce`: This optimization option causes GCC to perform aggressive dead-code elimination in SSA form. The `-fssa` option must also be specified in order to use this option and, like that option, this option is not available in GCC 4.x compilers.

`-fstack-check`: This code generation option tells GCC to add extra code to force the operating system to notice whenever the stack is extended, helping ensure that applications do not accidentally exceed the stack size. (The operating system must still monitor the stack size.) This option is primarily useful in multithreaded environments, where more than one stack is in use. The automatic stack overflow detection provided by most systems in single-stack (single-process) environments is usually sufficient without using this option.

`-fstack-limit-register=`*reg* | `-fstack-limit-symbol=`*SYM*: These code generation options tell GCC to add extra code that ensures the stack does not grow beyond a certain value where *reg* is the name of a register containing the limit, or *SYM* is the address of a symbol containing the limit. A signal is raised if the stack grows beyond that limit. For most targets, the signal is raised before the stack crosses the boundary, so the signal can be caught and handled without taking special precautions. If you are using this option, you should also use the GNU linker to ensure that register names and symbol addresses are calculated and applied correctly.

`-fstack-protector`: This optimization option tells GCC to generate extra code to check for buffer overflows in functions with vulnerable objects.

`-fstack-protector-all`: This optimization option tells GCC to generate extra code to check for buffer overflows in all functions.

`-fstats`: This C++ option tells GCC to display front-end processing statistics once compilation has completed. The GNU C++ (G++) development team uses this information.

`-fstrength-reduce`: This optimization option tells GCC to do loop strength reduction and iteration variable elimination.

`-fstrict-aliasing`: This optimization option tells GCC to use the strictest aliasing rules applicable to the language being compiled. For C (and C++), this performs optimizations based on expression type. Objects of two different types are assumed to be located at different addresses unless the types are structurally similar. For example, an `unsigned int` can be an alias for an `int`, but not for a `void *` or `double`. A character type can be an alias for any other type. This option can help detect aliasing errors in potentially complex datatypes such as unions.

`-fsyntax-only`: This diagnostic option tells GCC to check the code for syntax errors, without actually compiling any part of it.

`-ftemplate-depth-`*n*: This C++ option sets the maximum instantiation depth for template classes to *n*. Limits on template instantiation depth are used to detect infinite recursion when instantiating template classes. ANSI/ISO C++ standards limit instantiation depth to 17.

`-ftest-coverage`: This debugging option causes GCC to create two data files for use by the gcov code-coverage utility. The first of these files is source.bb, which provides a mapping of basic blocks to line numbers in the source code. The second of these files is source.bbg, which contains a list of all of the arcs in the program's flow graph. If this option is used with the `-fprofile-arcs` option, executing the compiled program will also create the data file source.da, which provides runtime execution counts used in conjunction with the information in the source.bbg file. Coverage data generally maps better to source files if no optimization options are used when generating code coverage information.

`-fthread-jumps`: This optimization option optimizes jumps that subsequently perform redundant comparisons, skipping those comparisons, and redirecting the code to the appropriate point later in the code execution flow.

`-ftime-report`: This debugging option causes GCC to display statistics about the time spent in each compilation pass.

`-ftls-model=`*`MODEL`*: This code generation option tells GCC to use a specific thread-local storage model. *`MODEL`* should be either `global-dynamic`, `local-dynamic`, `initial-exec`, or `local-exec`. The default is `global-dynamic`, unless the `-fpic` option is used, in which case the default is `initial-exec`.

`-ftracer`: This code generation pass simplifies the control flow of functions, allowing other optimizations to do a better job.

`-ftrapv`: This optimization option causes GCC to generate traps for signed overflow on addition, subtraction, and multiplication operations.

`-ftree-ccp`: This optimization option tells GCC to perform sparse conditional constant propagation on trees, and is enabled at optimization level 1 and higher.

`-ftree-ch`: This optimization option tells GCC to perform loop header copying on trees, which can improve the effectiveness of code motion optimization and saves a jump. This option is enabled at all optimization levels except for `-Os`, because it can increase code size.

`-ftree-copy-prop`: This optimization option tells GCC to perform copy propagation on trees, eliminating unnecessary copies. This option is enabled at all optimization levels.

`-ftree-copyrename`: This optimization option tells GCC to perform copy renaming on trees and is active at all optimization levels.

`-ftree-dce`: This optimization option tells GCC to perform dead-code elimination on trees and is enabled at all optimization levels.

`-ftree-dominator-opts`: This optimization option tells GCC to perform a variety of simple scalar cleanups based on a denominator tree traversal, including constant/copy propagation, expression simplification, redundancy elimination, and range propagation, and also performs jump threading, reducing jumps to jumps. This option is enabled at all optimization levels.

`-ftree-dse`: This optimization option tells GCC to perform dead store elimination on trees, and is enabled at all optimization levels.

`-ftree-fre`: This optimization option tells GCC to perform full redundancy elimination on trees, which only considers expressions that are computed on all paths to the redundant computation. This option is enabled at all optimization levels.

`-ftree-loop-im`: This optimization option tells GCC to perform loop invariant motion on trees, moving the operands of invariant conditions out of loops and moving invariants that would be hard to handle at the RTL level, such as function calls and operations that expand into nontrivial numbers of instructions.

`-ftree-loop-ivcanon`: This optimization option tells GCC to create a variable to track the number of loop iterations, which can be useful in subsequent loop unrolling.

`-ftree-loop-linear`: This optimization option tells GCC to perform linear loop transformations on trees, which can improve cache performance and enable further loop optimizations.

`-ftree-loop-optimize`: This optimization option tells GCC to perform loop optimization on trees and is enabled at all optimization levels.

`-ftree-lrs`: This optimization option tells GCC to perform live range splitting in SSA to normal translations and is enabled at all optimization levels.

`-ftree-pre`: This optimization option tells GCC to perform partial redundancy elimination on trees and is enabled at optimization levels 2 and 3.

`-ftree-salias`: This optimization option tells GCC to perform structure alias analysis on trees and is enabled at all optimization levels.

`-ftree-sink`: This optimization option tells GCC to perform forward store motion on trees and is enabled at all optimization levels.

`-ftree-sra`: This optimization option tells GCC to perform the scalar replacement of aggregates and is enabled at all optimization levels.

`-ftree-store-ccp`: This optimization option tells GCC to perform sparse conditional constant propagation on trees and is enabled at all optimization levels.

`-ftree-store-copy-prop`: This optimization option tells GCC to perform copy propagation of memory loads and stores and is enabled at optimization levels 2 and 3.

`-ftree-ter`: This optimization option tells GCC to perform temporary expression replacement in SSA to normal translations and is enabled at all optimization levels.

`-ftree-vect-loop-version`: This optimization option tells GCC to do loop versioning when doing loop vectorization, generating both vectorized and nonvectorized versions of the loop that are selected at runtime based on checks for alignment or dependencies. This option is enabled at all optimization levels except for `-Os`.

`-ftree-vectorize`: This optimization option tells GCC to perform loop vectorization on trees and is enabled at all optimization levels.

`-ftree-vectorizer-verbose=`*n*: Specifying this debugging option controls the amount of debugging information produced during loop vectorization. When $n = 0$, no output is produced. Higher values of *n* produce increasing amounts of information about loops that were vectorized, loops that were considered for vectorization, and loops that were not vectorized.

`-funroll-all-loops`: This optimization causes GCC to unroll all loops, even if the number of times they are executed cannot be guaranteed when the loop is entered. Though this usually makes programs run more slowly, it provides opportunities for subsequent optimization through code elimination.

`-funroll-loops`: This optimization causes GCC to unroll loops, but limits the loops that will be unrolled to those where the number of times they are executed can be determined at compile time or when entering the loop. Using this option implies both the `-fstrength-reduce` and `-frerun-cse-after-loop` options. Using this option makes code larger but does not guarantee improved performance or execution speed. It does provide opportunities for subsequent optimization through other GCC optimization options.

`-funsafe-loop-optimizations`: This optimization option tells GCC to enable additional loop optimizations by assuming that all loop indices are valid and that no loops with nontrivial exit conditions are not infinite.

`-funsafe-math-optimizations`: This optimization option enables GCC to perform optimizations for floating-point arithmetic that assumes that arguments and results are valid and may violate IEEE or ANSI standards. This option should never be used in conjunction with any standard optimization (`-O`) option because it can result in incorrect output in programs that depend on an exact implementation of the IEEE or ISO specifications for math functions.

`-funsigned-bitfields`: This C language option controls whether a bit field is signed or unsigned when neither keyword is specified. Bit fields are ordinarily signed by default because this is consistent with basic integer types such as `int`, which are signed types. Using the `-traditional` option forces all bit fields to be unsigned regardless of whether this option is specified.

`-funsigned-char`: This C language option forces the `char` datatype to be unsigned. This overrides any default character datatype defaults for a given system. This option is valuable when porting code between system types that have different defaults for the `char` datatype. Note that the `char` type is always distinct from `signed char` and `unsigned char` even though its behavior is always the same as either of those two.

`-funswitch-loops`: This optimization option tells GCC to move branches with loop invariant conditions outside loops, providing duplicates of the loop based on the condition result on both branches.

`-funwind-tables`: This code generation option tells GCC to enable exception handling and generates any static data used when propagating exceptions. This option is similar to the `-fexceptions` option, but generates static data rather than code. This option is rarely used from the command line and is usually incorporated into language processors that require this behavior (such as C++).

`-fuse-cxa-atexit`: This C++ option causes GCC to register destructors for objects with static storage duration using the `__cxa_atexit()` function rather than the `atexit` function. This option is required for fully standards-compliant handling of static destructors but only works on systems where the C library supports the `__cxa_atexit()` function.

`-fvar-tracking`: This debugging option tells GCC to run an additional variable tracking pass that computes where variables are stored at each code position and therefore provides better debugging information. If supported by the debug information format, this option is enabled when compiling for any optimization level and when generating debug information using `-g`.

`-fvariable-expansion-in-unroller`: This optimization option tells GCC to create multiple copies of local variables during loop unrolling.

`-fverbose-asm`: This code generation option inserts extra comments into generated assembly code to make it more readable. This option is generally only used during manual optimization or by people who are verifying the generated assembly code (such as the GCC maintenance and development teams).

`-fvisibility=`*VALUE*: This code generation option tells GCC to set the default ELF symbol visibility to `VALUE`, where *VALUE* is one of the standard C++ values `DEFAULT`, `INTERNAL`, `HIDDEN`, or `PROTECTED`. The default value is `DEFAULT` (public), which makes all symbols visible. See the discussion of visibility attributes in Chapter 2 for alternate ways of specifying symbol visibility through attributes.

`-fvisibility-inlines-hidden`: This C++ language option causes all inlined methods to be marked with `__attribute__ ((visibility ("hidden")))` so that they do not appear in the export table of shared objects, which can therefore significantly improve shared object load time.

`-fvolatile`: This code generation option tells GCC to consider all memory references through pointers to be volatile.

`-fvolatile-global`: This code generation option tells GCC to consider all memory references to `extern` and global data items to be volatile. This option does not cause GCC to consider static data items to be volatile.

`-fvolatile-static`: This code generation option tells GCC to consider all memory references to static data to be volatile.

`-fvpt`: This optimization option when used with the `-fprofile-arcs` option tells GCC to add code to collect information about expression values that can be used by the `-fbranch-probabilities` option during subsequent optimizations.

`-fvtable-gc`: This C++ option tells GCC to generate special relocations for vtables and virtual function references. This enables the linker to identify unused virtual functions and zero out vtable slots that refer to them. This option is commonly used with the `-ffunction-sections` option and the linker's `-Wl,--gc-sections` option, in order to also discard the functions themselves. This optimization requires that you are also using GNU as and GNU ld, and is not supported on all system types. Note that the `-Wl,--gc-sections` option is ignored unless the `-static` option is also specified.

`-fweb`: This optimization option tells GCC to construct the webs used for register allocation purposes and assign each web to an individual pseudoregister. This enables the register allocation pass to operate on these pseudoregisters and also strengthens other optimization passes, such as common subexpression elimination, loop optimization, and dead-code removal. This option is enabled by default when `-funroll-loops` is specified.

`-fwhole-program`: This optimization option tells GCC to assume that the current compilation unit represents the entire program, enabling more aggressive optimization because all functions (except `main`) and variables can be assumed to be static.

`-fworking-directory`: This preprocessor option tells GCC to generate line numbers in preprocessor output that identify the current working directory at compile time. This option is enabled when any debugging information is being generated.

`-fwrapv`: This code generation option tells GCC to assume that signed arithmetic overflow wraps around using two's-complement representation. This option is enabled by default for GCC's Java compiler, as required by the Java specification.

`-fwritable-strings`: This C language option tells GCC to store string constants in the writable data segment without uniquifying them. This option is provided for compatibility with older programs that assume they can write into string constants (even though this is poor form). Specifying the `-traditional` option also causes this behavior. This option is not supported in the GCC 4.x compilers.

`-fzero-link`: This Objective C and Objective C++ language option tells GCC to preserve calls of `objc_getClass()` that would ordinarily be replaced by static references that are initialized at load time. This option is only useful when compiling for the Mac OS X platform.

`-g`: This debugging option causes GCC to include debugging and symbol table information in object files, which can subsequently be used by GDB. The format of these object files and the debugging information that they contain depends on the native binary format associated with the platform, and can be one of the following: COFF (SVR3 systems), DWARF (SVR4), STABS (Linux), or XCOFF (AIX).

> On most systems that use STABS format, including the `-g` option enables the use of extra debugging information that only GDB can employ; this extra information makes debugging work better in GDB but will probably make other debuggers crash or refuse to read the program. If you want to ensure generation of the extra information, use `-gstabs+`, `-gstabs`, `-gxcoff+`, `-gxcoff`, `-gdwarf-1+`, `-gdwarf-1`, or `-gvms`. Users of the `-gvms` option have my inherent sympathy.
>
> Unlike most other C compilers, GCC allows you to use `-g` with optimization options such as `-O`, but this is not recommended. As you would expect, optimized code may occasionally produce surprising results due to optimization in loop structure, loop control, variable elimination, statement movement, result compression, and so on.
>
> Other than the format-related options listed previously, related debugging options include `-ggdb`, `-glevel` (and similar level options), profiling options such as `-p`, `-pg`, `-Q`, `-ftime-report`, `-fmem-report`, `-fprofile-arcs`, `-ftest-coverage`, `-dletters`, `-fdump-unnumbered`, `-fdump-translation-unit`, `-fdump-class-hierarchy`, `-fdump-tree`, `-fpretend-float`, `-print-multi-lib`, `-print-prog-name=program`, `-print-libgcc-file-name`, `-print-file-name=libgcc.a`, `-print-search-dirs`, `-dumpmachine`, `-dumpversion`, and `-dumpspecs`.

`-g`*`LEVEL`* | `-ggdb`*`LEVEL`* | `-gstabs`*`LEVEL`* | `-gcoff`*`LEVEL`* | `-gxcoff`*`LEVEL`* | `-gvms`*`LEVEL`*: These debugging options cause GCC to produce debugging information in various formats, using the `LEVEL` value to specify the amount of information displayed. The default value of `LEVEL` is 2. Level 1 produces minimal information, sufficient for displaying backtraces in parts of the program that the user does not expect to debug. Level 3 includes information such as the macro definitions present in the program.

`-gcoff`: This causes GCC to produce debugging information in the COFF format used by SDB on most System V systems prior to System V Release 4.

`-gdwarf`: This option causes GCC to produce debugging information in DWARF (version 1) format, used by the SDB debugger on most System V Release 4 systems.

`-gdwarf+`: This option causes GCC to produce debugging information in DWARF version 1 format using GNU extensions understood only by GDB. Debugging applications compiled with this option using other debuggers, or compiling them with toolchains that are not strictly composed on GNU tools may cause unexpected behavior.

`-gdwarf-2`: This option causes GCC to produce debugging information in DWARF version 2 format, which is used by the DBX debugger on IRIX 6.

`-gen-decls`: This Objective C option causes GCC to dump interface declarations for all classes seen in each source file to a file named sourcename.decl.

`-ggdb`: This option causes GCC to produce debugging information that is targeted for use by GDB. Using this option tells GCC to employ the debugging options that provide the greatest amount of detail for the target platform.

`-gstabs`: This option forces GCC to produce debugging information in the STABS format without using the GDB extensions. The STABS format is used by most debuggers on BSD and Linux systems. GDB can still be used to debug these applications but will not be able to debug them as elegantly as if the `-gstabs+` option had been specified.

`-gstabs+`: This option causes GCC to produce debugging information in STABS format, using the GDB extensions that are only meaningful to GDB. Debugging applications compiled with this option, using other debuggers or compiling them with toolchains that are not strictly composed on GNU tools may cause unexpected behavior.

`-gvms`: This sad, lonely debugging option is only relevant to users who are writing applications intended to be executed and debugged by the DEBUG application on VMS systems. `SYS$SYSOUT`, anyone?

`-gxcoff`: This option causes GCC to produce debugging information in XCOFF format, if supported on the execution platform. XCOFF is the format used by the classic Unix DBX debugger on IBM RS/6000 systems.

`-gxcoff+`: This option causes GCC to produce debugging information in XCOFF format, using debugging extensions that are only understood by GDB. Debugging applications compiled with this option using other debuggers or compiling them with toolchains that are not strictly composed on GNU tools may cause unexpected behavior.

`-H`: This option causes the preprocessor to print the name of each header file used during preprocessing. Each name is indented to show how deeply nested that `#include` statement is.

`--help`: This output option causes GCC to display a summary list of the command-line options that can be used with GCC. Using this option in conjunction with the `-v` option causes GCC to also pass the `--help` option to all subsequent applications invoked by GCC, such as the assembler, linker, and loader, which will also cause them to display a list of many of the standard command-line options that they accept. Adding the `-W` command-line option to the `--help` and `-v` options will also cause GCC and subsequent applications in the toolchain to display command-line options that can be used but have no documentation except for in this book.

`-I-`: This directory search option is used during preprocessing to identify additional directories that should be searched for the #include definition files used in C and C++ applications. Any directories specified using the `-I` option before the `-I-` option are only searched for files referenced as #include "file", not for files referenced as #include <file>. If you use the `-I` option to specify additional directories to search after using the `-I-` option, those additional directories will be searched both for files referenced as #include "file" and #include <file>.

> Using the `-I-` option also keeps the preprocessor from examining the working directory for #include files referenced in #include statements. You can subsequently use the `-I` option to explicitly search the working directory if you specify it by name on the command line.

■ **Note** Using the `-I-` option does not cause GCC to ignore the standard system directories in which #include files are typically located. To do this, specify the `-nostdinc` option.

`-IDIR`: This directory search option is used during preprocessing to identify additional directories that should be added to the beginning of the list of directories that are searched for #include files. This can be used to override #include files in an operating system's include directories that have the same name as #include files local to your application's source code. If you use more than one `-I` option on the GCC command line, the specified directories are scanned in the order that they are specified, from left to right, followed by the standard system directories.

■ **Note** You should not use this option to add #include directories that contain vendor-supplied system header files. Such directories should be specified using the `-isystem` option, which puts them in the search order after directories specified using the `-I` option, but before the system header files.

■ **Caution** Using the `-I` option to specify system include directories (such as `/usr/include` and `/usr/include/sys`) is not a good idea. GCC's installation procedure often corrects bugs in system header files by copying them to GCC's include directory and then modifying its copies. Because GCC is your friend, it will display a warning whenever you specify a system include directory using this option.

`-imacros` *`file`*: This preprocessor option causes cpp to load all macros defined in the specified file, but to discard any other output produced by scanning the file. All files specified by the `-imacros` options are processed before any files specified by using the `-include` option. This enables cpp to acquire all of the macros defined in the specified file without including any other definitions that it contains.

`-imultilib` *`dir`*: This preprocessor option tells GCC to use `dir` as a subdirectory of the directory that contains target-specific C++ headers.

`-include` *`file`*: This preprocessor option causes cpp to process the specified file as if it were specified using a `#include "`*`file`*`"` statement in the first line of the primary source file. The first directory searched for *file* is the preprocessor's working directory, which may differ from the directory containing the main source file. If the specified file is not found there, cpp searches for it through the remainder of the normal include file search chain. If the `-include` option is specified multiple times, the specified files are included in the order that they appear on the command line.

`-iprefix` *`prefix`*: This preprocessor option causes cpp to use `prefix` as the prefix for subsequent `-iwithprefix` options. If the specified prefix is a directory, it should end with a trailing `/`.

`-iquote` *`dir`*: This directory option tells GCC to search `dir` for header files specified using `#include "`*`file`*`"` before searching all directories specified using `-I` and before all system directories.

`-isysroot` *`dir`*: This preprocessor option tells GCC to use `dir` as the logical root directory for all searches for header/include files.

`-isystem` *`dir`*: This preprocessor option causes cpp to search `dir` for header files after all directories specified by the `-I` option have been searched, but before the standard system directories. The specified directory is also treated as a system include directory.

`-iwithprefix dir|-iwithprefixbefore dir`: These preprocessor options cause cpp to append the specified `dir` to any prefix previously specified with the `-iprefix` option, and to add the resulting directory to cpp's search path for include directories. Using the `-iwithprefix` option adds this directory to the beginning of cpp's search path (just as the `-I` option would). Using the `-iwithprefixbefore` option adds this directory to the end of cpp's search path (just as the `-idirafter` option would).

`-L`*`dir`*: This directory search option causes the linker to add the specified directory `dir` to the list of directories to be searched for libraries specified using the `-l` command-line option.

`-l`*`LIBRARY`*`|-l` *`LIBRARY`*: These linker options cause the linker to search the library with the base library name of *LIBRARY* (i.e., the full name of lib*LIBRARY*.a) when linking. Object files and libraries are searched based on the order in which they are specified on the linker command line. The only difference between these options and explicitly specifying the name of the library is that these options surround *LIBRARY* with `lib` and `.a`, and search for the specified library in multiple directories.

`-M`: This preprocessor option causes cpp to generate a rule suitable for use by the make program that describes the dependencies of the main source file, rather than actually preprocessing the source files. This make rule contains the name of that source file's corresponding output file, a colon, and the names of all included files, including those coming from all `-include` or `-imacros` command-line options. Unless explicitly specified using the `-MT` or `-MQ` command-line options, the name of the object file is derived from the name of the source file in the standard fashion, by replacing any existing extension with the `.o` extension associated with object files. If the rule is extremely long, it is broken into multiple lines whose new lines are escaped using the backslash character (`/`).

> You can use the `-M` preprocessor option with the `-MF` option to specify the output file, which is recommended if you are also using debugging options such as `-dM` to generate debugging output. Specifying this option also invokes the `-E` option, automatically defining the `__GNUC__`, `__GNUC_MINOR__`, and `__GNUC_PATCHLEVEL__` macros and causing the compilation process to stop after the preprocessing phase.

`-MD`: This preprocessor option is equivalent to specifying the `-M -MF file` options, except that the `__GNUC__`, `__GNUC_MINOR__`, and `__GNUC_PATCHLEVEL__` macros are not defined and that compilation continues after the preprocessing stage. Since this option does not take an output `file` argument, cpp first checks if the name of an output file has been specified using the `-o` option. If so, the output file is created using the basename of that file and replacing any existing extension with the `.d` extension. If not, the name of the output file is derived from the name of the input file, again replacing any existing suffix with the `.d` suffix. Because the `-MD` option does not imply the `-E` option, this option can be used to generate a dependency make rule output file as part of the complete compilation process.

■ **Note** If the `-MD` option is used in conjunction with the `-E` option, any `-o` option specifies the name of the dependency output file. If used without the `-E` option, the `-o` option specifies the name of the final object file.

`-MF file`: This preprocessor option along with the `-M` or `-MM` options identifies the name of a file to which cpp should write dependency information. If the `-MF` option is not specified, the dependency rules are sent to the place to which preprocessor output would have been sent.

`-MG`: This preprocessor option along with the `-M` or `-MM` options causes cpp to treat missing header files as generated files that should be located in the same directory as the source file. This option also suppresses generating preprocessed output, because a missing header file is considered an error. This option is often used when automatically updating Makefiles.

`-MM`: This preprocessor option causes GCC to generate the same output rule as that produced by the `-M` option, the difference being that the generated rules do not list include files that are found in system include directories, or include files that are included from system include files.

`-MMD`: This preprocessor option causes GCC to generate the same output rule as that produced by the `-M` option, the difference being that the generated rules do not list user include files or other user include files that are included by user include files.

`-MP`: This preprocessor option causes cpp to add a fake target for each dependency other than the main file, causing each to depend on nothing through a dummy rule. These rules work around errors that may be generated by the make program if you remove header files without making corresponding Makefile updates.

`-MQ target`: This preprocessor option causes cpp to change the output target in the rule emitted by dependency-rule generation. Instead of following the standard extension-substitution naming convention, using this option sets the name of the output file to the name that you specify. Any characters that have special meaning to the make program are automatically quoted.

`-MT target`: This preprocessor option causes cpp to change the output target in the rule emitted by dependency-rule generation. Instead of following the standard extension-substitution naming convention, using this option sets the name of the output file to the exact filename that you specify.

`-no-integrated-cpp`: This debugging option for C and C++ applications causes GCC to invoke the external cpp rather than the internal C preprocessor that is included with GCC. The default is to use the internal cpp that is provided with GCC. Specifying this option in conjunction with the `-B` option enables you to integrate a custom C preprocessor into your GNU toolchain and allows you to integrate a user-supplied cpp that you then specify via the `-B` option. For example, you could name your preprocessor mycpp and then cause GCC to use it by specifying the `-no-integrated-cpp -Bmy` option sequence.

`-nodefaultlibs`: This linker option tells GCC not to use the standard system libraries when linking.

`-nostartfiles`: This linker option tells GCC not to use the standard system start files (crt0.o, etc.) when linking.

`-nostdinc`: This C language directory search option prevents the preprocessor from searching the standard system directories for #include files specified using `#include <file>` statements.

`-nostdinc++`: This C++ language directory search option prevents the preprocessor from searching the C++-specific system directories for #include files specified using `#include <file>` statements. Standard system include directories such as `/usr/include` and `/usr/include/sys` are still searched.

`-nostdlib`: This linker option tells GCC not to use either the standard system libraries or the startup files when linking.

`-O | -O1`: These optimization options cause GCC to attempt to reduce the size and improve the performance of the target application. On most systems, the `-O` option turns on the `-fthread-jumps` and `-fdelayed-branch` options.

> Without optimization, GCC's primary goal is to compile applications as quickly as possible. A secondary goal is to make it easy to subsequently debug those applications if necessary. Compiling with optimization will almost certainly take more time and will also require more memory when compiling any sizable function or module. Optimization may also combine variables or modify the execution sequence of an application, which can make it difficult to debug an optimized application. You rarely want to specify an optimization option when compiling an application for debugging unless you are debugging the optimization process itself.

`-O0`: This optimization option explicitly disables optimization. This option is the equivalent of not specifying any `-O` option. While seemingly meaningless, this option is often used in complex Makefiles where the optimization level is specified in an environment variable or command-line Makefile option.

`-O2`: This optimization option causes GCC to attempt additional optimizations beyond those performed for optimization level 1. In this optimization level, GCC attempts all supported optimizations that do not trade off between size and performance. This includes all optimization options with the exception of loop unrolling (`-funroll-loops`), function inlining (`-finline-functions`), and register renaming (`-frename-registers`). As you would expect, using the `-O2` option increases both compilation time and the performance of compiled applications.

`-O3`: This optimization option causes GCC to attempt all performance optimizations, even if they may result in a larger compiled application. This includes all optimization options performed at optimization levels 1 and 2, plus loop unrolling (`-funroll-loops`), function inlining (`-finline-functions`), and register renaming (`-frename-registers`).

`-Os`: This optimization option tells GCC to optimize the resulting object code and binary for size rather than for performance.

`-o` *`file`*: This output option tells GCC to write its output binary to the file *file*. This is independent of the type of output that is being produced: preprocessed C code, assembler output, and object module, or a final executable. If the `-o` option is not specified, executable output will be written to the following files:

- *Executables*: Written to a file named a.out (regardless of whether a.out is the execution format)
- *Object files*: Written to files with the input suffix replaced with `.o` (*file*.c output is written to *file*.o)
- *Assembler output*: Written to files with the input suffix replaced with `.s` (for example, assembler output for the file *file*.c output is written to *file*.s)
- *Preprocessed C source code*: Written to standard output

`-P`: This preprocessor option causes the preprocessor to inhibit the generation of line markers in its output and is usually used when the output from the preprocessor will be used with a program that may not understand the line markers.

`-p`: This debugging option causes GCC to generate extra code that will produce profiling information that is suitable for the analysis program, prof. This option must be used both when compiling and linking the source file(s) that you want to obtain profiling information about.

`--param` *`NAME=VALUE`*: This optimization option provides control over the parameters used to control various optimization options. For example, GCC will not inline functions that contain more than a certain number of instructions. The `--param` command-line option gives you fine-grained control over limits such as this. All of these parameters are integer values.

Possible parameters that you can specify are the following:

- `max-delay-slot-insn-search`: The maximum number of instructions to consider when looking for an instruction to fill a delay slot. Increasing values mean more aggressive optimization, resulting in increased compilation time with a potentially small improvement in performance.
- `max-delay-slot-live-search`: The maximum number of instructions to consider while searching for a block with valid live register information when trying to fill delay slots. Increasing this value means more aggressive optimization, resulting in increased compilation time.
- `max-gcse-memory`: The approximate maximum amount of memory that will be allocated in order to perform global common subexpression elimination optimization. If more memory than the specified amount is required, global common subexpression optimization will not be done.
- `max-gcse-passes`: The maximum number of passes of global common subexpression elimination to run.
- `max-inline-insns`: Functions containing more than this number of instructions will not be inlined. This option is functionally equivalent to using the `-finline-limit` option with the same value.
- `max-pending-list-length`: The maximum number of pending dependencies scheduling will allow before flushing the current state and starting over. Large functions with few branches or calls can create excessively large lists that needlessly consume memory and resources.

`-pass-exit-codes`: This output option causes GCC to return the numerically highest error code produced during any phase of the compilation process. GCC typically exists with a standard Unix error code of 1 if an error is encountered in any phase of the compilation.

`-pedantic`: This diagnostic/warning option causes GCC to display all warnings demanded for strict ISO C and ISO C++ compliance. Using this option does not verify ISO compliance, because it only issues warnings for constructs for which ISO C and ISO C++ require such a message (plus some that have been added in GCC but are not strictly mandatory for ISO C/C++). This option also causes GCC to refuse to compile any program that uses extensions and C/C++ syntax that are not ISO compliant. Valid ISO C and ISO C++ programs should compile properly with or without this option (though some may require additional restrictive options such as `-ansi` or an `-std` option specifying a specific version of ISO C).

> Using the `-pedantic` option does not generate warning messages for alternate keywords whose names begin and end with __, and are also disabled for expressions that follow these keywords. These extensions are typically only used in system software, rather than application software.

`-pedantic-errors`: This diagnostic/warning message causes GCC to display error messages rather than warnings for non–ISO C/C++ constructs.

`-pg`: This debugging option causes GCC to generate extra code that produces profile information suitable for use by the analysis program gprof. This option must be used both when compiling and linking the source file(s) that you want to obtain profiling information about.

`-pie`: This linker option tells GCC to produce a position-independent executable for targets that support that type of executable. When using this option for final linking, you must specify the same code generation options as were used to generate each object module.

`-pipe`: This output option causes GCC to use pipes rather than temporary files when exchanging data between various stages of compilation. This can cause problems on systems where non-GNU tools (such as a custom preprocessor, assembler, and so on) are used as part of the compilation toolchain.

`-print-file-name=`*`LIBRARY`*: This debugging option causes GCC to display the full pathname of the specified library. This option is often used when you are not linking against the standard or default system libraries, but you do want to link with a specific library, such as libgcc.a, as in the following example:

```
gcc -nodefaultlibs foo.c bar.c... 'gcc -print-file-name=libgcc.a'
```

When used in this form, surrounding the command `gcc -print-file-name=libgcc.a` with backquotes causes the command to be executed and its output displayed, which is then incorporated on the compilation command line as an explicit reference to the target library.

■ **Note** If the specified library is not found, GCC simply echoes the library name.

`-print-libgcc-file-name`: This debugging option is a shortcut for using the option `-print-file-name=libgcc.a`, and is used in the same circumstances.

`-print-multi-directory`: This debugging option causes GCC to print the directory name corresponding to the multilib selected by any other switches that are given on the command line. This directory is supposed to exist in the directory defined by the `GCC_EXEC_PREFIX` environment variable.

`-print-multi-lib`: This debugging option causes GCC to display the mapping from multilib directory names to compiler switches that enable them. This information is extracted from the specification files used by the compiler, in which the directory name is separated from the switches by a semicolon, and each switch starts with an @ symbol instead of the traditional dash/minus symbol, with no spaces between multiple switches.

WHAT ARE MULTILIBS?

Multilibs are libraries that are built multiple times, each with a different permutation of available machine-specific compiler flags. This makes it easy for GCC to produce output targeted for multiple platforms and to take advantage of different types of optimizations for similar but different platforms by having precompiled versions of libraries targeted for each.

Multilibs are typically built when you are using GCC with multiple targets for a given architecture where you need to support different machine-specific flags for various combinations of targets, architectures, subtargets, subarchitectures, CPU variants, special instructions, and so on.

You can display any multilibs available on your system by executing the `gcc -print-multi-lib` command. Multilibs are specified in entries in the compiler specification files that are stored in the install-dir/lib/architecture/version/specs file associated with each GCC installation.

`-print-prog-name=`*`PROGRAM`*: This debugging option causes GCC to display the full pathname of the specified program, which is usually a part of the GNU toolchain.

■ **Note** If the specified program is not found, GCC simply echoes the name of the specified program.

`-print-search-dirs`: This debugging option causes GCC to print the name of its installation directory and the program and library directories that it will search for mandatory files, and then exit. This option is useful when debugging installation problems reported by GCC. To resolve installation problems, you can either rebuild GCC correctly, or symlink or move any problematic components into one of the directories specified in the output of this command. You can often temporarily hack around installation problems by setting the environment variable `GCC_EXEC_PREFIX` to the full pathname (with a trailing `/`) of the directory where missing components are actually installed.

`-Q`: This debugging option causes GCC to print the name of each function as it is compiled and print some general statistics about each pass of the compiler.

`-rdynamic`: This linker option tells GCC to pass the `-export-dynamic` flag to the ELF linker on targets that support it, which instructs the linker to add information for all symbols in the code to the dynamic symbol table, not just those that are used.

`-remap`: This preprocessor command-line option enables special code in the preprocessor that is designed to work around filename limitations in filesystems, such as the MS-DOS FAT filesystem that only permits short, silly filenames.

`-S`: This output option causes GCC to stop after generating the assembler code for any specified input files. The assembler file for a given source file has the same name as the source file but has an `.s` extension instead of the original extension of the input source file.

`-s`: This linker option causes the linker to remove all symbol table and relocation information from the final executable.

`-save-temps`: This debugging option causes GCC to preserve all temporary files produced during the compilation process, storing them in the working directory of the compilation process. This produces `.i` (preprocessed C input) and `.s` (assembler) files for each file specified for compilation. These files have the same basename as the original input files but a different extension.

`-shared`: This linker option causes the linker to produce a shared object that can then be linked with other objects to form an executable. When using this option you should make sure that all of the shared objects that you will eventually link together are compiled with the same set of `-fpic`, `-fPIC`, or model suboption compiler options.

`-shared-libgcc` | `-static-libgcc`: These linker options cause the linker to force the use of the shared or static version of libgcc.a on systems that provide both. These options have no effect if a shared version of libgcc.a is not available. The shared version is generally preferable because this makes it easier to do operations such as throwing and catching exceptions across different shared libraries.

`-specs=`*`file`*: This directory search option tells GCC to process *`file`* to potentially override the default specs used by the compiler driver to identify default options and programs used during the compilation chain.

`-static`: This linker option causes the linker to prevent linking against shared libraries on systems that support them.

`-std=std`: This C language option tells GCC which C standard the input file is expected to conform to. You can use the features of a newer C standard even without specifying this option as long as they do not conflict with the default ISO C89 standard. Specifying a newer version of ISO C using `-std` essentially enables GCC's support for features in the specified standard to the default features found in ISO C89. Specifying a newer C standard changes the warnings that will be produced by the `-pedantic` option, which will display the warnings associated with the new base standard, even when you specify a GNU extended standard.

Possible values for `std` are the following:

- `c89` | `iso9899:1990`: ISO C89 (the same as using the `-ansi` switch).
- `iso9899:199409`: ISO C89 as modified in amendment 1.
- `c99` | `iso9899:1999`: ISO C99. This standard is not yet completely supported. For additional information, see `http://gcc.gnu.org/version/c99status.html`, where *version* is a major GCC version such as gcc-3.1, gcc-3.2, gcc-3.3, and so on. At the time this book was written the standard names `c9x` and `iso9899:199x` could also be specified but were deprecated.
- `gnu89`: ISO C89 with some GNU extensions and ISO C99 features. This is the default C standard used by gcc (i.e., the value used when the `-std` option is not given).
- `gnu99`: ISO C99 with some GNU extensions. This will become the default once ISO C99 is fully supported in GCC. The name `gnu9x` can also be specified, but is deprecated.

`-symbolic`: This linker option causes the linker to bind references to global symbols when building a shared object and to warn about any unresolved references.

`--sysroot=dir`: This directory search option tells GCC to use `dir` as the root of the file system in which header files and libraries are located.

`--target-help`: This C language output option causes GCC to print a list of options that are specific to the compilation target. Using this option is the easiest way to get an up-to-date list of all platform-specific GCC options for the target platform.

`-time`: This debugging option causes GCC to display summary information that lists user and system CPU time (in seconds) consumed by each step in the compilation process (cc1, as, ld, and so on). *User time* is the time actually spent executing the specified phase of the compilation process. *System time* is the time spent executing operating system routines on behalf of that phase of the compilation process.

`-traditional`: This C language option tells GCC to attempt to support some aspects of traditional C compilers and non-ANSI C code, and also automatically invokes the parallel `-traditional-cpp` option for GCC's internal C preprocessor. This option can only be used if the application being compiled does not reference header (#include) files that do not contain ISO C constructs. Though much beloved by K&R C fans, this option is deprecated and may disappear in a future release of GCC.

■ **Tip** When using this option, you may also want to specify the `-fno-builtin` option if your application implements functions with the same names as built-in GCC functions.

Some of the backward-compatibility features activated by this option are the following:

- Older function call sequence is acceptable; parameter types can be defined outside the parentheses that delimit the parameters but before the initial bracket for the function body.
- All automatic variables not declared using the `register` keyword are preserved by the `longjmp` function. In ISO C, any automatic variables not declared as volatile have undefined values after a return.
- All `extern` declarations are global even if they occur inside a function definition. This includes implicit function declarations.
- Newer keywords such as `typeof`, `inline`, `signed`, `const`, and `volatile` are not recognized. Alternative keywords such as `__typeof__`, `__inline__`, and so on, can still be used.
- Comparisons between pointers and integers are always allowed.
- Integer types `unsigned short` and `unsigned char` are always promoted to `unsigned int`.
- Floating-point literals that are out of range for that datatype are not an error.
- String constants are stored in writable space and are therefore not necessarily constant.
- The character escape sequences `\x` and `\a` evaluate as the literal characters `x` and `a`, rather than being a prefix for the hexadecimal representation of a character and a bell, respectively.

`-traditional-cpp`: This option when compiling C applications causes GCC to modify the behavior of its internal preprocessor to make it more similar to the behavior of traditional C preprocessors. See the GNU CPP manual for details (`http://gcc.gnu.org/onlinedocs/cpp`).

`-trigraphs`: This option while compiling a C application causes GCC to support ISO C trigraphs. The character set used in C source code is the 7 bit ASCII character set, which is a superset of a superset of the ISO 646-1983 Invariant Code Set. Trigraphs are sequences of three characters (introduced by two question marks) that the compiler replaces with their corresponding punctuation characters. Specifying the `-trigraphs` option enables C source files to be written using only characters in the ISO Invariant Code Set by providing an ISO-compliant way of representing punctuation or international characters for which there is no convenient graphical representation on the development system. The `-ansi` option implies the `-trigraphs` option because it enforces strict conformance to the ISO C standard.

The nine standard trigraphs and their replacements are the following:

```
Trigraph:       ??(  ??)      ??<   ??>        ??=      ??/   ??'  ??!     ??-
Replacement:    [     ]       {       }        #        \      ^     |      ~
```

`-U`*`NAME`*: This preprocessor option causes cpp to cancel any previous definition of `NAME`, regardless of whether it is built in or explicitly defined using the `-D` option.

`-undef`: This preprocessor option causes cpp not to predefine any system-specific macros. Standard system-independent macros are still defined.

`-V` *`VERSION`*: This target-related option causes GCC to attempt to run the specified version of GCC if multiple versions are installed on your system. Using this option can be handy if you have multiple versions of GCC installed on your system.

`-v`: This output option causes GCC to print the commands executed during each stage of compilation, along with the version number of each command.

`--version`: This output option tells GCC to display version and build information, and then exit.

`-W`: This diagnostic/warning option is a deprecated synonym for `-Wextra` in GCC 4.x.

`-Wa,option`: This causes GCC to pass `option` as an option to the assembler. If `option` contains commas, each comma is interpreted as a separator for multiple options.

`-Wabi`: This option when compiling C++ applications causes GCC to display a warning when it generates code that may not be compatible with the generic C++ ABI. Eliminating warnings of this type typically requires that you modify your code. The most common causes of these warnings are padding related, either when using bit fields or when making assumptions about the length of words or data structures.

`-Waggregate-return`: This diagnostic/warning option elicits a warning if any functions that return structures or unions are defined or called. (It also elicits a warning in languages where you can return an array.) This option is not implied by specifying the `-Wall` option.

`-Wall`: This diagnostic/warning option activates the majority of GCC's warnings.

■ **Note** Though you would assume that using GCC's `-Wall` option turns on all warnings, that is not the case. The following warning-related options are not automatically activated when you specify `-Wall` and must be separately specified if desired: `-W`, `-Waggregate-return`, `-Wbad-function-cast`, `-Wcast-align`, `-Wcast-qual`, `-Wconversion`, `-Wdisabled-optimization`, `-Werror`, `-Wfloat-equal`, `-Wformat-nonliteral`, `-Wformat-security`, `-Wformat=2`, `-Winline`, `-Wlarger-than-len`, `-Wlong-long`, `-Wmissing-declarations`, `-Wmissing-format-attribute`, `-Wmissing-noreturn`, `-Wmissing-prototypes`, `-Wnested-externs`, `-Wno-deprecated-declarations`, `-Wno-format-y2k`, `-Wno-format-extra-args`, `-Wpacked`, `-Wpadded`, `-Wpointer-arith`, `-Wredundant-decls`, `-Wshadow`, `-Wsign-compare`, `-Wstrict-prototypes`, `-Wtraditional`, `-Wundef`, and `-Wunreachable-code`. See the explanation of these options in this appendix for details on exactly what additional warnings they will generate.

`-Walways-true`: This warning option tells GCC to generate warnings about tests that are always true due to inherent attributes of the associated datatypes. This warning is enabled by `-Wall`.

`-Wassign-intercept`: This Objective C and Objective C++ warning option tells GCC to issue a warning whenever an Objective C assignment is intercepted by the garbage collector.

`-Wbad-function-cast`: This diagnostic/warning option when compiling a C application causes GCC to display a warning message when a function call is cast to a type that does not match the function declaration. Though this is something that many C programmers traditionally do, using this option can be useful to help you detect casts to datatypes of different sizes.

`-Wc++-compat`: This option tells GCC to issue a warning about any ISO C constructs used that are outside the common subset shared by ISO C and ISO C++.

`-Wcast-align`: This diagnostic/warning option causes GCC to display a warning message whenever a pointer is cast such that the target will need to change the alignment of the specified data structure. For example, casting a `char *` to an `int *` on machines where integers can only be accessed at 2- or 4-byte boundaries would generate this warning.

`-Wcast-qual`: This diagnostic/warning option causes GCC to display a warning message whenever casting a pointer removes a type qualifier from the target type. For example, casting a `const char *` to an ordinary `char *` would generate this warning.

`-Wchar-subscripts`: This diagnostic/warning option causes GCC to display a warning message whenever an array subscript has type `char`.

`-Wcomment`: This diagnostic/warning option causes GCC to display a warning message whenever a comment-start sequence (`/*`) appears within another (`/*`) comment or whenever a newline is escaped within a `//` comment. This is an incredibly helpful option to detect most cases of "comment overflow."

`-Wconversion`: This diagnostic/warning option causes GCC to display a warning message whenever a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This helps identify conversions that would change the width or signedness of a variable, which is a common cause of error on machines with alignment requirements. This option will not generate warnings for explicit casts such as (`unsigned`) `-1` because these will be preresolved during compilation.

`-Wctor-dtor-privacy`: This diagnostic/warning option when compiling a C++ application causes GCC to display a warning whenever a class seems unusable, because all the constructors or destructors are private and the class has no friends (i.e., it grants no access to other classes or functions) or public static member functions.

`-Wdeclaration-after-statement`: This C language warning option tells GCC to issue a warning whenever a declaration follows a statement in a basic block. This is not supported by ISO C90 or versions of GCC prior to 3.0, but was introduced in C99 and is now allowed by GCC.

`-Wdisabled-optimization`: This diagnostic/warning option causes GCC to display a warning message whenever a requested optimization pass is disabled or skipped. This rarely indicates a problem with your GCC installation or code but instead usually means that GCC's optimizers are simply unable to handle the code because of its size, complexity, or the amount of time that the requested optimization pass would require.

`-Wdiv-by-zero`: This diagnostic/warning option causes GCC to display a warning message whenever the compiler detects and attempts to divide an integer by zero. You can disable this warning by using the `-Wno-div-by-zero` option. This warning is not generated when an application attempts floating-point division by zero, because this can occasionally be used in applications to generate values for infinity and NaN.

`-Weffc++`: This diagnostic/warning option, when compiling a C++ application, causes GCC to display a warning message whenever application code violates various guidelines described in Scott Meyers' *Effective C++* (Addison-Wesley, 2005. ISBN: 0-321-33487-6) and *More Effective C++* (Addison-Wesley, 1995. ISBN: 0-201-63371-X), such as the following:

- Define a copy constructor and an assignment operator for classes with dynamically allocated memory (Item 11, *Effective C++*).
- Prefer initialization to assignment in constructors (Item 12, *Effective C++*).
- Make destructors virtual in base classes (Item 14, *Effective C++*).
- Have `operator=` return a reference to `*this` (Item 15, *Effective C++*).
- Do not try to return a reference when you must return an object (Item 23, *Effective C++*).
- Distinguish between prefix and postfix forms of increment and decrement operators (Item 6, *More Effective C++*).
- Never overload the operators &&, ||, or , (Item 7, *More Effective C++*).

■ **Note** Ironically, some of the standard header files used by GCC do not follow these guidelines, so activating this warning option may generate unexpected messages about system files. You can ignore warnings from outside your code base or use a utility such as `grep -v` to filter out those warnings.

`-Werror`: This diagnostic/warning option causes GCC to make all warnings into errors.

`-Werror-implicit-function-declaration`: This diagnostic/warning option causes GCC to display a warning message whenever a function is used before it has been declared.

`-Wextra`: This diagnostic/warning option causes GCC to display extra warning messages when it detects any of the following events in the code that is being compiled:

- A function can return either with or without a value. If a function returns a value in one case, both a return statement with no value (such as `return;`) or an implicit return after reaching the end of a function will trigger a warning.
- The left side of a comma expression has no side effects. (A *comma expression* is an expression that contains two operands separated by a comma. Although GCC evaluates both operands, the value of the expression is the value of the right operand. The left operand of a comma expression is used to do an assignment or produce other side effects—if it produces a value, it is discarded by GCC.) To suppress the warning, cast the left-side expression to `void`.
- An unsigned value is compared against zero using `<` or `<=`.
- A comparison such as `x <= y <= z` appears. GCC interprets this as `((x <= y) < z)`, which compares the return value of the comparison of *x* and *y* against the value of *z*, which is usually not what is intended.
- Storage-class specifiers such as `static` are not the first things in a declaration, which is suggested by modern C standards.
- A return type of a function has a type qualifier. This has no effect because the return value of a function is not an assigned lvalue.
- Unused arguments to a function call are present. This warning will only be displayed if the `-Wall` or `-Wunused` options are also specified.
- A comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. To eliminate this warning, specify the `-Wno-sign-compare` command-line option.
- An aggregate has a partly bracketed initializer, which is usually seen when initializing data structures that contain other data structures. The values passed to an internal data structure must also be enclosed within brackets.
- An aggregate has an initializer that does not initialize all members of the structure.

`-Wfatal-errors`: This warning option tells GCC to abort compilation whenever any error occurs, rather than trying to continue.

`-Wfloat-equal`: This diagnostic/warning option causes GCC to display a warning message whenever floating-point values are compared for equality. Because floating-point values are often used as approximations for infinitely precise real numbers, it is not always possible to precisely compare such approximation. If you are doing this, a better suggestion is to compare floating-point values by determining whether they fall within an acceptable range of values by using relational operators.

`-Wformat`: This diagnostic/warning option causes GCC to display a warning message whenever the arguments to calls to `printf`, `scanf`, `strftime` (X11), `strfmon` (X11), and similar functions do not have types appropriate to the specified format string. If the `-pedantic` option is used with this option, warnings will be generated for any use of format strings that are not consistent with the programming language standard being used.

`-Wformat=2`: This diagnostic/warning option is the same as explicitly invoking the `-Wformat`, `-Wformat-nonliteral`, and `-Wformat-security` options.

`-Wformat-nonliteral`: This diagnostic/warning option causes GCC to display a warning message whenever a format string is not a string literal and therefore cannot be checked, unless the format function takes its format arguments as a variable argument list (`va_list`).

`-Wformat-security`: This diagnostic/warning option causes GCC to display a warning message whenever calls to the `printf()` and `scanf()` functions use a format string that is not a string literal and there are no format arguments, as in `printf (foo);`. At the time this book was written, this option was a subset of the warnings generated by the `-Wformat-nonliteral` option but was provided to explicitly detect format strings that may be security holes.

`-Wformat-y2k`: This warning option when `-Wformat` is also specified tells GCC to also issue warnings about `strftime()` formats that may produce a two-digit year.

`-Wimplicit`: This diagnostic/warning option is the same as explicitly invoking the `-Wimplicit-int` and `-Wimplicit-function-declaration` options.

`-Wimplicit-function-declaration`: This diagnostic/warning option causes GCC to display a warning message whenever a function is used before being declared.

`-Wimplicit-int`: This diagnostic/warning option causes GCC to display a warning message whenever a declaration does not specify a type, which therefore causes the declared function or variable to default to being an integer.

`-Wimport`: This warning option tells GCC to issue a warning the first time that a #import directive is used.

`-Winit-self`: This C, C++, and Objective C warning option, when the `-Wuninitialized` option and optimization levels 1 and higher are being used, tells GCC to issue warnings about any uninitialized variables that are initialized by being set to themselves.

`-Winline`: This diagnostic/warning option causes GCC to display a warning message whenever a function that was declared as inline cannot be inlined.

`-Winvalid-pch`: This warning option tells GCC to issue a warning if a precompiled header is found in the search path but cannot be used.

`-Wl,option`: This causes GCC to pass `option` as an option to the linker. If `option` contains commas, each is interpreted as a separator for multiple options.

`-Wlarger-than-len`: This diagnostic/warning option causes GCC to display a warning message whenever an object of larger than `len` bytes is defined.

`-Wlong-long`: This diagnostic/warning option causes GCC to display a warning message whenever the `long long` type is used. This warning option is automatically enabled when the `-pedantic` option is specified. You can inhibit these warning messages in this case by using the `-Wno-long-long` option.

`-Wmain`: This diagnostic/warning option causes GCC to display a warning message whenever the type of `main()` or the number of arguments passed to it is suspicious. A program's main routine should always be an externally linked function that returns an integer value and takes either zero, two, or three arguments of the appropriate types.

`-Wmissing-braces`: This diagnostic/warning option causes GCC to display a warning message whenever an aggregate or union initializer is not correctly bracketed so that it explicitly follows the conventions of the aggregate or union. As an example, the following expression would generate this warning:

```
int a[2][2] = { 0, 1, 2, 3 };
```

`-Wmissing-declarations`: This diagnostic/warning option causes GCC to display a warning message whenever a global function is defined without a previous declaration, even if the definition itself provides a prototype. Using this option detects global functions that are not declared in header files.

`-Wmissing-field-initializers`: This warning option tells GCC to issue a warning if a structure initializer does not initialize all of the fields in the structure.

`-Wmissing-format-attribute`: This diagnostic/warning option for C programs causes GCC to display a warning message whenever a function such as `printf()` or `scanf()` contains a format string that contains more attributes than are provided in subsequent arguments to the call. If the `-Wformat` option is also specified, GCC will also generate warnings about similar occurrences in other functions that appear to take format strings.

`-Wmissing-include-dirs`: This C, C++, and Objective C warning option tells GCC to issue a warning if a user-specified include directory does not exist.

`-Wmissing-noreturn`: This diagnostic/warning option causes GCC to display a warning message whenever functions are used that might be candidates for the `noreturn` attribute `(__attribute__((noreturn)) prototype;)`.

`-Wmissing-prototypes`: This diagnostic/warning option causes GCC to display a warning message whenever compiling a C application in which a global function is defined without a previous prototype declaration and is intended to detect global functions that are not declared in header files. This warning is issued even if the definition itself provides a prototype.

`-Wmultichar`: This diagnostic/warning option causes GCC to display a warning message whenever a multicharacter constant (e.g., foo) is used. This option is enabled by default, but can be disabled by specifying the `-Wno-multichar` option. Multicharacter constants should not be used in portable code because their internal representation is platform-specific.

`-Wnested-externs`: This diagnostic/warning option when compiling a C application causes GCC to display a warning message whenever an `extern` declaration is encountered within a function.

`-Wno-deprecated`: This diagnostic/warning option causes GCC not to display a warning message whenever deprecated features are used.

`-Wno-deprecated-declarations`: This diagnostic/warning option causes GCC to not display a warning message whenever functions, variables, and types marked as deprecated (through the `deprecated` attribute) are used.

`-Wno-div-by-zero`: This warning option tells GCC to suppress compile-time warnings about integer division by zero.

`-Wno-endif-labels`: This warning option tells GCC to suppress warnings whenever `#else` or `#endif` statements are followed by additional text.

`-Wno-format-y2k`: This diagnostic/warning option causes GCC not to display a warning message whenever `strftime()` formats are used that may yield only a two-digit year.

`-Wno-format-extra-args`: This diagnostic/warning option when `-Wformat` is also specified causes GCC to not display a warning message whenever excess arguments are supplied to a `printf()` or `scanf()` function. Extra arguments are ignored, as specified in the C standard. Warnings will still be displayed if the unused arguments are not all pointers and lie between used arguments that are specified with $ operand number specifications.

`-Wno-import`: This diagnostic/warning option causes GCC to not display a warning message whenever `#import` statements are encountered in an Objective C application. (The `#import` statement is identical to C's `#include` statement, but will not include the same include file multiple times.)

`-Wno-int-to-pointer-cast`: This C language warning option tells GCC to suppress warnings when integers of one size are cast to pointers of another.

`-Wno-invalid-offsetof`: This C++ warning option tells GCC to suppress warnings from applying the `offsetof()` macro to a non-POD (plain old data) type, which is undefined according to the 1998 ISO C++ standard.

`-Wno-multichar`: This warning option tells GCC to suppress warnings if multicharacter constants are used.

`-Wno-non-template-friend`: This diagnostic/warning option when compiling a C++ application causes GCC not to display a warning message whenever nontemplatized friend functions are declared within a template. The C++ language specification requires that friends with unqualified IDs declare or define an ordinary, nontemplate function. Because unqualified IDs could be interpreted as a particular specialization of a templatized function in earlier versions of GCC, GCC now checks for instances of this in C++ code by using the `-Wnon-template-friend` option as a default. The `-Wno-non-template-friend` option can be used to disable this check but keep the conformant compiler code.

`-Wno-pmf-conversions`: This diagnostic/warning option causes GCC not to display a warning message whenever C++ disables the diagnostic for converting a bound pointer of a member function to a plain pointer.

`-Wno-pointer-to-int-cast`: This C language warning option tells GCC to suppress warnings whenever an integer of one size is cast to a pointer type of another size.

`-Wno-pragmas`: This warning option tells GCC to suppress warnings related to misuse of pragmas, including invalid syntax, incorrect parameters, or conflicts with other pragmas.

`-Wno-protocol`: This diagnostic/warning option when compiling an Objective C application causes GCC to not display a warning message if methods required by a protocol are not implemented in the class that adopts it.

`-Wno-return-type`: This diagnostic/warning option causes GCC to suppress warning messages whenever a function is defined with a return type that defaults to `int`, or when a return without a value is encountered in a function whose return type is not `void`. This option does not suppress warning messages when compiling C++ applications that contain nonsystem, nonmain functions without a return type.

`-Wno-sign-compare`: This diagnostic/warning option causes GCC to suppress displaying a warning message whenever a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.

`-Wnon-template-friend`: This option causes GCC to check for unqualified IDs that could be interpreted as a particular specialization of a templatized function when compiling C++ applications under earlier versions of GCC and display an error message. The C++ language specification requires that friends with unqualified IDs declare or define an ordinary, nontemplate function. This option is enabled by default.

`-Wnon-virtual-dtor`: This diagnostic/warning option when compiling a C++ application causes GCC to not display a warning message whenever a class declares a nonvirtual destructor that should probably be virtual because the class may be used polymorphically.

`-Wnonnull`: This option tells GCC to issue a warning whenever a null pointer is passed as an argument that is marked as requiring a nonnull value by the nonnull attribute.

`-Wold-style-cast`: This diagnostic/warning option when compiling a C++ application causes GCC to display a warning message whenever a `C-stylestyle` cast to a non-`void` type is used within a C++ program. Newer cast statements such as `const_cast`, `reinterpret_cast`, and `static_cast` should be used instead, because they are less vulnerable to unintended side effects.

`-Wold-style-definition`: This C language warning option tells GCC to issue a warning whenever an old-style function definition is encountered.

`-Woverlength-strings`: This warning option tells GCC to issue a warning about strings that are longer than the specified maximum in a given C standard. In ISO C89 the limit is 509 characters. In ISO C99 the limit is 4,095 characters. This option is enabled by the `-pedantic` option.

`-Woverloaded-virtual`: This diagnostic/warning option when compiling a C++ application causes GCC to display a warning message whenever a function declaration hides virtual functions from a base class, typically because a virtual function with the same name is already present in a base class.

`-Wp,option`: This causes GCC to pass `option` as an option to the preprocessor. If `option` contains commas, each comma is interpreted as a separator for multiple options.

`-Wpacked`: This diagnostic/warning option causes GCC to display a warning message whenever a structure is specified as *packed*, but the `packed` attribute has no effect on the layout or size of the structure.

`-Wpadded`: This diagnostic/warning option causes GCC to display a warning message whenever padding is included in a data structure, regardless of whether it is used to align a single element or the entire data structure. This warning is displayed because it is often possible to reduce the size of the structure simply by rearranging its components.

`-Wparentheses`: This diagnostic/warning option causes GCC to display a warning message whenever parentheses are omitted in certain contexts, such as when there is an assignment in a context where a truth value is expected, when operators are nested whose precedence is commonly confused, or when there may be confusion about the `if` statement to which an `else` branch belongs. Warnings caused by the latter two problems can easily be corrected by adding brackets to explicitly identify nesting.

`-Wpointer-arith`: This diagnostic/warning option causes GCC to display a warning message whenever anything depends on the size of a function type or of `void`. GNU C assigns these types a size of 1 for convenience in calculations and pointer comparisons.

`-Wpointer-sign`: This C and Objective C language warning option tells GCC to issue a warning when assigning or passing pointers as arguments where source and destination have different signedness. This option is enabled by the `-Wall` or `-pedantic` options.

`-Wredundant-decls`: This diagnostic/warning option causes GCC to display a warning message whenever anything is declared more than once in the same scope.

`-Wreorder`: This diagnostic/warning option when compiling a C++ application causes GCC to display a warning message whenever the order of member initializers given in the code does not match the order in which they were declared. A warning is displayed to inform you that GCC is reordering member initializers to match the declaration.

`-Wreturn-type`: This diagnostic/warning option causes GCC to display a warning message whenever a function is defined with a return type that defaults to `int`, or when a return without a value is encountered in a function whose return type is not `void`. When compiling C++ applications, nonsystem functions without a return type (and which are not `main()`) always produce this error message, even when the `-Wno-return-type` option is encountered. This option is active by default.

`-Wselector`: This diagnostic/warning option when compiling an Objective C application causes GCC to display a warning message whenever a selector defines multiple methods of different types.

`-Wsequence-point`: This diagnostic/warning option when compiling a C application causes GCC to display a warning message whenever the compiler detects code that may have undefined semantics because of violations of sequence-point rules in the C standard. Sequence-point rules help the compiler order the execution of different parts of the program and can be violated by code sequences with undefined behavior such as `a = a++`, `a[n] = b[n++]`, and `a[i++] = i;`. Some more complicated cases may not be identified by this option, which may also occasionally give a false positive.

`-Wshadow`: This diagnostic/warning option causes GCC to display a warning message whenever a local variable shadows another local variable, parameter, or global variable, or whenever a built-in function is shadowed.

`-Wsign-compare`: This diagnostic/warning option causes GCC to display a warning message whenever a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. This option is automatically invoked when you specify the `-W` option.

`-Wsign-promo`: This diagnostic/warning option when compiling a C++ application causes GCC to display a warning message whenever overload resolution chooses a promotion from an unsigned or enumerated type to a signed type over a conversion to an unsigned type of the same size. Earlier versions of GCC would try to preserve unsignedness.

`-Wstack-protector`: This option when `-fstack-protect` is also specified tells GCC to issue a warning about any functions that will not be protected.

`-Wstrict-aliasing`: This option tells GCC to issue warnings for any code that might break the strict aliasing rules that the compiler is using as a basis for optimization. This option is enabled by the `-Wall` option.

`-Wstrict-aliasing=2`: This warning option when `-fstrict-warning` is also specified tells GCC to perform stricter checks for aliasing violations than the `-Wstrict-aliasing` option. This produces more warnings but may generate warnings about some cases that are actually safe.

`-Wstrict-null-sentinel`: This C++ language option tells GCC to issue a warning whenever an uncast `NULL` is being used as a sentinel, which is not portable across different C++ compilers.

`-Wstrict-prototypes`: This diagnostic/warning option when compiling a C application causes GCC to display a warning message whenever a function is declared or defined without specifying the types of its arguments.

`-Wswitch`: This diagnostic/warning option is the same as specifying both the `–Wswitch-default` and the `–Wswitch-enum` warning options.

`-Wswitch-default`: This warning option tells GCC to issue a warning whenever a switch statement does not have a default case.

`-Wswitch-enum`: This warning option tells GCC to issue a warning whenever a switch statement has an index of an enumerated type but lacks a case statement for one or more possible values of that type.

`-Wsynth`: This diagnostic/warning option when compiling a C++ application causes GCC to display a warning message whenever GCC's synthesis behavior does not match that of Cfront. Cfront is a family of C++ compilers from many different vendors that serves as a front end for C++ code, translating it so that it can subsequently be compiled by a standard C compiler.

`-Wsystem-headers`: This diagnostic/warning option causes GCC to display a warning message whenever potentially invalid constructs are found in system header files. Using this command-line option tells GCC to display warnings about system headers as if they occurred in application code. To display warnings about unknown pragmas found in system headers, you must also specify the `-Wunknown-pragmas` option.

`-Wtraditional`: This diagnostic/warning option when compiling a C application causes GCC to display a warning message whenever constructs are encountered that behave differently in traditional and ISO C or are only found in ISO C, and for problematic constructs that should generally be avoided. Some examples of these are the following:

- Macro parameters that appear within string literals in the macro body. Traditional C supports macro replacement within string literals, but ISO C does not.
- Preprocessor directives that do not begin with the hash symbol (`#`) as the first character on a line. Preprocessor directives such as `#pragma` that are not supported by traditional C can thus be "hidden" by indenting them. For true portability, you may want to generally avoid preprocessor directives such as `#elif` that are not supported by traditional C.
- Functionlike macros that appear without arguments.
- The `U` integer constant suffix, or the `F` or `L` floating-point constant suffixes.
- Functions that are declared external in one block and subsequently used after the end of the block.
- A switch statement that has an operand of type `long`.
- Nonstatic function declarations that follow static ones.
- Integer constants. The ISO type of a decimal integer constant has a different width or signedness from its traditional type.
- ISO string concatenation.
- Initialization of automatic aggregates, which are nonstatic local arrays and structures.
- Conflicts between identifiers and labels.
- Union initialization of nonzero unions.
- Prototype conversions between fixed and floating-point values and vice versa. You can use the `-Wconversion` option to display additional warnings related to possible conversion problems.

`-Wtrigraphs`: This diagnostic/warning option causes GCC to display a warning message whenever trigraphs without comments are encountered that might change the meaning of the program.

`-Wundeclared-selector`: This Objective C and Objective C++ language option tells GCC to issue a warning if an `@selector()` expression referring to an undeclared selector is encountered. This option checks for this condition whenever an `@selector()` expression is encountered, unlike the similar `-Wselector` option, which checks for this condition in the final stages of compilation.

`-Wundef`: This diagnostic/warning option causes GCC to display a warning message whenever an undefined identifier is evaluated in a `#if` preprocessor directive.

`-Wuninitialized`: This diagnostic/warning option causes GCC to display a warning message whenever an automatic variable is used without first being initialized, or if an existing nonvolatile variable may be changed by a `setjmp` call. These warnings are only generated when using the `-O`, `-O1`, `-O2`, or `-O3` optimization options, and then only for nonvolatile variables that are candidates for register allocation.

`-Wunknown-pragmas`: This diagnostic/warning option causes GCC to display a warning message whenever it encounters an unknown `#pragma` preprocessor directive.

`-Wunreachable-code`: This diagnostic/warning option causes GCC to display a warning message whenever GCC detects code that will never be executed.

`-Wunsafe-loop-optimizations`: This warning option tells GCC to issue a warning when a loop cannot be optimized because the compiler cannot make assumptions about the bounds of loop indices.

`-Wunused`: This diagnostic/warning option provides a convenient shortcut for specifying all of the `-Wunused-function`, `-Wunused-label`, `-Wunused-value`, and `-Wunused-variable` options. In order to get a warning about an unused function parameter, you must either specify the `-W` and `-Wunused` options or separately specify the `-Wunused-parameter` option.

`-Wunused-function`: This diagnostic/warning option causes GCC to display a warning message whenever a static function is declared but not defined, or when a noninline static function is not used.

`-Wunused-label`: This diagnostic/warning option causes GCC to display a warning message whenever a label is declared but not used.

`-Wunused-parameter`: This diagnostic/warning option causes GCC to display a warning message whenever a function parameter is unused aside from its declaration.

`-Wunused-value`: This diagnostic/warning option causes GCC to display a warning message whenever a statement computes a result that is not used.

`-Wunused-variable`: This diagnostic/warning option causes GCC to display a warning message whenever a local variable or nonconstant static variable is unused aside from its declaration.

`-Wvariadic-macros`: This option tells GCC to issue a warning if variadic macros are used in pedantic ISO C90 mode, or if alternate GNU syntax for these is used in pedantic C99 mode. This option is active by default.

`-Wvolatile-register-var`: This warning option tells GCC to issue a warning if a register variable is declared as volatile.

`-Wwrite-strings`: This C and C++ language warning option tells GCC to impose a type of `const char[LENGTH]` on string constants and to issue a warning when copying the address of one into a non-`const char` pointer. This option is enabled by default for C++ programs.

`-w`: This diagnostic/warning option causes GCC not to display any warning messages.

`-Xassembler` *`option`*: This causes GCC to pass `option` as an option to the assembler, and is often used to supply system-specific assembler options that GCC does not recognize. Each option is a single token. If you need to pass an option that takes an argument, you must use the `–Xassembler` option twice, once for the option and once for the argument.

`-Xlinker` *`option`*: This causes GCC to pass `option` as an option to the linker, and is often used to supply system-specific linker options that GCC does not recognize. Each option is a single token. If you need to pass an option that takes an argument, you must use the `–Xlinker` option twice, once for the option and once for the argument.

`-Xpreprocessor` *`option`*: This causes GCC to pass `option` as an option to the preprocessor, and is often used to supply system-specific preprocessor options that GCC does not recognize. Each option is a single token. If you need to pass an option that takes an argument, you must use the `–Xpreprocessor` option twice, once for the option and once for the argument.

`-x [`*`language`*`|none]`: This output option identifies the output *language* to be generated rather than letting the compiler choose a default based on the extension of the input file. This option applies to all following input files until the next `-x` option. Possible values for *language* are `ada`, `assembler`, `assembler-with-cpp`, `c`, `c-header`, `c++`, `c++-cpp-output`, `cpp-output`, `f77`, `f77-cpp-output`, `java`, `objc-cpp-output`, `objective-c`, and `ratfor`. Specifying *none* turns off the language specification, reverting to GCC's defaults based on the extension of the input file.

APPENDIX B

Machine- and Processor-Specific Options for GCC

GCC provides hundreds of machine-specific options that you will rarely need to use unless you are compiling for a specific platform and need to take advantage of some of its unique characteristics. This appendix provides a summary and a discussion of machine-specific options for GCC, organized by the platform to which they are relevant. For your convenience if you are using versions of GCC other than 4.x, this appendix includes machine- and processor-specific options for some older targets that are no longer supported in the GCC 4.x compiler family. Where obsolete platform options are discussed, a note identifies the GCC versions to which they are relevant.

Machine- and architecture-specific configuration information for GCC is stored in the `gcc/config` subdirectory of a GCC source code installation. In theory, each supported system has its own directory that contains general configuration information as well as specific information for supported variants of that processor or architecture. The one exception to this rule is Darwin support, which lives in the `gcc/config` subdirectory because it is an OS kernel and execution environment rather than an architecture (and actually spans multiple architectures). It is treated as a machine in this appendix because there are a large number of Darwin-specific GCC options available, thanks to the fact that it is open source, it runs on both x86 and PPC architectures, and Apple depends on it for the lovely Mac OS X.

The majority of the machine- and CPU-specific options available in GCC are specific values for GCC's `-m` command-line option, which enables you to identify characteristics of the machine for which GCC is generating code.

Note The options discussed in this section are only relevant if you are using a version of GCC that was either compiled to run directly on the specified platform (not always possible) or if you are using a version of GCC that has been built as a cross-compiler, generating binaries for the specified platform even though it is actually executing on another platform. For more information about cross-compilers, see Chapter 14.

Alpha Options

The 64-bit Alpha processor family was originally developed by Digital Equipment Corporation (DEC) and inherited by its purchasers, Compaq Computer and (later) Hewlett-Packard. The Alpha was an extremely fast processor for its time whose widespread adoption was hampered by VMS and DEC's series of one-night stands with a variety of flavors of Unix.

GCC options available when compiling code for Unix-like operating systems running on the DEC Alpha family of processors are the following:

`-malpha-as`: Specifying this option tells GCC to generate code that is to be assembled by the DEC assembler.

`-mbuild-constants`: Specifying this option causes GCC to construct all integer constants using code that checks to see if the program can construct the constant from smaller constants in two or three instructions. If it cannot, GCC outputs the constant as a literal and generates code to load it from the data segment at runtime. Normally, GCC only performs this check on 32- or 64-bit integer constants. The goal of this option is to keep constants out of the data segment and in the code segment whenever possible. This option is typically used when building a dynamic loader for shared libraries, because the loader must be able to relocate itself before it can locate its data segment.

`-mbwx`: Specifying this option tells GCC to generate code to use the optional BWX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mcix`: Specifying this option tells GCC to generate code to use the optional CIX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mcpu=CPU-type`: Specifying this option tells GCC to use the instruction set and instruction scheduling parameters that are associated with the machine type `CPU-type`. Instruction scheduling is the phase of compilation that sequences instructions in order to maximize possible parallelism and minimize any time that instructions spend waiting for results of other instructions. You can specify either the chip name (EV-style name) or the corresponding chip number. If you do not specify a processor type, GCC will default to the processor on which the GCC was built. Supported values for `CPU-type` are the following:

- `ev4` | `ev45` | `21064`: Schedule as an EV4 without instruction set extensions
- `ev5` | `21164`: Schedule as an EV5 without instruction set extensions
- `ev56` | `2164a`: Schedule as an EV5 and support the BWX instruction set extension
- `pca56` | `21164pc` | `21164PC`: Schedule as an EV5 and support the BWX and MAX instruction set extensions
- `ev6` | `21264`: Schedule as an EV6 and support the BWX, FIX, and MAX instruction set extensions
- `ev67` | `21264a`: Schedule as an EV6 and support the BWX, CIX, FIX, and MAX instruction set extensions

`-mexplicit-relocs`: Specifying this option tells GCC to generate code that explicitly marks which type of symbol relocation should apply to which instructions. See the discussion of the `-msmall-data` and `–mlarge-data` options for additional information related to symbol relocation when the `-explicit-relocs` option is specified. This option is essentially a workaround for older assemblers that could only do relocation by using macros.

`-mfix`: Specifying this option tells GCC to generate code to use the optional FIX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mfloat-ieee`: Specifying this option tells GCC to generate code that uses IEEE single and double precision instead of VAX F and G floating-point arithmetic.

`-mfloat-vax`: Specifying this option tells GCC to generate code that uses VAX F and G floating-point arithmetic instead of IEEE single and double precision.

`-mfp-regs`: Specifying this option causes GCC to generate code that uses the floating-point register (FPR) set. This is the default.

`-mfp-rounding-mode=`*`rounding-mode`*: Using this option enables you to specify the IEEE rounding mode used to round off floating-point numbers. The *`rounding-mode`* must be one of the following:

- `c`: Chopped rounding mode. Floating-point numbers are rounded toward zero.
- `d`: Dynamic rounding mode. A field in the floating-point control register (FPCR) (see the reference manual for the Alpha architecture and processors) controls the rounding mode that is currently in effect. The C library initializes this register for rounding toward plus infinity. Unless your program modifies this register, a `rounding-mode` of `d` means that floating-point numbers are rounded toward plus infinity.
- `m`: Round toward minus infinity.
- `n`: Normal IEEE rounding mode. Floating-point numbers are rounded toward the nearest number that can be represented on the machine, or to the nearest even number that can be represented on the machine if there is no single nearest number.

`-mfp-trap-mode=`*`trap-mode`*: Specifying this option controls what floating-point related traps are enabled. *`trap-mode`* can be set to any of the following values:

- `n`: Normal. The only traps that are enabled are the ones that cannot be disabled in software, such as the trap for division by zero. This is the default setting.
- `su`: Safe underflow. This option is similar to `u trap-mode`, but marks instructions as safe for software completion. (See the Alpha architecture manual for details.)
- `sui`: Safe underflow inexact. This enables inexact traps as well as the traps enabled by `su trap-mode`.
- `u`: Underflow. This enables underflow traps as well as the traps enabled by the normal `trap-mode`.

`-mgas`: Specifying this option tells GCC to generate code that is to be assembled by the GNU assembler.

`-mieee`: Specifying this option causes GCC to generate fully IEEE-compliant code except that the `INEXACT-FLAG` is not maintained. The Alpha architecture provides floating-point hardware that is optimized for maximum performance and is generally compliant with the IEEE floating-point standard. This GCC option makes generated code fully compliant.

When using this option, the preprocessor macro `_IEEE_FP` is defined during compilation. The resulting code is less efficient but is able to correctly support denormalized numbers and exceptional IEEE values such as Not-a-Number and plus/minus infinity.

`-mieee-conformant`: Specifying this option tells GCC to mark the generated code as being IEEE conformant. You must not use this option unless you also specify `-mtrap-precision=i` and either `-mfp-trap-mode=su` or `-mfp-trap-mode=sui`. Specifying this option inserts the line `.eflag 48` in the function prologue of the generated assembly file. Under DEC Unix, this line tells GCC to link in the IEEE-conformant math library routines.

`-mieee-with-inexact`: Specifying this option tells GCC to generate fully IEEE-compliant code and to maintain the status of the `INEXACT-FLAG`. In addition to the preprocessor macro `_IEEE_FP`, the macro `_IEEE-FP-EXACT` is also defined during compilation. Because little code depends on the `EXACT-FLAG`, this option is rarely used. Code compiled with this option may execute more slowly than code generated by default.

`-mlarge-data`: Specifying this option causes GCC to store all data objects in the program's standard data segment. This increases the possible size of the data area to just below 2GB, but may require additional instructions to access data objects. Programs that require more than 2GB of data must use `malloc` or `mmap` to allocate the data in the heap instead of in the program's data segment.

■ **Note** When generating code for shared libraries on an Alpha, specifying the `-fPIC` option implies the `-mlarge-data` option.

`-mlarge-text`: Specifying this option causes GCC not to make any assumptions about the final size of the code that it produces, increasing overall code size but providing more flexibility. This option is the default.

`-mlong-double-64`: Specifying this option tells GCC to use the default size of 64 bits for both the `long double` and `double` datatypes. This is the default.

`-mlong-double-128`: Specifying this option tells GCC to double the size of the `long double` datatype to 128 bits. By default, it is 64 bits and is therefore internally equivalent to the `double` datatype.

`-mmax`: Specifying this option tells GCC to generate code to use the optional MAX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mmemory-latency=`*`time`*: Specifying this option tells GCC how to set the latency that the scheduler should assume for typical memory references as seen by the application. The specified value supplied for *time* depends on the memory access patterns used by the application and the size of the external cache on the machine. Supported values for *time* are the following:

- `number`: A decimal number representing clock cycles.
- `L1` | `L2` | `L3` | `main`: Use internal estimates of the number of clock cycles for typical EV4 and EV5 hardware for the Level 1, 2, and 3 caches, as well as `main` memory. (The level 1, 2, and 3 caches are also often referred to as Dcache, Scache, and Bcache, respectively.) Note that L3 is only valid for EV5.

`-mno-bwx`: Specifying this option tells GCC not to generate code to use the optional BWX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mno-cix`: Specifying this option tells GCC not to generate code to use the optional CIX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mno-explicit-relocs`: Specifying this option tells GCC to generate code that follows the old Alpha model of generating symbol relocations through assembler macros. Use of these macros does not allow optimal instruction scheduling.

`-mno-fix`: Specifying this option tells GCC not to generate code to use the optional FIX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mno-fp-regs`: Specifying this option causes GCC to generate code that does not use the floating-point register set. When the floating-point register set is not used, floating-point operands are passed in integer registers as if they were integers and floating-point results are passed in `'$0'` instead of `'$f0'`. This is a nonstandard calling sequence, so any function with a floating-point argument or return value called by code compiled with `-mno-fp-regs` must also be compiled with that option. Specifying this option implies the `-msoft-float` option.

`-mno-max`: Specifying this option tells GCC not to generate code to use the optional MAX instruction set. The default is to use the instruction sets supported by a CPU type specified by using the `-mcpu=CPU-type` option, or the instruction sets supported on the CPU on which GCC was built if the `-mcpu=CPU-type` option was not specified.

`-mno-soft-float`: Specifying this option tells GCC to use hardware floating-point instructions for floating-point operations. This is the default.

`-msmall-data`: Specifying this option causes GCC to store objects that are 8 bytes long or smaller in a small data area (the sdata and sbss sections). These sections are accessed through 16-bit relocations based on the `$gp` register. This limits the size of the small data area to 64K, but allows variables to be directly accessed using a single instruction. This option can only be used when the `-explicit-relocs` option has also been specified.

■ **Note** When generating code for shared libraries on an Alpha, specifying the `-fpic` option implies the `-msmall-data` option.

`-msmall-text`: Specifying this option causes GCC to assume that the size of the code produced by the compiler is smaller than 4MB and that therefore any part of the resulting code can be reached by a single branch instruction. This reduces the number of instructions required to make any function call, thereby reducing overall code size.

`-msoft-float`: Specifying this option tells GCC to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC, but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system, or these function calls will be identified as unresolved.

■ **Note** Ironically, Alpha implementations without floating-point operations are still required to have floating-point registers.

`-mtls-kernel`: Specifying this option tells GCC that the system for which it is generating code uses the OSF/1 PAL code, and should therefore use the `rduniq` and `wruniq` calls for thread pointer built-ins rather than the `rdval` and `wrval` calls that are normally used.

`-mtls-size=`*`number`*: Specifying this option enables you to tell GCC the bit size of offsets used for thread-local storage, where *`number`* is that value.

`-mtrap-precision=`*`trap-precision`*: Floating-point traps are imprecise in the Alpha architecture; without software assistance, it is impossible to recover from a floating trap, and programs that trap must usually be terminated. Specifying this option causes GCC to generate code that can help operating system trap handlers determine the exact location that caused a floating-point trap. The following levels of precision can be specified as the value of *`trap-precision`*:

- `f`: Function precision. The trap handler can determine the function that caused a floating-point exception.
- `i`: Instruction precision. The trap handler can determine the exact instruction that caused a floating-point exception.
- `p`: Program precision. The trap handler can only identify which program caused a floating-point exception. This is the default.

`-mtune=`*`CPU-type`*: Specifying this option tells GCC to set only the instruction scheduling parameters based on the specified `CPU-type`. The instruction set is not changed. Possible values for *`CPU-type`* are the same as those that can be specified using the `-mcpu=`*`CPU-type`* option.

Alpha/VMS Options

Many workstations using Alpha processors from DEC (and later Compaq Computer or Hewlett-Packard) shipped with a quaint operating system called VMS. For years, VMS was the only operating system for DEC computers for which support was officially available from DEC.

The sole GCC option available when compiling code for Alpha systems running VMS is the following:

`-mvms-return-codes`: Specifying this option causes GCC to return VMS error codes from `main`, rather than the default POSIX-style error codes used by the majority of the known universe.

AMD x86-64 Options

The options in this section can only be used when compiling code targeted for 64-bit AMD processors (the GCC x86-64 build target). For the discussion of options that can only be used on i386 and AMD x86-64 systems, see the section of this appendix titled "i386 and AMD x86-64 Options." For the discussion of options that can only be used on IA-64 (64-bit Intel) systems, see the section of this appendix titled "IA-64 Options."

GCC options available when compiling code for 64-bit AMD systems are the following:

`-m32`: Specifying this option tells GCC to generate code for a 32-bit environment. The 32-bit environment sets `int`, `long`, and `pointer` to 32 bits and generates code that runs on any i386 system.

`-m64`: Specifying this option tells GCC to generate code for a 64-bit environment. The 64-bit environment sets `int` to 32 bits and `long` and `pointer` to 64 bits and generates code for AMD's x86-64 architecture.

`-mcmodel=kernel`: Specifying this option tells GCC to generate code for the kernel code model. The kernel runs in the negative 2GB of the address space. This model must be used for Linux kernel code.

`-mcmodel=large`: Specifying this option tells GCC to generate code for the large code model. This model makes no assumptions about addresses and sizes of sections. This option is reserved for future expansion—GCC does not currently implement this model.

`-mcmodel=medium`: Specifying this option tells GCC to generate code for the medium code model. This means that the program is linked in the lower 2GB of the address space, but symbols can be located anywhere in the address space. Programs can be statically or dynamically linked, but building shared libraries is not supported by this memory model.

`-mcmodel=small`: Specifying this option tells GCC to generate code for the small code model. This means that the program and its symbols must be linked in the lower 2GB of the address space, pointers are 64 bits, and programs can be statically or dynamically linked. This is the default code model.

`-mno-red-zone`: Specifying this option tells GCC not to use a so-called *red zone* for x86-64 code. The red zone is mandated by the x86-64 ABI, and is a 128-byte area beyond the location of the stack pointer that will not be modified by signal or interrupt handlers and can therefore be used for temporary data without adjusting the stack pointer. Using the red zone is enabled by default.

AMD 29K Options

The AMD 29000 processor is a RISC microprocessor descended from the Berkeley RISC design and includes a memory-management unit (MMU) as well as support for the AMD 29027 floating-point unit (FPU). Like conceptually similar processors such as the SPARC, the 29000 has a large register set split into local and global sets and provides sophisticated mechanisms for protecting, manipulating, and managing registers and their contents. The AMD 29000 family of processors includes the 29000, 29005, 29030, 29035, 29040, and 29050 microprocessors.

■ **Note** Support for this processor family was marked as obsolete in GCC 3.1 and was fully purged in GCC version 3.2.2. These options are therefore only of interest if you are using a version of GCC that is earlier than 3.2.1 and that you are certain provides support for this processor family.

GCC options available when compiling code for the AMD AM29000 family of processors are the following:

`-m29000`: Specifying this option causes GCC to generate code that only uses instructions available in the basic AMD 29000 instruction set. This is the default.

`-m29050`: Specifying this option causes GCC to generate code that takes advantage of specific instructions available on the AMD 29050 processor.

`-mbw`: Specifying this option causes GCC to generate code that assumes the system supports byte and half-word write operations. This is the default.

`-mdw`: Specifying this option causes GCC to generate code that assumes the processor's DW bit is set, which indicates that byte and half-word operations are directly supported by the hardware. This is the default.

`-mimpure-text`: Specifying this option in conjunction with the `-shared` option tells GCC not to pass the `-assert` pure-text option to the linker when linking a shared object.

`-mkernel-registers`: Specifying this option causes GCC to generate references to registers `gr64` through `gr95` instead of to registers `gr96` through `gr127` (the latter is the default). This option is often used when compiling kernel code that wants to reserve and use a set of global registers that are disjointed from the set used by user-mode code. Any register names passed as compilation options using GCC's `-f` option must therefore use the standard user-mode register names.

`-mlarge`: Specifying this option causes GCC to always use calli instructions, regardless of the size of the output file. You should use this option if you expect any single file to compile into more than 256K of code.

`-mnbw`: Specifying this option causes GCC to generate code that assumes the processor does not support byte and half-word operations. Specifying this option automatically sets the related `-mndw` option.

`-mndw`: Specifying this option causes GCC to generate code that assumes the processor's DW bit is not set, indicating that byte and half-word operations are not directly supported by the hardware. This option is automatically set if you specify the `-mnbw` option.

`-mno-impure-text`: Specifying this option tells GCC to pass the `-assert` pure-text option to the linker when linking a shared object.

`-mno-multm`: Specifying this option causes GCC not to generate multm or multmu instructions. This option is used when compiling for 29000-based embedded systems that do not have trap handlers for these instructions.

`-mno-reuse-arg-regs`: Specifying this option tells GCC not to reuse incoming argument registers for copying out arguments.

`-mno-soft-float`: Specifying this option tells GCC to use hardware floating-point instructions for floating-point operations. This is the default.

`-mno-stack-check`: Specifying this option causes GCC not to insert a call to `__msp_check` after each stack adjustment.

`-mno-storem-bug`: Specifying this option causes GCC to not generate code that keeps mtsrim, insn, and storem instructions together. This option should be used when compiling for the 29050 processor, which can handle the separation of these instructions.

`-mnormal`: Specifying this option causes GCC to use the normal memory model that generates call instructions only when calling functions in the same file, and generates calli instructions otherwise. This memory model works correctly if each file occupies less than 256K but allows the entire executable to be larger than 256K. This is the default.

`-mreuse-arg-regs`: Specifying this option tells GCC to use only incoming argument registers for copying out arguments. This helps detect functions that are called with fewer arguments than they were declared with.

`-msmall`: Specifying this option causes GCC to use a small memory model that assumes that all function addresses are either within a single 256K segment or at an absolute address of less than 256K. This allows code to use the call instruction instead of a `const`, `consth`, and `calli` sequence.

`-msoft-float`: Specifying this option tells GCC to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system or these function calls will be identified as unresolved.

`-mstack-check`: Specifying this option causes GCC to insert a call to `__msp_check()` after each stack adjustment. This option is often used when compiling kernel or general operating system code.

`-mstorem-bug`: Specifying this option causes GCC to generate code for AMD 29000 processors that cannot handle the separation of mtsrim, insn, and storem instructions. This option should be used on most 29000 processors, with the exception of the 29050.

`-muser-registers`: Specifying this option causes GCC to use the standard set of global registers (`gr96` through `gr127`). This is the default.

ARC Options

The ARC processor is a highly customizable 32-bit RISC processor core that is often used in embedded systems.

GCC options available when compiling code for ARC processors are the following:

`-EB`: Specifying this option causes GCC to generate code that is compiled for big endian mode (where the most significant byte of a word has the lowest address—the word is stored big end first).

`-EL`: Specifying this option causes GCC to generate code that is compiled for little endian mode (where the most significant byte of a word has the highest significance—the word is stored little end first). This is the default.

`-mcpu=`*`CPU`*: Specifying this option causes GCC to generate code compiled for the specific ARC variant named *`CPU`*. The variants supported by GCC depend on the GCC configuration. Possible values are `arc`, `arc5`, `arc6`, `arc7`, `arc8`, and `base`. All ARC variants support `-mcpu=base`, which is the default.

`-mdata=`*`data-section`*: Specifying this option causes GCC to store data in the section whose name is specified as *data-section*. This command-line option can be overridden for specific functions by using the `__attribute__` keyword to set the `section` attribute for those functions, as explained in Appendix A.

`-mmangle-cpu`: Specifying this option causes GCC to add the name of the CPU at the beginning of all public symbol names. Many multiple-processor systems use ARC CPU variants with different instruction and register set characteristics. Using this option prevents code compiled for one CPU from being linked with code compiled for another.

`-mrodata=`*`readonly-data-section`*: Specifying this option causes GCC to store read-only data in the section whose name is specified as *readonly-data-section*. This command-line option can be overridden for specific functions by using the `__attribute__` keyword to set the `section` attribute for those functions, as explained in Appendix A.

`-mtext=`*`text-section`*: Specifying this option causes GCC to put functions in the section whose name is specified as *text-section*, rather than in the default text section. This command-line option can be overridden for specific functions by using the `__attribute__` keyword to set the `section` attribute for those functions, as explained in Appendix A.

ARM Options

The Advanced RISC Machines (ARM) processor is a 32-bit processor family that is extremely popular in embedded, low-power systems. The ARM acronym originally stood for Acorn RISC Machine because the processor was originally designed by Acorn Computer Systems. Advanced RISC Machines Ltd. was formed to market and develop the ARM processor family, related chips, and associated software, at which point the more modern acronym expansion was adopted.

■ **Note** The ARM instruction set is a complete set of 32-bit instructions for the ARM architecture. The Thumb instruction set is an extension to the 32-bit ARM architecture that provides very high code density through a subset of the most commonly used 32-bit ARM instructions that have been compressed into 16-bit-wide operation codes. On execution, these 16-bit instructions are decoded to enable the same functions as their full 32-bit ARM instruction equivalents.

The ARM Procedure Call Standard (APCS) is frequently referenced in ARM-oriented command-line options for GCC. APCS is a set of standards that defines the use of registers on ARM processors, conventions for using the stack, mechanisms for passing and returning arguments across function calls, and the format and use of the stack.

GCC options available when compiling code for ARM processors are the following:

`-mabi=`*`name`*: Specifying this option tells GCC to generate code for the specified ARM Application Binary Interface (ABI). Possible values for *name* are `apcs-gnu` (the 26- and 32-bit versions of the old APCS), `atpcs` (ARM/Thumb Procedure Call Standard), `aapcs` (ARM Architecture Procedure Call Standard), `aapcs-linux` (ARM Architecture Procedure Call Standard for Linux), and `iwmmxt` (the ABI used on ARM processors that support Intel's wireless MMX technology).

`-mabort-on-noreturn`: Specifying this option causes GCC to generate a call to the `abort` function at the end of each `noreturn` function. The call to `abort` is executed if the function tries to return.

`-malignment-traps`: Specifying this option causes GCC to generate code that will not trap if the MMU has alignment traps enabled by replacing unaligned accesses with a sequence of byte accesses. This option is only relevant for ARM processors prior to the ARM 4 and is ignored on later processors because these have instructions to directly access half-word objects in memory. This option is not supported in any GCC 4.x compiler.

ARM architectures prior to ARM 4 had no instructions to access half-word objects stored in memory. However, a feature of the ARM architecture allows a word load to be used when reading from memory even if the address is unaligned, because the processor core rotates the data as it is being loaded. Specifying this option tells GCC that such misaligned accesses will cause an MMU trap and that it should replace the misaligned access with a series of byte accesses. The compiler can still use word accesses to load half-word data if it knows that the address is aligned to a word boundary.

`-mapcs`: Specifying this option is the same as specifying the `-mapcs-frame` option.

`-mapcs-26`: Specifying this option causes GCC to generate code for an ARM processor running with a 26-bit program counter. The generated code conforms to calling standards for the APCS 26-bit option. The `-mapcd-26` option replaces the `-m2` and `-m3` options provided in older releases of GCC. The `–mapcs-26` option itself is obsolete in the GCC 4.x family of compilers, having largely been replaced by the `-mabi=apcs-gnu` option.

`-mapcs-32`: Specifying this option causes GCC to generate code for an ARM processor running with a 32-bit program counter. The generated code conforms to calling standards for the APCS 32-bit option. The `-mapcd-26` option replaces the `-m6` option provided in older releases of GCC. The `-mapcs-32` option itself is obsolete in the GCC 4.x family of compilers, having largely been replaced by the `-mabi=apcs-gnu` option.

`-mapcs-float`: Specifying this option tells GCC to pass floating-point arguments in floating-point registers.

`-mapcs-frame`: Specifying this option causes GCC to generate a stack frame that is compliant with the APCS for all functions, even if this is not strictly necessary for correct execution of the code. Using this option in conjunction with the `-fomit-frame-pointer` option causes GCC not to generate stack frames for leaf functions (functions that do not call other functions). The default is to not generate stack frames.

`-mapcs-reentrant`: Specifying this option tells GCC to generate re-entrant, position-independent code.

`-mapcs-stack-check`: Specifying this option causes a stack limit of 64K to be set in the code produced by GCC, with tests imposed to ensure that this stack limit is not exceeded. Used primarily in bare metal ARM code—that is, code that is designed to run on ARMs without an operating system.

`-march=`*name*: Using this option enables you to identify the ARM architecture used on the system for which you are compiling. Like the `-mcpu` option, GCC uses this name to determine the instructions that it can use when generating assembly code. This option can be used in conjunction with or instead of the `-mcpu=` option. Possible values for *name* are `armv2`, `armv2a`, `armv3`, `armv3m`, `armv4`, `armv4t`, `armv5`, `armv5t`, `armv5te`, `armv6`, `armv6j`, `iwmmxt`, and `ep9312`. See the `-mcpu=`*name* and `-mtune=`*name* options for related information.

`-marm`: Specifying this option identifies the system as an ARM system without Thumb support.

`-mbig-endian`: Specifying this option causes GCC to generate code for an ARM processor running in big endian mode.

`-mbsd`: Specifying this option causes GCC to emulate the native BSD-mode compiler. This is the default if `-ansi` is not specified. This option is only relevant when compiling code for RISC iX, which is Acorn's version of Unix that was supplied with some ARM-based systems such as the Acorn Archimedes R260. This option is no longer supported in the GCC 4.x compilers.

`-mcallee-super-interworking`: Specifying this option causes GCC to insert an ARM instruction set header before executing any externally visible function. *Interworking mode* is a mode of operation in which ARM and Thumb instructions can interact. This header causes the processor to switch to Thumb mode before executing the rest of the function. This allows these functions to be called from noninterworking code.

`-mcaller-super-interworking`: Specifying this option enables calls via function pointers (including virtual functions) to execute correctly regardless of whether the target code has been compiled for interworking or not. This option causes a slight increase in the cost of executing a function pointer but facilitates interworking in all circumstances.

`-mcirrus-fix-invalid-isns`: Specifying this option (with one of the best names I've ever seen) causes GCC to insert NOOPs in the code that it generates to avoid invalid combinations and sequences of operations in some ARM-based products from Cirrus Logic.

`-mcpu=`*name*: Using this option enables you to specify the particular type of ARM processor used on the system for which you are compiling. GCC uses this name to determine the instructions that it can use when generating assembly code. Possible values for *name* are `arm2`, `arm250`, `arm3`, `arm6`, `arm60`, `arm600`, `arm610`, `arm620`, `arm7`, `arm7m`, `arm7d`, `arm7dm`, `arm7di`, `arm7dmi`, `arm70`, `arm700`, `arm700i`, `arm710`, `arm710c`, `arm7100`, `arm7500`, `arm7500fe`, `arm7tdmi`, `arm8`, `strongarm`, `strongarm110`, `strongarm1100`, `arm8`, `arm810`, `arm9`, `arm9e`, `arm920`, `arm920t`, `arm940t`, `arm9tdmi`, `arm10tdmi`, `arm1020t`, `arm1136j-s`, `arm1136jf-s`, `arm1176jz-s`, `arm1176jzf-s`, `ep9312`, `iwmmxt`, `mpcore`, `mpcorenovfp`, and `xscale`. See the `-march=`*name* and `-mtune=`*name* options for related information.

■ **Note** Because ARM licenses its cores to different manufacturers and is constantly developing new processors, the previous list is likely to change by the time that you read this.

`-mfloat-abi=`*name*: Using this option provides a single centralized mechanism for identifying how GCC should generate floating-point instructions. Valid values for *name* are `soft` (generates floating-point instructions as library calls, equivalent to `-msoft-float`), `hard` (generates actual floating-point instructions, equivalent to `-mhard-float`), and `softfp` (generates actual floating point instructions but uses the `-msoft-float` calling conventions). The older options are still supported.

`-mfp=`*number* | `-mfpe=`*number* | `-mfpu=`*name*: Using these options enables you to specify the version of floating-point emulation that is available on the system for which you are compiling. Possible values for *number* are `2` and `3`, which identify different internal implementations. Possible values for *name* are `fpa`, `fpe2`, `fpe3`, `maverick`, and `vfp`. You should use `-mpfu=`*name* (the most modern of these options) in any new scripts or Makefiles, as the other options are technically obsolete; `-mfp=2` and `-mfpe=2` are now internal aliases for `-mfpu=fpe2`, while `-mfp=3` and `-mfpe=3` are now aliases for `-mfpu=fpe3`.

`-mhard-float`: Specifying this option causes GCC to generate output containing floating-point instructions, which is the default. This option still works, but has been conceptually replaced by `-mfloat-abi=hard`.

`-mlittle-endian`: Specifying this option causes GCC to generate code for an ARM processor running in little endian mode. This is the default.

`-mlong-calls`: Specifying this option causes GCC to perform function calls by first loading the address of the function into a register and then performing a subroutine call on this register. This option is necessary if the target function will lie outside of the 64MB addressing range of the offset-based version of the subroutine call instruction. Specifying this option does not affect calls to static functions, functions that have the `short-call` attribute, functions that are inside the scope of a `#pragma no_long_calls` directive, and functions whose definitions have already been compiled within the current compilation unit. Specifying this option does affect weak function definitions, functions with the `long-call` attribute or the `section` attribute, and functions that are within the scope of a `#pragma long_calls` directive.

`-mno-alignment-traps`: Specifying this option causes GCC to generate code that assumes that the MMU will not trap unaligned accesses. This produces better code for ARM processors prior to the ARM 4, where the target instruction set does not have half-word memory access operations. Unfortunately, you cannot use this option to access unaligned word objects, since the processor will only fetch one 32-bit aligned object from memory. This option is not supported in the GCC 4.x family of compilers.

`-mno-apcs-frame`: Specifying this option causes GCC not to generate stack frames that are compliant with the ARM Procedure Call Standard for all functions unless they are necessary for correct execution of the code. This option is the default.

`-mno-long-calls`: Specifying this option causes GCC to perform function calls in the standard fashion, rather than by the mechanism described in the `-mlong-calls` option. This is the default.

`-mno-sched-prolog`: Specifying this option causes GCC to suppress possible optimizations, preventing reordering instructions in the function prologue and preventing merging those instructions with the instructions in the function's body. Specifying this option means that all functions will start with a recognizable set of instructions from one of the sets of default function prologues, which can then be used to identify the beginning of functions within an executable piece of code. Using this option typically results in larger executables than the default option, `-msched-prolog`.

`-mno-short-load-bytes`: This is a deprecated alias for the `-mno-alignment-traps` option.

`-mno-short-load-words`: This is a deprecated alias for the `-malignment-traps` option.

`-mno-soft-float`: Specifying this option tells GCC to use hardware floating-point instructions for floating-point operations. This option implies `-mhard-float`, but should not be used in new code for logical reasons, because the new `-mfloat-abi=`*name* option provides three options, and therefore *not soft-float* can mean one of two options to anyone who inherits your code.

`-mno-symrename`: Specifying this option causes GCC not to run the assembler post-processor, symrename, after assembling code on a RISC iX system. This post-processor is normally run in order to modify standard symbols in assembly output so that resulting binaries can be successfully linked with the RISC iX C library. This option is only relevant for versions of GCC that are running directly on a RISC iX system: no such post-processor is provided by GCC when GCC is used as a cross-compiler. This option is not supported by the GCC 4.x family of compilers, because RISC iX systems are not supported by GCC 4.x.

`-mno-thumb-interwork`: Specifying this option causes GCC to generate code that does not support calls between the ARM and Thumb instruction sets. This option is the default. See the `-mthumb-interwork` option for related information.

`-mno-tpcs-frame`: Specifying this option causes GCC not to generate stack frames that are compliant with the Thumb Procedure Call Standard. This option is the default. See the `-mtpcs-frame` option for related information.

`-mno-tpcs-leaf-frame`: Specifying this option causes GCC to not generate stack frames that are compliant with the Thumb Procedure Call Standard. This option is the default. See the `-mtpcs-leaf-frame` option for related information.

`-mnop-fun-dllimport`: Specifying this option causes GCC to disable support for the `dllimport` attribute.

`-mpic-register=`*name*: Using this option enables you to specify the register (`name`) to be used for PIC addressing. When this option is not supplied, the default register used for PIC addressing is register `R10` unless stack checking is enabled, in which case register `R9` is used.

`-mpoke-function-name`: Specifying this option causes GCC to write the name of each function into the text section immediately preceding each function prologue. Specifying this option means that all functions will be preceded by a recognizable label that can then be used to identify the beginning of functions within an executable piece of code.

`-msched-prolog`: Specifying this option causes GCC to optimize function prologue and body code sequences, merging operations whenever possible. This option is the default.

`-mshort-load-bytes`: This is a deprecated alias for the `-malignment-traps` option, which itself is not supported in the GCC 4.x family of compilers.

`-mshort-load-words`: This is a deprecated alias for the `-mno-alignment-traps` option, which itself is not supported in the GCC 4.x family of compilers.

`-msingle-pic-base`: Specifying this option causes GCC to treat the register used for PIC addressing as read-only, rather than loading it in the prologue for each function. The runtime system is responsible for initializing this register with an appropriate value before execution begins.

`-msoft-float`: Specifying this option causes GCC to generate output containing library calls for floating point. These libraries are not provided as part of GCC but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system, or these function calls will be identified as unresolved. This option still works but has been conceptually replaced by `-mfloat-abi=soft`.

■ **Tip** Specifying this option also changes the calling convention used in the output file. You must therefore compile all of the modules of your program with this option, including any libraries that you reference. You must also compile libgcc.a, the library that comes with GCC, with this option in order to be able to use it in your applications.

`-mstructure-size-boundary=`*n*: Using this option causes GCC to round the size of all structures and unions up to a multiple of the number of bits (*n*) specified with this option. Valid values are `8` and `32`, and vary for different output formats; the default value is 8 for COFF output toolchains. Code compiled with one value will not necessarily work with code or libraries compiled with the other value. Specifying the larger number might produce faster, more efficient code, but may also increase the size of the program.

`-mthumb`: Specifying this option causes GCC to generate code for the 16-bit Thumb instruction set. The default is to use the 32-bit ARM instruction set.

`-mthumb-interwork`: Specifying this option causes GCC to generate code that supports calls between the ARM and Thumb instruction sets. If this option is not specified, the two instruction sets cannot be reliably used inside one program. Specifying this option causes GCC to generate slightly larger executables than the ARM-specific default option, `-mno-thumb-interwork`.

`-mtpcs-frame`: Specifying this option causes GCC to generate stack frames that are compliant with the Thumb Procedure Call Standard. Specifying this option only affects nonleaf functions (i.e., functions that call other functions). The default is `-mno-tpcs-frame`—in other words, not to generate compliant stack frames.

`-mtpcs-leaf-frame`: Specifying this option causes GCC to generate stack frames that are compliant with the Thumb Procedure Call Standard. Specifying this option only affects leaf functions (i.e., functions that do not call other functions). The default is `-mno-tpcs-leaf-frame`—in other words, not to generate compliant stack frames.

`-mtune=`*`name`*: Specifying this option causes GCC to tune the generated code as though the target ARM processor were of type *name*, but to still generate code that conforms to the instructions available for an ARM processor specified using the `-mcpu` option. Using these two options together can provide better performance on some ARM-based systems. See the `-march=`*`name`* and `-mcpu=`*`name`* options for related information.

`-mwords-little-endian`: Specifying this option causes GCC to generate code for a little endian word order but a big endian byte order (i.e., of the form 32107654). This option only applies when generating code for big endian processors and should only be used for compatibility with code for big endian ARM processors that was generated by versions of GCC prior to 2.8.

`-mxopen`: Specifying this option causes GCC to emulate the native X/Open-mode compiler. This is the default if `-ansi` is not specified. This option is only relevant when compiling code for RISC iX, which is Acorn's version of Unix that was supplied with some ARM-based systems such as the Acorn Archimedes R260. This option is not supported in the GCC 4.x compiler family, because RISC iX systems are no longer supported.

AVR Options

Atmel's AVR processors are microcontrollers with a RISC core running single-cycle instructions. They provide a well-defined I/O structure that limits the need for external components, and are therefore frequently used in embedded systems.

GCC options available when compiling code for AVR processors are the following:

`-mcall-prologues`: Specifying this option causes GCC to expand function prologues and epilogues as calls to the appropriate subroutines, reducing code size.

`-mdeb`: Specifying this option generates extensive debugging information.

`-minit-stack=`*`n`*: Specifying this option enables you to define the initial stack address, which may be a symbol or a numeric value. The value `__stack` is the default.

`-mint8`: Specifying this option tells GCC to use an 8-bit `int` datatype. When this option is used, a `char` will also be 8 bits, a `long` will be 16 bits, and a `long long` will be 32 bits. Though this option does not conform to C standards, using it will almost always produce smaller binaries.

`-mmcu=`*`MCU`*: Specifying this option enables you to specify the AVR instruction set or `MCU` type for which code should be generated. Possible values for the instruction set are the following:

- `avr1`: The minimal AVR core. This value is not supported by the GCC C compiler but only by the GNU assembler. Associated `MCU` types are `at90s1200`, `attiny10`, `attiny11`, `attiny12`, `attiny15`, and `attiny28`.
- `avr2`: The classic AVR core with up to 8K of program memory space. This is the default. Associated `MCU` types are `at90s2313`, `at90s2323`, `attiny22`, `at90s2333`, `at90s2343`, `at90s4414`, `at90s4433`, `at90s4434`, `at90s8515`, `at90c8534`, and `at90s8535`.
- `avr3`: The classic AVR core with up to 128K of program memory space. Associated `MCU` types are `atmega103`, `atmega603`, `at43usb320`, and `at76c711`.
- `avr4`: The enhanced AVR core with up to 8K of program memory space. Associated `MCU` types are `atmega8`, `atmega83`, and `atmega85`.
- `avr5`: The enhanced AVR core with up to 128K of program memory space. Associated `MCU` types are `atmega16`, `atmega161`, `atmega163`, `atmega32`, `atmega323`, `atmega64`, `atmega128`, `at43usb355`, and `at94k`.

`-mno-interrupts`: Specifying this option causes GCC to generate code that is not compatible with hardware interrupts, reducing code size.

`-mno-tablejump`: Specifying this option causes GCC not to generate tablejump instruction, which may increase code size.

`-mshort-calls`: Specifying this option causes GCC to use the rjmp/rcall instructions (which have limited range) on devices with greater than 8K of memory.

`-msize`: Specifying this option causes GCC to output instruction sizes to the assembler output file.

`-mtiny-stack`: Specifying this option tells GCC to generate code that only uses the low 8 bits of the stack pointer.

Blackfin Options

Analog Devices' Blackfin processors are very interesting 32-bit RISC processors that are primarily designed for audio, video, communications, and related embedded projects, offering a complete 32-bit RISC programming model on an SIMD (single-instruction, multiple-data stream) architecture.

GCC options available when compiling code for Blackfin processors are the following:

`-mcsync-anomaly`: Specifying this option tells GCC to generate extra code that guarantees that CSYNC or SSYNC instructions do not occur too soon after any conditional branch. This option works around a hardware anomaly and is active by default.

`-mid-shared-library`: Specifying this option tells GCC to generate code that supports shared library using library IDs, and implies the generic GCC `-fPIC` option. This option therefore supports things such as execute in place and shared libraries without requiring virtual memory management.

`-mlong-calls`: Specifying this option tells GCC to perform function calls by first loading the address of the function into a register and then performing a subroutine call on this register. This option is necessary if the target function will lie outside of the 24-bit addressing range of the offset-based version of the subroutine call instruction.

`-mlow64k`: Specifying this option tells GCC that the entire program will fit into the low 64K of memory and to perform any possible optimizations related to that fact.

`-mno-csync-anomaly`: Specifying this option tells GCC not to generate any extra NOOPs necessary to prevent CSYNC or SSYNC instructions from occurring too quickly after conditional branches.

`-mno-id-shared-library`: Specifying this option tells GCC not to generate code that supports shared libraries using library IDs. This option is active by default.

`-mno-long-calls`: Specifying this option causes GCC to perform function calls in the standard fashion, rather than by the mechanism described in the `-mlong-calls` option. This is the default.

`-mno-low64k`: Specifying this option tells GCC that the program size is unknown and not to attempt to perform optimizations based on assuming that the program is less than 64K. This option is active by default.

`-mno-omit-leaf-frame-pointer`: Specifying this option tells GCC to retain the frame pointer in a register for leaf functions, simplifying debugging but generating larger binaries due to the setup, save, and restore operations.

`-mno-specld-anomaly`: Specifying this option tells GCC not to generate the extra code used to prevent speculative loads after jumps.

`-momit-leaf-frame-pointer`: Specifying this option tells GCC not to keep the frame pointer in a register for leaf functions. This reduces the number of instructions required to set up, save, and restore frame pointers but makes debugging more complex.

`-mshared-library-id=`*n*: Specifying this option enables you to specify the ID number (*n*) of an ID-based shared library that is being compiled.

`-mspecld-anomaly`: Specifying this option tells GCC to generate extra code to prevent speculative loads after jump instructions. This option works around a hardware anomaly, and is active by default.

Clipper Options

Clipper is a family of RISC processors that were primarily used in older Intergraph Unix workstations.

■ **Note** Support for this processor family was marked as obsolete in GCC 3.1 and was fully purged in GCC version 3.2.2. These options are therefore only of interest if you are using a version of GCC that is earlier than 3.2.1 and that you are certain provides support for this processor family.

GCC options available when compiling code for Clipper processors are the following:

`-mc300`: Specifying this option causes GCC to generate code for a C300 Clipper processor. This is the default.

`-mc400`: Specifying this option causes GCC to generate code for a C400 Clipper processor. The generated code uses floating-point registers `f8` through `f15`.

Convex Options

Convex Computer systems were minisupercomputers that were targeted for use by small to medium-sized businesses. Systems such as the C1, C2, and C3 were high-performance vector-processing systems that were substantially less expensive than the competing systems from Cray Research. The later Exemplar systems were based on the Hewlett-Packard PA-RISC CPU series. Convex was acquired by HP in 1995.

■ **Note** Support for this processor family was marked as obsolete in GCC 3.1 and was fully purged in GCC version 3.2.2. These options are therefore only of interest if you are using a version of GCC that is earlier than 3.2.1 and that you are certain provides support for this processor family.

GCC options available when compiling code for Convex systems are the following:

`-margcount`: Specifying this option causes GCC to generate code that puts an argument count in the word preceding each argument list. This argument-count word is compatible with the regular C compiler provided by ConvexOS, and may be needed by some programs.

`-mc1`: Specifying this option causes GCC to generate code targeted for Convex C1 systems, but which will run on any Convex machine. This option defines the preprocessor symbol `__convex_c1__`.

`-mc2`: Specifying this option causes GCC to generate code targeted for Convex C2 systems, but which should also run on Convex C3 machines, though scheduling and other optimizations are chosen for maximum performance on C2 systems. This option defines the preprocessor symbol `__convex_c2__`.

`-mc32`: Specifying this option causes GCC to generate code targeted for Convex C32xx systems, using scheduling and other optimizations chosen for maximum performance on C32xx systems. This option defines the preprocessor symbol `__convex_c32__`.

`-mc34`: Specifying this option causes GCC to generate code targeted for Convex C34xx systems, using scheduling and other optimizations chosen for maximum performance on C34xx systems. This option defines the preprocessor symbol `__convex_c34__`.

`-mc38`: Specifying this option causes GCC to generate code targeted for Convex C38xx systems, using scheduling and other optimizations that are chosen for maximum performance on C38xx systems. This option defines the preprocessor symbol `__convex_c38__`.

`-mlong32`: Specifying this option causes GCC to define type `long` as 32 bits, the same as type `int`. This is the default.

`-mlong64`: Specifying this option causes GCC to define type `long` as 64 bits, the same as type `long long`. This option is essentially useless because there is no support for this convention in the GCC libraries.

`-mnoargcount`: Specifying this option causes GCC to omit the argument-count word. This is the default. See the `-margcount` option for related information.

`-mvolatile-cache`: Specifying this option causes GCC to generate code in which volatile references are cached. This is the default.

`-mvolatile-nocache`: Specifying this option causes GCC to generate code in which volatile references bypass the data cache, going directly to memory. This option is only needed for multiprocessor code that does not use standard synchronization instructions. Making nonvolatile references to volatile locations will not necessarily work.

CRIS Options

Axis Solutions' Code Reduced Instruction Set (CRIS) processors are frequently used in network-oriented embedded applications.

GCC options available when compiling code for CRIS systems are the following:

`-m16-bit`: Specifying this option tells GCC to align the stack frame, writable data, and constants to all be 16-bit aligned. The default is 32-bit alignment.

`-m32-bit`: Specifying this option tells GCC to align the stack frame, writable data, and constants to all be 32-bit aligned. This is the default.

`-m8-bit`: Specifying this option tells GCC to align the stack frame, writable data, and constants to all be 8-bit aligned. The default is 32-bit alignment.

`-maout`: This deprecated option is a NOOP that is only recognized with the `cris-axis-aout` GCC build target.

`-march=`*`architecture-type`* | `-mcpu=`*`architecture-type`*: Specifying either of these options causes GCC to generate code for the specified architecture. Possible values for *`architecture-type`* are `v3` (for ETRAX 4), `v8` (for ETRAX 100), and `v10` (for ETRAX 100 LX). The default is `v0` except for the `cris-axis-linux-gnu` GCC build target, where the default is `v10`.

`-mbest-lib-options`: Specifying this option tells GCC to select the most feature-rich set of options possible based on all other options that have been specified.

`-mcc-init`: Specifying this option tells GCC not to use condition-code results from previous instructions, but to always generate compare and test instructions before using condition codes.

`-mconst-align`: Specifying this option tells GCC to align constants for the maximum single data access size for the chosen CPU model. The default is to arrange for 32-bit alignment. ABI details such as structure layout are not affected by this option.

`-mdata-align`: Specifying this option tells GCC to align individual data for the maximum single data access size for the chosen CPU model. The default is to arrange for 32-bit alignment. ABI details such as structure layout are not affected by this option.

`-melf`: This deprecated option is a NOOP that is only recognized with the `cris-axis-elf` and `cris-axis-linux-gnu` GCC build targets.

`-melinux`: Specifying this option tells GCC to select a GNU/Linux-like multilib, using include files and the instruction set for `-march=v8`. This option is only recognized with the `cris-axis-aout` GCC build target.

`-melinux-stacksize=`*`n`*: Specifying this option tells GCC to include instructions in the program so that the kernel loader sets the stack size of the program to *n* bytes. This option is only available on the `cris-axis-aout` GCC build target.

`-metrax100`: Specifying this option is the same as the `-march=v8` option.

`-metrax4`: Specifying this option is the same as the `-march=v3` option.

`-mgotplt`: Specifying this option, when used with the `-fpic` or `-fPIC` options, tells GCC to generate instruction sequences that load addresses for functions from the PLT (procedure linkage table) part of the GOT (global offset table) rather than by making calls to the PLT. The GOT holds the resolved virtual addresses of symbols. The PLT provides the glue between a function call and the dynamically linked target of that call. This option is active by default.

`-mlinux`: This deprecated option is a NOOP that is only recognized with the `cris-axis-linux-gnu` GCC build target.

`-mmax-stack-frame=`*`n`*: Specifying this option causes GCC to display a warning when the stack frame of a function exceeds *n* bytes.

`-mmul-bug-workaround`: Specifying this option tells GCC to generate additional code to work around a hardware bug in multiplication instructions on CRIS processors, such as the ETRAX 100, where the problem may be present.

`-mno-const-align`: Specifying this option tells GCC not to align the constants for the maximum single data access size for the chosen CPU model. The default is to arrange for 32-bit alignment. ABI details such as structure layout are not affected by this option.

`-mno-data-align`: Specifying this option tells GCC not to align individual data for the maximum single data access size for the chosen CPU model. The default is to arrange for 32-bit alignment. ABI details such as structure layout are not affected by this option.

`-mno-gotplt`: When used with the `-fpic` or `-fPIC` options, specifying this option tells GCC to make calls to the PLT rather than load addresses for functions from the PLT part of the GOT.

`-mno-mul-bug-workaround`: Specifying this option tells GCC not to generate the additional code required to work around a hardware bug in multiplication instructions on CRIS processors, such as the ETRAX 100, where the problem may be present.

`-mno-prologue-epilogue`: Specifying this option tells GCC to omit the normal function prologue and epilogue that sets up the stack frame, and not to generate return instructions or return sequences in the code. This option is designed to simplify visual inspection of compiled code, as it may generate code that stops on certain registers. No warnings or errors are generated when call-saved registers must be saved, or when storage for local variables needs to be allocated.

`-mno-side-effects`: Specifying this option tells GCC not to emit instructions with side effects when using addressing modes other than post-increment.

`-mno-stack-align`: Specifying this option tells GCC not to align the stack frame for the maximum single data access size for the chosen CPU model. The default is to arrange for 32-bit alignment. ABI details such as structure layout are not affected by this option.

`-moverride-best-lib-options`: Specifying this option tells GCC not to automatically select the most feature-rich set of options possible based on all other options that have been specified. This option is required for some files compiled with multilib support. (This option is not available in the GCC 4.x compilers.)

`-mpdebug`: Specifying this option tells GCC to enable verbose CRIS-specific debugging information in the assembly code. This option also turns off the `#NO_APP` formatted-code indicator at the beginning of the assembly file.

`-mprologue-epilogue`: Specifying this option tells GCC to include the normal function prologue and epilogue that set up the stack frame. This is the default.

`-mstack-align`: Specifying this option tells GCC to align the stack frame for the maximum single data access size for the specified CPU model. The default is to arrange for 32-bit alignment. ABI details such as structure layout are not affected by this option.

`-mtune=`*`architecture-type`*: Specifying this option causes GCC to tune the generated code for the specified `architecture-type`, except for the ABI and the set of available instructions. The choices for *`architecture-type`* are the same as those for the `-march=`*`architecture-type`* option.

`-sim`: Specifying this option tells GCC to link with input-output functions from a simulator library. Code, initialized data, and zero-initialized data are allocated consecutively. This option is only recognized for the `cris-axis-aout` and `cris-axis-elf` GCC build targets.

`-sim2`: Specifying this option tells GCC to link with input-output functions from a simulator library and to pass linker options to locate initialized data at the memory address 0x40000000 and zero-initialized data at 0x80000000. Code, initialized data, and zero-initialized data are allocated consecutively. This option is only recognized for the `cris-axis-aout` and `cris-axis-elf` GCC build targets.

CRX Options

National Semiconductor's CRX microcontrollers are 32-bit RISC processors that are a follow-up to its 16-bit CompactRISC CR16 processors.

GCC options available when compiling code for CRX-based systems are the following:

`-mloopnesting=n`: Specifying this option tells GCC to restrict nested `do` loops to the nesting level specified by *n*. (This option is not available in the GCC 4.x compilers.)

`-mmac`: Specifying this option tells GCC to support the use of the multiply-accumulate instructions. This option is disabled by default.

`-mno-push-args`: Specifying this option tells GCC not to use push instructions to pass outgoing arguments when functions are called.

`-mpush-args`: Specifying this option tells GCC to use push instructions to pass outgoing arguments when functions are called. This option is enabled by default.

D30V Options

Mitsubishi's D30V processor is a RISC processor with integrated hardware support for a real-time MPEG-2 decoder, and therefore targets embedded multimedia applications.

■ **Note** I've never actually seen one of these processors and I believe that GCC was just beginning to be used when Mitsubishi was developing its hardware prototypes. These processors are not supported in the GCC 4.x family of compilers. I would not try to use them with anything later than GCC 3.2. Support for them was dropped long ago in related tools such as GDB 5.2.

GCC options available when compiling code for D30V-based systems are the following:

`-masm-optimize`: Specifying this option tells GCC to pass the `-O` option to the assembler during optimization. The assembler uses the `-O` option to automatically parallelize adjacent short instructions whenever possible.

`-mbranch-cost=n`: Specifying this option tells GCC to increase the internal cost of a branch to the value specified as *n*. Higher costs mean that the compiler will generate more instructions in order to avoid doing a branch. Avoiding branches is a common performance optimization for RISC processors. The default is 2.

`-mcond-exec=n`: Specifying this option enables you to identify the maximum number of conditionally executed instructions that replace a branch as the value *n*. The default is 4.

`-mextmem` | `-mextmemory`: Specifying these options tells GCC to link the text, data, bss, strings, rodata, rodata1, and data1 sections into external memory, which starts at location 0x80000000.

`-monchip`: Specifying this option tells GCC to link the text section into on-chip text memory, which starts at location 0x0. This option also tells GCC to link data, bss, strings, rodata, rodata1, and data1 sections into on-chip data memory, which starts at location 0x20000000.

`-mno-asm-optimize`: Specifying this option tells GCC to not pass the `-O` option to the assembler, thereby suppressing optimization.

Darwin Options

Darwin is the open source operating system environment that underlies Apple's Mac OS X operating system. Based on a combination of the Mach operating system kernel and technologies from FreeBSD, it is a freely available operating system that has its roots in the Berkeley Standard Distribution of

Unix. As you'll see in this section, Apple has heavily leveraged and customized the GCC tools and related open source technologies such as binutils for use as compilers, linkers, and so on, to compile Darwin, Darwin system utilities that run from the Mac OS X command line, and the graphical components of Mac OS X, known as Aqua. Because GCC, binutils, and so on, are all open source software, we all benefit from Apple's adoption of GCC.

Of course, as this appendix shows, adopting GCC to a new operating system results in many new options for GCC (and especially the Darwin version of ld). The GCC-specific options for Darwin are the following:

`-all_load`: Specifying this option causes GCC to load all members of static archive libraries.

`-arch_errors_fatal`: Specifying this option causes GCC compilers to treat any errors associated with files that were compiled for the wrong architecture as fatal. This option is especially important when producing the new generation of Mac OS X fat (multiarchitecture) binaries that contain both PowerPC and x86 executables.

`-bind_at_load`: Specifying this option causes GCC compilers to mark an object file or executable such that the Mac OS X dynamic linker will bind all undefined references when the file is loaded or executed.

`-bundle`: Specifying this option causes GCC compilers to produce a Mach-O bundle format file. Headers are located in the first segment; all sections are placed in the proper segments and are padded to ensure proper segment alignment. The file type for the resulting bundle is `MH_BUNDLE`.

`-bundle_loader` *`executable`*: Specifying this option identifies the executable that will be loading the object file(s) that are being linked. Undefined symbols in the bundle are checked against the specified executable just as if it were a dynamic library that the bundle was linked with. If the `-twolevel_namespace` option is used to segment the symbol namespace, symbol searches are based on the placement of the `-bundle_loader` option. If the `-flat_namespace` option is specified, the executable is searched before all dynamic libraries. This option is only valid for 32-bit executables.

`-dynamiclib`: Specifying this option causes GCC compilers to produce a dynamic library using the Darwin libtool command rather than an executable when linking.

`-F`*`dir`*: Specifying this option causes GCC compilers to add the frameworks directory *`dir`* to the beginning of the list of directories to be searched for header files. Frameworks is the term for the directory structure and associated set of naming conventions used with Apple's Xcode development environment and tools.

`-force_cpusubtype_ALL`: Specifying this option causes GCC compilers to produce output files that have the `ALL` subtype, a set of instructions common to all processors, rather than the one controlled by the `-mcpu` or `-march` options. This option is especially important when producing the new generation of Mac OS X fat (multiarchitecture) binaries that contain both PowerPC and x86 executables.

`-gused`: Specifying this option causes GCC compilers to generate debugging information only for symbols that are used. This option is on by default. For projects that use the STABS debugging format, specifying this option also enables the `-feliminate-unused-debug-symbols` option.

`-gfull`: Specifying this option causes GCC compilers to generate debugging information for all symbols and types.

`-mfix-and-continue` | `-ffix-and-continue` | `-findirect-data`: Specifying any of these options causes GCC to generate code that is suitable for fast turnaround development, enabling GDB to dynamically load new object files into programs that are already running. The `-findirect-data` and `-ffix-and-continue` synonyms for `-mfix-and-continue` are provided for backward compatibility.

`-mmacosx-version-min=`*`version`*: Specifying this option identifies Mac OS X version *`version`* as the earliest version of Mac OS X that an executable will run on. The value for *`version`* can be a two- or three-digit dot-separated version identifier. Common values for *`version`* are `10.1`, `10.2`, and `10.3.9`.

`-mone-byte-bool`: Specifying this option tells GCC compilers to use a single byte to store Boolean values, so that `sizeof(bool)` is 1. This option is only meaningful on Darwin/PowerPC platforms where `sizeof(bool)` is normally 4. Using this option generates object code that is incompatible with object code generated without specifying this option. You will therefore need to use this option with none or all of the object code involved in an executable, including system libraries.

■ **Note** GCC for the Darwin platform also supports a number of options that it simply passes through to the Darwin linker in order to invoke specific linker options. Since these are actually options for the Darwin linker rather than the vanilla GNU linker, they are not discussed here—for details, see the man page for ld, the GNU linker/loader, on a Darwin/Mac OS X system. These options are the following: `-allowable_client` *`CLIENT-NAME`*, `-client_name` *`CLIENT-NAME`*, `-compatibility_version` *`NUMBER`*, `-current_version` *`NUMBER`*, `-dead_strip`, `-dependency-file`, `-dylib_file`, `-dylinker_install_name` *`NAME`*, `-dynamic`, `-exported_symbols_list`, `-filelist`, `-flat_namespace`, `-force_flat_namespace`, `-headerpad_max_install_names`, `-image_base`, `-init`, `-install_name` *`NAME`*, `-keep_private_externs`, `-multi_module`, `-multiply_defined`, `-multiply_defined_unused`, `-noall_load`, `-no_dead_strip_inits_and_terms`, `-nofixprebinding`, `-nomultidefs`, `-noprebind`, `-noseglinkedit`, `-pagezero_size`, `-prebind`, `-prebind_all_twolevel_modules`, `-private_bundle`, `-read_only_relocs`, `-sectalign`, `-sectobjectsymbols`, `-whyload`, `-seg1addr`, `-sectcreate`, `-sectobjectsymbols`, `-sectorder`, `-segaddr`, `-segs_read_only_addr`, `-segs_read_write_addr`, `-seg_addr_table`, `-seg_addr_table_filename`, `-seglinkedit`, `-segprot`, `-segs_read_only_addr`, `-segs_read_write_addr`, `-single_module`, `-sub_library`, `-sub_umbrella`, `-twolevel_namespace`, `-umbrella`, `-undefined`, `-unexported_symbols_list`, `-weak_reference_mismatches`, `-whatsloaded`, and `-whyload`.

FR-V Options

The Fujitsu FR-V is a Very Long Instruction Word (VLIW) processor core that was developed to support embedded applications in the digital consumer electronics, cell phone, mobile electronics, and automotive navigation markets. The FR-V is a very low-power core that provides a 16-bit integer instruction set, a 32-bit integer instruction set, a media instruction set, a digital signal instruction set, and a floating-point instruction set. The floating-point instruction set is based on the IEEE 754 specification.

GCC options when compiling code for the FR-V processor are the following:

`-macc-4`: Specifying this option causes the GCC compilers to only use the first four media accumulator registers.

`-macc-8`: Specifying this option causes the GCC compilers to use all eight media accumulator registers.

`-malign-labels`: Specifying this option causes the GCC compilers to align labels to an 8-byte boundary by inserting NOOPs. This option only has an effect when VLIW packing is enabled with `-mpack`.

`-malloc-cc`: Specifying this option causes the GCC compilers to dynamically allocate condition code registers.

`-mcond-exec`: Specifying this option causes the GCC compilers to use conditional execution and is active by default. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mcond-move`: Specifying this option causes the GCC compilers to enable the use of conditional-move instructions. This option is active by default. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mcpu=`*CPU*: Specifying this option enables you to specify the type of processor for which the GCC compilers will generate code. Possible values for *CPU* are `frv`, `fr550`, `tomcat`, `fr500`, `fr450`, `fr405`, `fr400`, `fr300`, and `simple`.

`-mdouble`: Specifying this option causes the GCC compilers to use an ABI that supports double-word floating-point instructions.

`-mdword`: Specifying this option causes the GCC compilers to use an ABI that supports double-word instructions.

`-mfdpic`: Specifying this option causes the GCC compilers to use the FDPIC ABI, which uses function descriptors to represent pointers to functions. If no `PIC-` or `PIE`-related options are specified, using this option implies `-fPIE`. If the `-fpic` or `-fpie` options are specified, using this option assumes that global offset table (GOT) entries and small data are within a 12-bit range from the GOT base address. If the `-fPIC` or `-fPIE` options are specified, GOT offsets are computed with 32 bits.

`-mfixed-cc`: Specifying this option causes the GCC compilers to use only `icc0` and `fcc0` rather than trying to dynamically allocate condition code registers.

`-mfpr-32`: Specifying this option causes the GCC compilers to use only the first 32 floating-point registers.

`-mfpr-64`: Specifying this option causes the GCC compilers to use all 64 floating-point registers.

`-mgpr-32`: Specifying this option causes the GCC compilers to use only the first 32 general-purpose registers.

`-mgpr-64`: Specifying this option causes the GCC compilers to use all 64 general-purpose registers.

`-mgprel-ro`: Specifying this option causes the GCC compilers to enable the use of GPREL (global pointer relative) relocations in the FDPIC ABI for data that is known to be in read-only sections. This option is enabled by default unless the `-fpic` or `-fpie` options are being used. When the `-fPIC` or `-fPIE` options are being used, this option avoids the need for GOT entries for referenced symbols. If you need these entries, you can use this option to enable them.

`-mhard-float`: Specifying this option causes the GCC compilers to use hardware instructions for floating-point operations.

`-minline-plt`: Specifying this option causes the GCC compilers to enable inlining of PLT (procedure linkage table) entries in function calls to functions that are not known to bind locally. This option has no effect unless the `-mfdpic` option is also specified. This option is enabled by default if the `-fPIC` or `-fpic` options are being used, or when optimization options such as `-O3` or above are specified.

`-mlibrary-pic`: Specifying this option causes the GCC compilers to generate position-independent extended application binary interface (EABI) code.

`-mlinked-fp`: Specifying this option causes the GCC compilers to conform to the EABI requirement of always creating a frame pointer whenever a stack frame is allocated. This option is enabled by default and can be disabled with `-mno-linked-fp`.

`-mlong-calls`: Specifying this option causes the GCC compilers to use indirect addressing to call functions outside the current compilation unit, which enables functions to be placed anywhere within the 32-bit address space.

`-mmedia`: Specifying this option enables the GCC compilers to use media instructions.

`-mmuladd`: Specifying this option causes the GCC compilers to use the multiply and add/subtract instructions.

`-mmulti-cond-exec`: Specifying this option causes the GCC compilers to optimize the `&&` and `||` operations in conditional execution and is active by default. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mnested-cond-exec`: Specifying this option causes the GCC compilers to enable nested conditional execution optimizations and is active by default. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mno-cond-exec`: Specifying this option prevents the GCC compilers from using conditional execution. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mno-cond-move`: Specifying this option prevents the GCC compilers from using conditional-move instructions. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mno-double`: Specifying this option prevents the GCC compilers from using double-word floating-point instructions.

`-mno-dword`: Specifying this option prevents the GCC compilers from using double-word instructions.

`-mno-eflags`: Specifying this option prevents the GCC compilers from marking ABI switches in `e_flags`.

`-mno-media`: Specifying this option prevents the GCC compilers from using media instructions.

`-mno-muladd`: Specifying this option prevents the GCC compilers from using multiply and add/subtract instructions.

`-mno-multi-cond-exec`: Specifying this option prevents the GCC compilers from optimizing the `&&` and `||` operations in conditional execution. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mno-nested-cond-exec`: Specifying this option prevents the GCC compilers from optimizing nested conditional execution. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mno-optimize-membar`: Specifying this option prevents the GCC compilers from automatically removing redundant membar instructions from the code that they generate.

`-mno-pack`: Specifying this option prevents the GCC compilers from packing VLIW instructions.

`-mno-scc`: Specifying this option prevents the GCC compilers from using conditional set instructions. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-mno-vliw-branch`: Specifying this option prevents the GCC compilers from running a pass to pack branches into VLIW instructions. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-moptimize-membar`: Specifying this option causes the GCC compilers to remove redundant membar instructions from the code that they generate. This option is enabled by default.

`-mpack`: Specifying this option causes the GCC compilers to pack VLIW instructions. This option is the default.

`-mscc`: Specifying this option causes the GCC compilers to use conditional set instructions and is active by default. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

`-msoft-float`: Specifying this option causes the GCC compilers to use library routines for floating-point operations.

`-mTLS`: Specifying this option causes the GCC compilers to assume a large TLS (thread-local storage) segment when generating thread-local code.

`-mtls`: Specifying this option prevents the GCC compilers from assuming that a large TLS segment is being used when generating thread-local code.

`-mtomcat-stats`: Specifying this option causes the GCC compilers to cause the GNU assembler to print Tomcat processor statistics.

`-multilib-library-pic`: Specifying this option causes the GCC compilers to link with the libraries that support position-independent code, and is implied by the `-mlibrary-pic` option, as well as by `-fPIC` and `-fpic` unless the `-mfdpic` option is specified. You should never have to explicitly supply this option.

`-mvliw-branch`: Specifying this option causes the GCC compilers to run an extra pass to pack branches into VLIW instructions and is active by default. This option is primarily used for debugging the compiler itself and may be removed in future versions of GCC.

H8/300 Options

H8 is a family of 8-bit microprocessors featuring an H8/300 CPU core and a variety of on-chip supporting modules that provide a variety of system functions. Often used as microcontrollers in embedded environments, H8/300-based microprocessors are available from Hitachi, SuperH, and now Renesas.

GCC options available when compiling code for H8/300-based systems are the following:

`-malign-300`: Specifying this option when compiling for the H8/300H and the H8/S processors tell GCC to use the same alignment rules as for the H8/300. The default for the H8/300H and H8/S is to align `longs` and `floats` on 4-byte boundaries. Specifying the `-malign-300` option causes `longs` and `floats` to be aligned on 2-byte boundaries. This option has no effect on the H8/300.

`-mh`: Specifying this option tells GCC to generate code for the H8/300H processor.

`-mint32`: Specifying this option tells GCC to make `int` data 32 bits long by default, rather than 16 bits long.

`-mn`: Specifying this option tells GCC to generate code for the H8S and H8/300 processors in normal mode. Either the `-mh` or the `-ms` switches must also be specified when using this option.

`-mrelax`: Specifying this option tells GCC to shorten some address references at link time, when possible, by passing the `-relax` option to ld.

`-ms`: Specifying this option tells GCC to generate code for the H8/S processor.

`-ms2600`: Specifying this option tells GCC to generate code for the H8/S2600 processor. The `-ms` option must also be specified when using this option.

HP/PA (PA/RISC) Options

HP/PA stands for Hewlett-Packard Precision Architecture, the original name for what are now commonly referred to as Precision Architecture, Reduced Instruction Set Computing (PA-RISC) systems. PA-RISC is a microprocessor architecture developed by Hewlett-Packard's Systems and VLSI Technology Operation and owes some of its design to the RISC technologies introduced in Apollo's DN10000 RISC systems. Apollo was consumed by HP in the late 1980s, a sad time for us all. PA-RISC CPUs are used in many later HP workstations.

GCC options available when compiling code for PA-RISC systems are the following:

`-march=`*`architecture-type`*: Specifying this option tells GCC to generate code for the specified architecture. The choices for *`architecture-type`* are 1.0 for PA 1.0, 1.1 for PA 1.1, and 2.0 for PA 2.0 processors. The file /usr/lib/sched.models on an HP-UX system identifies the proper architecture option for specific machines. Code compiled for lower numbered architectures will run on higher numbered architectures, but not the other way around.

`-mbig-switch`: Specifying this option tells GCC to generate code suitable for big switch tables. You should only use this option if the assembler or linker complains about branches being out of range within a switch table.

`-mdisable-fpregs`: Specifying this option tells GCC to prevent floating-point registers from being used. This option is used when compiling kernels that perform lazy context switching of floating-point registers. GCC will abort compilation if you use this option and attempt to perform floating-point operations in the application that you are compiling.

`-mdisable-indexing`: Specifying this option tells GCC not to use indexing address modes. You should only use this option if you are running GCC on a PA-RISC system running Mach—and what are the chances of that?

`-mfast-indirect-calls`: Specifying this option tells GCC to generate code that assumes calls never cross space boundaries. This enables GCC to generate code that performs faster indirect calls. This option will not work in nested functions or when shared libraries are being used.

`-mhp-ld`: Specifying this option tells GCC to use options specific to the HP linker. For example, this passes the `-b` option to the linker when building shared libraries. This option does not have any effect on the linker itself, just on the options that are passed to it. This option is only available on 64-bit versions of GCC.

`-mgas`: Specifying this option enables GCC to use assembler directives that are only understood by the GNU assembler. This option should therefore not be used if you are using the HP assembler with GCC.

`-mgnu-ld`: Specifying this option tells GCC to use options specific to the GNU linker. For example, this passes the `-shared` option to ld when building a shared library. This option is only available on 64-bit versions of GCC.

`-mjump-in-delay`: Specifying this option tells GCC to fill the delay slots of function calls with unconditional jump instructions by modifying the return pointer for the function call to be the target of the jump.

`-mlinker-opt`: Specifying this option tells GCC to enable the optimization pass in the HP-UX linker. Optimization makes symbolic debugging impossible, but will provide improved performance in most cases. If you are using GCC on HP-UX 8 or 9 systems, using the HP-UX linker may display erroneous error messages when linking some programs.

`-mlong-calls`: Specifying this option causes GCC to use long call sequences in order to ensure that a function call is always able to reach the stubs generated by the linker.

`-mlong-load-store`: Specifying this option tells GCC to generate the three-instruction load and store sequences that are sometimes required by the HP-UX 10 linker. This option should be unnecessary if you are using the standard GNU linker/loader. This option is the same as the `+k` option provided by HP compilers.

`-mno-space-regs`: Specifying this option tells GCC to generate code that assumes the target has no space registers. This option enables GCC to generate faster indirect calls and use unscaled index address modes. Typically, this option should only be used on PA-RISC 1.0 systems or when compiling a kernel.

`-mpa-risc-1-0`: A deprecated synonym for the `-march=1.0` option.

`-mpa-risc-1-1`: A deprecated synonym for the `-march=1.1` option.

`-mpa-risc-2-0`: A deprecated synonym for the `-march=2.0` option.

`-mportable-runtime`: Specifying this option tells GCC to use the portable calling conventions proposed by HP for ELF systems.

`-mschedule=`*`CPU-type`*: Specifying this option tells GCC to schedule code according to the constraints for the machine type `CPU-type`. Possible choices for *`CPU-type`* are `700`, `7100`, `7100LC`, `7200`, and `8000`. The file /usr/lib/sched.models on an HP-UX system shows the proper scheduling option for your machine. The default value is `8000`.

`-msio`: Specifying this option causes GCC to generate the predefined symbol `_SIO` (server IO). See the `-mwsio` option for related predefines.

`-msoft-float`: Specifying this option tells GCC to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC, but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system, or these function calls will be identified as unresolved.

■ **Tip** Specifying the `-msoft-float` option changes the calling conventions in the output file. All of the modules of a program must be compiled using this option in order to be successfully linked. You will also need to compile libgcc.a, the library that comes with GCC, with `-msoft-float` in order to use this option when compiling applications.

`-munix=`*`UNIX-STD`*: Specifying this option causes GCC to generate compiler predefines and select the appropriate start file for the specified *`UNIX-STD`*. Possible values for *`UNIX-STD`* are `93`, `95`, and `98`. The value `93` is supported on all versions of HP-UX, the value `95` is only supported on HP-UX 10.10 and later, and the value `98` is only supported on HP-UX 11.11 and later. The default values for UNIX-STD are `93` on HP-UX 10.0, `95` on HP-UX 10.10 through 11.00, and `98` for HP-UX 11.11 and later releases. Specifying `-munix=93` provides the same predefines as GCC 3.3 and 3.4. Specifying `-munix=95` provides additional predefines for `XOPEN_UNIX` and `_XOPEN_SOURCE_EXTENDED`, and uses the startfile unix95.o. Specifying `-munix=98` provides additional predefines for `_XOPEN_UNIX`, `_XOPEN_SOURCE_EXTENDED`, `_INCLUDE__STDC_A1_SOURCE`, and `_INCLUDE_XOPEN_SOURCE_500`, and uses the startfile unix98.o.

`-mwsio`: Specifying this option causes GCC to generate the predefines `__hp9000s700`, `__hp9000s700__`, and `_WSIO` (workstation IO). This option is the default and can be used on both HP-IX and HP-UX systems. (This option is not available in the GCC 4.x compilers.)

`-static`: Specifying this standard GCC option evokes slightly different behavior on HP-UX systems than when compiling static binaries for other platforms. On HP-UX systems, specifying this option also passes specific options to the linker to resolve a dependency in the `setlocale()` call on `libdld.sl`, which causes the resulting binary to be dynamic. The `-nolibdld` option can be specified to prevent GCC from passing these options to the linker.

`-threads`: Specifying this option causes GCC to add support for multithreading, using the DCE (Distributed Computing Environment) thread library. Specifying this option sets other options for both the preprocessor and the linker.

i386 and AMD x86-64 Options

The Intel x86 family of 32-bit processors, commonly referred to as *i386 processors*, is the best known and most widely used type of processor found in modern desktop computers. It is easy to argue their merits against other popular processors such as Apple's PPC G4 and G5 processors, but it is impossible to argue with their ubiquity. AMD64 is AMD's family of compatible 64-bit processors.

The options in this section can be used when compiling any i386 or x86-64 code. For the discussion of options that can only be used on AMD x86-64 systems, see the section earlier in this appendix titled "AMD x86-64 Options."

GCC options available when compiling code for all i386 and 64-bit AMD systems are the following:

`-m128bit-long-double`: Specifying this option tells GCC to use 128 bits to store `long doubles`, rather than 12 bytes (96 bits) as specified in the i386 application binary interface. Pentium and newer processors provide higher performance when `long doubles` are aligned to 8- or 16-byte boundaries, though this is impossible to reach when using 12-byte `long doubles` for array access. If you specify this option, any structures and arrays containing `long doubles` will change size, and function calling conventions for functions taking `long double` arguments will also be modified.

`-m32`: Specifying this option tells GCC to generate code for a 32-bit environment.

`-m386`: This option is a deprecated synonym for the `-mtune=i386` option.

`-m3dnow`: Specifying this option enables the use of built-in functions that allow direct access to the 3DNow extensions of the instruction set. (3DNow is a multimedia extension added by AMD to enhance graphics performance.)

`-m486`: This option is a deprecated synonym for the `-mtune=i486` option.

`-m64`: Specifying this option tells GCC to generate code for a 64-bit environment.

`-m96bit-long-double`: Specifying this option tells GCC to set the size of `long doubles` to 96 bits as required by the i386 application binary interface. This is the default.

`-maccumulate-outgoing-args`: Specifying this option tells GCC to compute the maximum amount of space required for outgoing arguments in the function prologue. This is faster on most modern CPUs because of reduced dependencies, improved scheduling, and reduced stack usage when the preferred stack boundary is not equal to 2. The downside is a notable increase in code size. Specifying this option implies the `-mno-push-args` option.

`-malign-double`: Specifying this option tells GCC to align `double`, `long double`, and `long long` variables on a two-word boundary. This produces code that runs somewhat faster on Pentium or better systems at the expense of using more memory. Specifying this option aligns structures in a way that does not conform to the published application binary interface specifications for the 80386 processors.

`-march=`*`CPU-type`*: Specifying this option tells GCC to generate instructions for the specified machine type `CPU-type`. Possible values for *`CPU-type`* are `athlon`, `athlon64`, `athlon-4`, `athlon-fx`, `athlon-tbird`, `athlon-mp`, `athlon-xp`, `c3`, `c3-2`, `i386`, `i486`, `i586` (equivalent to `pentium`), `i686` (equivalent to `pentiumpro`), `k6`, `k6-2`, `k6-3`, `k8`, `opteron`, `pentium`, `pentium-m`, `pentium-mmx`, `pentiumpro`, `pentium2`, `pentium3`, `pentium3m`, `pentium4`, `pentium4m`, `prescott`, `winchip-c6`, and `winchip2`. Specifying the `-march=`*`CPU-type`* option implies the `-mtune=`*`CPU-type`* option. See the later discussion of the `-mtune=`*`CPU-type`* option for a verbose explanation of all supported values for *`CPU-type`*.

`-masm=`*`dialect`*: Specifying this option tells GCC to output asm instructions using the selected *dialect*, which can be either `intel` or `att`. The `att` dialect is the default.

`-mcmodel=`*`model`*: Specifying this option causes GCC to generate code for the indicated memory model. Valid values for *model* are `small` (the program and all symbols must be linked in the lower 2GB of address space and use 64-bit pointers), `kernel` (for programs such as the Linux kernel, which run in the negative 2GB of the address space), `medium` (the program must be linked in the lower 2GB of address space but symbols can be anywhere in the address space—this model can not be used to build shared libraries), and `large` (anything can be anywhere in the address space, though GCC does not currently support this code model).

`-mcpu=`*`CPU-type`*: This option is a deprecated synonym for the `-mtune=`*`CPU-type`* option.

`-mfpmath=`*`unit`*: Specifying this option tells GCC to generate floating-point arithmetic for the floating-point unit *`unit`*. Valid choices for *`unit`* are the following:

- `387`: Use the standard 387 floating-point coprocessor present in most i386 chips and emulated otherwise. Code compiled with this option will run almost everywhere. The temporary results are computed in 80-bit precision instead of precision specified by the type, resulting in slightly different results compared to most other chips. See the standard GCC option `-ffloat-store` for more information. This is the default if the `-mfpmath` option is not specified.

- `sse`: Use scalar floating-point instructions present in the Streaming SIMD Extensions (SSE) instruction set. This instruction set is supported by Intel Pentium III and newer chips, and in the Athlon 4, Athlon XP, and Athlon MP chips from AMD. Earlier versions of the SSE instruction set only supported single-precision arithmetic, which means that double- and extended-precision math is still done using the standard 387 instruction set. The versions of the SSE instruction set provided in Pentium 4 and AMD x86-64 chips provide direct support for double-precision arithmetic. If you are compiling for chips other than these, you should use the `-march=`*`CPU-type`*, `-msse`, and `-msse2` options to enable SSE extensions and to use this option effectively. The `-mfpmath=sse` option is the default for GCC compiled for the x86-64 target.
- `sse,387`: Attempt to utilize both instruction sets at once. This effectively doubles the number of available registers, as well as the amount of execution resources on chips with separate execution units for 387 and SSE. Using this option may cause problems because the GCC register allocator does not always model separate functional units well.

`-mieee-fp`: Specifying this option tells GCC to use IEEE floating-point comparisons, which correctly handle the case where the result of a comparison is unordered.

`-minline-all-stringops`: Specifying this option tells GCC to inline all string operations. By default, GCC only inlines string operations when the destination is known to be aligned to at least a 4-byte boundary. This enables more inlining, which increases code size, but may also improve the performance of code that depends on fast `memcpy()`, `strlen()`, and `memset()` for short lengths.

`-mlarge-data-threshold=`*`number`*: Specifying this option when `-mcmodel=medium` is used tells GCC that data greater than *`number`* should be placed in a large data section. The value for *`number`* must be the same across all of the object code that you are linking together and defaults to `65535`.

`-mmmx`: Specifying this option enables the use of built-in functions that allow direct access to the MMX extensions of the instruction set.

`-mno-3dnow`: Specifying this option disables the use of built-in functions that allow direct access to the 3DNow extensions of the instruction set.

`-mno-align-double`: Specifying this option tells GCC not to align `double`, `long double`, and `long long` variables on a two-word boundary, using a one-word boundary instead. This reduces memory consumption but may result in slightly slower code. Specifying this option aligns structures containing these datatypes in a way that conforms to the published application binary interface specifications for the 80386 processor.

`-mno-align-stringops`: Specifying this option tells GCC not to align the destination of inlined string operations. This switch reduces code size and improves performance when the destination is already aligned.

`-mno-fancy-math-387`: Specifying this option tells GCC to avoid generating the sin, cos, and sqrt instructions for the 80387 floating-point units because these instructions are not provided by all 80387 emulators. This option has no effect unless you also specify the `-funsafe-math-optimizations` option. This option is overridden when `-march` indicates that the target CPU will always have an FPU and so the instruction will not need emulation. This option is the default for GCC on FreeBSD, OpenBSD, and NetBSD systems.

`-mno-fp-ret-in-387`: Specifying this option tells GCC not to use the FPU registers for function return values, returning them in ordinary CPU registers instead. The usual calling convention on 80386 systems returns `float` and `double` function values in an FPU register, even if there is no FPU. This assumes that the operating system provides emulation support for an FPU, which may not always be correct.

`-mno-ieee-fp`: Specifying this option tells GCC not to use IEEE floating-point comparisons.

`-mno-mmx`: Specifying this option disables the use of built-in functions that allow direct access to the MMX extensions of the instruction set.

`-mno-push-args`: Specifying this option tells GCC to use standard `SUB/MOV` operations to store outgoing parameters, rather than the potentially smaller `PUSH` operation. In some cases, using `SUB/MOV` rather than `PUSH` may improve performance because of improved scheduling and reduced dependencies.

`-mno-red-zone`: Specifying this option tells GCC not to use the red zone mandated by the x86-64 ABI. The red zone is a 128-byte area beyond the stack pointer that will not be modified by signal or interrupt handles, and can therefore be used for temporary data storage without adjusting the stack pointer.

`-mno-sse`: Specifying this option disables the use of built-in functions that allow direct access to the SSE extensions of the instruction set.

`-mno-sse2`: Specifying this option disables the use of built-in functions that allow direct access to the SSE2 extensions of the instruction set.

`-mno-svr3-shlib`: Specifying this option tells GCC to place uninitialized local variables in the data segment. This option is only meaningful on System V Release 3 (SVR3) systems.

`-mno-tls-direct-seg-refs`: Specifying this option enables GCC to add the thread base pointer when accessing thread-local storage variables through offsets from the TLS segment. This is necessary if the operating system that you are compiling for does not map the TLS segment to cover the entire TLS area.

`-momit-leaf-frame-pointer`: Specifying this option tells GCC not to keep the frame pointer in a register for leaf functions. This avoids generating the instructions to save, set up, and restore frame pointers and makes an extra register available in leaf functions. The option `-fomit-frame-pointer` removes the frame pointer for all functions, which might make debugging harder.

`-mpentium`: This option is a deprecated synonym for the `-mtune=i586` and `-mtune=pentium` options.

`-mpentiumpro`: This option is a deprecated synonym for the `-mtune=i686` and `-mtune=pentiumpro` options.

`-mpreferred-stack-boundary=`*num*: Specifying this option tells GCC to attempt to keep the stack boundary aligned to a 2**num* byte boundary. This extra alignment consumes extra stack space and generally increases code size. Code that is sensitive to stack space usage, such as embedded systems and operating system kernels, may want to reduce the preferred alignment to `-mpreferred-stack-boundary=2`. If the `-mpreferred-stack-boundary` option is not specified, the default is 4 (16 bytes or 128 bits), except when optimizing for code size (using the `-Os` option), in which case the default is the minimum correct alignment (4 bytes for x86, and 8 bytes for x86-64).

On Pentium and Pentium Pro systems, `double` and `long double` values should be aligned to an 8-byte boundary (see the `-malign-double` option), or they will incur significant runtime performance penalties. On Pentium III systems, the SSE datatype `__m128` incurs similar penalties if it is not 16-byte aligned.

■ **Note** To ensure proper alignment of the values on the stack, the stack boundary must be as aligned as required by any value stored on the stack. Therefore, every function and library must be generated such that it keeps the stack aligned. Calling a function compiled with a higher preferred stack boundary from a function compiled with a lower preferred stack boundary will probably misalign the stack. The GNU folks recommend that libraries that use callbacks always use the default setting.

`-mpush-args`: Specifying this option tells GCC to use the `PUSH` operation to store outgoing parameters. This method is shorter and usually at least as fast as using `SUB/MOV` operations. This is the default.

`-mregparm=`*num*: Specifying this option controls the number of registers used to pass integer arguments, specifid as *num*. By default, no registers are used to pass arguments, and at most, three registers can be used. You can also control this behavior for a specific function by using the function attribute `regparm`.

■ **Note** If you use this switch to specify a nonzero number of registers, you must build all modules with the same value, including any libraries that the program uses. This includes system libraries and startup modules.

`-mrtd`: Specifying this option tells GCC to use a function-calling convention where functions that take a fixed number of arguments return with the `ret(`*num*`)` instruction, which pops their arguments during the return. This saves one instruction in the caller because there is no need to pop the arguments there. This calling convention is incompatible with the one normally used on Unix, and therefore cannot be used if you need to call libraries that have been compiled with generic Unix compilers. When using this option, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf()`); if you do not, incorrect code will be generated for calls to those functions. You must also be careful to never call functions with extra arguments, which would result in seriously incorrect code. This option takes its name from the 680x0 rtd instruction.

■ **Tip** To optimize heavily used functions, you can specify that an individual function is called with the calling sequence indicated by the `-mrtd option` by using the function attribute `stdcall`. You can also override the `-mrtd` option for specific functions by using the function attribute `cdecl`.

`-msoft-float`: Specifying this option tells GCC to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system, or these function calls will be identified as unresolved.

■ **Note** On machines where a function returns floating-point results in the 80387 register stack, some floating-point opcodes may be emitted even if `-msoft-float` is used.

`-msse`: Specifying this option enables the use of built-in functions that allow direct access to the SSE extensions of the instruction set.

`-msse2`: Specifying this option enables the use of built-in functions that allow direct access to the SSE2 extensions of the instruction set.

`-msselibm`: Specifying this option causes GCC to use special versions of certain math library (`libm`) options that come with an SSE ABI implementation. This option is useful with the `-mfpmath=sse` option to avoid moving values between SSE registers and the i387 floating-point stack.

`-msseregparm`: Specifying this option causes GCC to use the SSE register passing conventions for floating-point and `double` arguments and return values. If you use this option, all object code and libraries that will be linked or used together must also be compiled with this option.

`-msvr3-shlib`: Specifying this option tells GCC to place uninitialized local variables in the bss segment. This option is only meaningful on System V Release 3 (SVR3) systems.

`-mthreads`: Specifying this option tells GCC to support thread-safe exception handling on Mingw32 platforms. Code that relies on thread-safe exception handling must compile and link all code with this option. When compiling, specifying the `-mthreads` option sets the `-D_MT` symbol. When linking, specifying this option links in a special thread helper library (equivalent to the `-lmingwthrd` option) that cleans up per-thread exception-handling data.

`-mtls-direct-seg-refs`: Specifying this option enables GCC to access thread-local storage variables through offsets from the TLS segment, without adding the thread base pointer. Whether or not this is valid depends on whether the operating system that you are compiling for maps the TLS segment to cover the entire TLS area. This option is active by default when using Glibc.

`-mtune=`*`CPU-type`*: Specifying this option tells GCC to generate object code that is tuned to and scheduled for the specified *`CPU-type`*, with the exception of the ABI and the set of available instructions. (The instruction set is selected using the `-march=`*`CPU-type`* option.) Possible values for *`CPU-type`* are the following (organized alphabetically):

- `athlon`, `athlon-tbird`: AMD Athlon CPUs with MMX, 3DNow, enhanced 3DNow, and SSE prefetch instruction set support.
- `athlon-4`, `athlon-xp`, `athlon-mp`: Improved AMD Athlon CPUs with MMX, 3DNow, enhanced 3DNow, and full SSE instruction set support.
- `c3`: VIA C3 CPU with MMX and 3DNow instruction set support.
- `c3-2`: VIA C3-2 CPU with MMX and SSE instruction set support.
- `generic`: IA32/AMD64/EM64T processors. You should only use this option if you do not know the specific `CPU-type` on which people will be running your code. Note that there is no corresponding `-march=generic` option, because the `-march` option identifies the instruction set the compiler can use, and no generic instruction set is applicable to all processors.
- `i386`: Original Intel i386 CPU.
- `i486`: Intel's i486 CPU.
- `i586`, `pentium`: Intel Pentium CPU without MMX support.
- `i686`: All processors in the i686 family. The same as `generic`, except that when this option is specified with `-march=i686`, the resulting code will use the Pentium Pro instruction set.
- `k6`: AMD K6 CPU with MMX instruction set support.

- `k6-2`, `k6-3`: Improved versions of the AMD K6 CPU with MMX and 3DNow instruction set support.
- `k8`, `opteron`, `athlon64`, `athlon-fx`: CPUs based on the AMD K8 core with x86-64 instruction set support, which provides a superset of the MMX, SSE, SSE2, 3DNow, enhanced 3DNow, and 64-bit instruction set extensions.
- `nocona`: Improved version of the Intel Pentium 4 CPU with 64-bit extensions, MMX, SSE, SSE2, and SSE3 instruction set support.
- `pentium-m`: Low-power version of the Intel Pentium 3 CPU with MMX, SSE, and SSE2 instruction set support.
- `pentium-mmx`: Intel Pentium core CPU with MMX instruction set support.
- `pentiumpro`: Intel Pentium Pro CPU.
- `pentium2`: Intel Pentium 2 CPU based on the Pentium Pro core, but with MMX instruction set support.
- `pentium3`, `pentium3m`: Intel Pentium 3 CPUs based on the Pentium Pro core, but with MMX and SSE instruction set support.
- `pentium4`, `pentium4m`: Intel Pentium 4 CPUs with MMX, SSE, and SSE2 instruction set support.
- `prescott`: Improved version of the Intel Pentium 4 CPU with MMX, SSE, SSE2, and SSE3 instruction set support.
- `winchip-c6`: IDT WinChip C6 CPU, which is essentially similar to i486 with additional MMX instruction set support.
- `winchip2`: IDT WinChip 2 CPU, which is essentially similar to i486 with additional MMX and 3DNow instruction set support.

Note Picking a specific *CPU-type* using the `-mtune=`*CPU-type* option will schedule the code appropriately for that particular chip, but GCC will not generate any code that does not run on the i386 without the `-march=`*CPU-type* option being used to identify a particular *CPU-type*.

IA-64 Options

The options in this section can only be used when compiling code targeted for 64-bit Intel processors (the GCC ia-64 build target). This does not include EMT64 chips, which are supported as a subset of the x86-64 build target. For a discussion of options that can be used on i386 systems, see the section earlier in this appendix titled "i386 and AMD x86-64 Options." For a discussion of options that can only be used on AMD64 (64-bit AMD) systems, see the section earlier in this appendix titled "AMD x86-64 Options."

GCC options available when compiling code for 64-bit Intel systems are the following:

`-mauto-pic`: Specifying this option tells GCC to generate position-independent code (i.e., code that is self-relocatable). Specifying this option implies the `-mconstant-gp` option. This option is useful when compiling firmware code.

`-mb-step`: Specifying this option tells GCC to generate code that works around Itanium B step errata.

`-mbig-endian`: Specifying this option tells GCC to generate code for a big endian target. This is the default for HP-UX systems using the IA-64 processor.

`-mconstant-gp`: Specifying this option tells GCC to generate code that uses a single constant global pointer value. This is useful when compiling kernel code because all pointers are offset from a single global.

`-mdwarf2-asm`: Specifying this option tells GCC to generate assembler code for DWARF2 line number debugging. This information may be useful when not using the GNU assembler.

`-mearly-stop-bits`: Specifying this option enables GCC to put stop bits in object code earlier than immediately preceding the instruction that generated the stop bits. This may help with scheduling.

`-mfixed-range=`*`register-range`*: Specifying this option tells GCC to generate code that treats the specified range of registers as fixed registers. A *register range* is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma. A fixed register is one that the register allocator cannot use. This option is useful when compiling kernel code.

`-mgnu-as`: Specifying this option tells GCC to generate code for the GNU assembler. This is the default.

`-mgnu-ld`: Specifying this option tells GCC to generate code for the GNU linker. This is the default.

`-milp32`: Specifying this option causes GCC to generate code for a 32-bit environment. This option is specific to the HP-UX platform.

`-milp64`: Specifying this option causes GCC to generate code for a 64-bit environment. This option is specific to the HP-UX platform.

`-minline-float-divide-max-throughput`: Specifying this option tells GCC to generate code for inline floating-point division using a maximum throughput algorithm.

`-minline-float-divide-min-latency`: Specifying this option tells GCC to generate code for inline floating-point division using a minimum latency algorithm.

`-minline-int-divide-max-throughput`: Specifying this option tells GCC to generate code for inline integer division using a maximum throughput algorithm.

`-minline-int-divide-min-latency`: Specifying this option tells GCC to generate code for inline integer division using a minimum latency algorithm.

`-minline-sqrt-max-throughput`: Specifying this option tells GCC to generate inline code for square root calculations using a maximum throughput algorithm.

`-minline-sqrt-min-latency`: Specifying this option tells GCC to generate inline code for square root calculations using a minimum latency algorithm.

`-mlittle-endian`: Specifying this option tells GCC to generate code for a little endian target. This is the default for AIX 5 and Linux systems using the IA-64 processor.

`-mno-dwarf2-asm`: Specifying this option tells GCC not to generate assembler code for DWARF2 line number debugging.

`-mno-early-stop-bits`: Specifying this option prevents GCC from putting stop bits in object code anywhere other than immediately preceding the instruction that generated the stop bits.

`-mno-gnu-as`: Specifying this option tells GCC not to generate code for the GNU assembler, but to assume that assembly will be done using a system assembler.

`-mno-gnu-ld`: Specifying this option tells GCC not to generate code for the GNU linker, but to assume linking/loading will be done using a system linker.

`-mno-inline-float-divide`: Specifying this option prevents GCC from generating code for inline floating point division.

`-mno-inline-int-divide`: Specifying this option prevents GCC from generating code for inline integer division.

`-mno-pic`: Specifying this option tells GCC to generate code that does not use a global pointer register. This results in code that is not position-independent and violates the IA-64 ABI.

`-mno-register-names`: Specifying this option tells GCC not to generate `in`, `loc`, and `out` register names for the stacked registers.

`-mno-sched-ar-data-spec`: Specifying this option tells GCC to disable data speculative scheduling after reload using the ld.a and associated chk.a instructions.

`-mno-sched-ar-in-data-spec`: Specifying this option tells GCC to disable speculative scheduling of the instructions that are dependent on data-speculative scheduling loads after reload.

`-mno-sched-br-data-spec`: Specifying this option tells GCC to disable data-speculative scheduling before reload using the ld.a and associated chk.a instructions.

`-mno-sched-br-in-data-spec`: Specifying this option tells GCC to disable speculative scheduling of the instructions that are dependent on data-speculative scheduling loads before reload.

`-mno-sched-contol-ldc`: Specifying this option tells GCC to disable the use of the ld.c instruction to check control-speculative loads.

`-mno-sched-control-spec`: Specifying this option tells GCC to disable control-speculative scheduling using the ld.s and associated chk.s instructions.

`-mno-sched-count-spec-in-critical-path`: Specifying this option tells GCC not to consider speculative dependencies when computing instruction priorities.

`-mno-sched-in-control-spec`: Specifying this option tells GCC to disable speculative scheduling of instructions that are dependent on control-speculative scheduling of load instructions.

`-mno-sched-ldc`: Specifying this option tells GCC to disable simple data speculation checks using the ld.c instruction and to only use the chk.c instruction.

`-mno-sched-prefer-non-control-spec-insns`: Specifying this option enables GCC to use control-speculative scheduling instructions whenever possible.

`-msched-prefer-non-data-spec-insns`: Specifying this option enables GCC to use data-speculative scheduling instructions whenever possible.

`-mno-sched-spec-verbose`: Specifying this option tells GCC not to display information about speculative scheduling.

`-mno-sdata`: Specifying this option tells GCC to disable optimizations that use the small data section. This option may be useful for working around optimizer bugs.

`-mno-volatile-asm-stop`: Specifying this option tells GCC not to generate a stop bit immediately before and after volatile `asm` statements.

`-mt`: Specifying this option sets flags for the preprocessor and linker that add support for multi-threading using the POSIX threading library. This option is specific to the HP-UX platform.

`-mregister-names`: Specifying this option tells GCC to generate `in`, `loc`, and `out` register names for the stacked registers. This may make assembler output more readable.

`-msched-ar-data-spec`: Specifying this option tells GCC to use data-speculative scheduling after reload using the ld.a and associated chk.a instructions. This option is enabled by default.

`-msched-ar-in-data-spec`: Specifying this option tells GCC to perform speculative scheduling of the instructions that are dependent on data-speculative scheduling load after reload, and is only useful if `-msched-ar-data-spec` is also specified. This option is enabled by default.

`-msched-br-data-spec`: Specifying this option tells GCC to use data-speculative scheduling before reload using the ld.a and associated chk.a instructions. This option is disabled by default.

`-msched-br-in-data-spec`: Specifying this option tells GCC to perform speculative scheduling of the instructions that are dependent on data-speculative scheduling load before reload, and is only useful if `-msched-br-data-spec` is also specified. This option is enabled by default.

`-msched-contol-ldc`: Specifying this option tells GCC to use the ld.c instruction to check control-speculative loads. If no speculatively scheduled dependent instructions follow the load, ld.sa is used instead, and ld.c is used to check it. This option is disabled by default.

`-msched-control-spec`: Specifying this option tells GCC to enable control-speculative scheduling of loads using the ld.s and associated chk.s instructions. This option is disabled by default.

`-msched-count-spec-in-critical-path`: Specifying this option tells GCC to consider speculative dependencies when computing instruction priorities. This option is disabled by default.

`-msched-in-control-spec`: Specifying this option tells GCC to enable speculative scheduling instructions that are dependent on control-speculative loads, and is only useful if `-msched-control-spec` is also specified. This option is enabled by default.

`-msched-ldc`: Specifying this option tells GCC to enable simple data-speculation checks using the ld.c instruction. This option is enabled by default.

`-msched-prefer-non-control-spec-insns`: Specifying this option tells GCC to only use control-speculative scheduling instructions when no other choices exist. This causes control speculation to be much more conservative. This option is disabled by default.

`-msched-prefer-non-data-spec-insns`: Specifying this option tells GCC to only use data-speculative scheduling instructions when no other choices exist. This causes data speculation to be much more conservative. This option is disabled by default.

`-msched-spec-verbose`: Specifying this option tells GCC to display information about speculative scheduling.

`-msdata`: Specifying this option tells GCC to enable optimizations that use the small data section. Though this option provides performance improvements, problems have been reported when using these optimizations.

`-mtls-size=`*`tls-tls`*: Specifying this option tells GCC to set the size of immediate TLS in bits to *`tls-size`*. Valid values are `14`, `22`, and `64`.

`-mtune=`*`CPU-type`*: Specifying this option tells GCC to generate object code that is tuned to and scheduled for the specified `CPU-type`. Possible values for *`CPU-type`* are the following (organized alphabetically):

- `itanium`: Intel's original Itanium IA-64 processor, with a clock speed of 733MHz, a ten-stage execution pipeline, and up to 4MB of L3 cache
- `itanium1`: a synonym for the Itanium
- `itanium2`: Intel's second-generation Itanium process, with clock speeds of 900Mhz to 1.7GHz, a shorter, seven-stage execution pipeline, and up to 9MB of L3 cache
- `mckinley`: a synonym for the Itanium 2
- `merced`: a synonym for the Itanium

`-mvolatile-asm-stop`: Specifying this option tells GCC to generate a stop bit immediately before and after volatile `asm` statements.

`-pthread`: Specifying this option sets flags for the preprocessor and linker that add support for multithreading using the POSIX threading library. This option is specific to the HP-UX platform.

Intel 960 Options

Intel's i960 family consists of high-performance, 32-bit embedded RISC processors supported by an outstanding selection of development tools (such as the one that you're reading about). Intel's i960 processors were often used in high-performance, embedded networking and imaging scenarios.

■ **Note** Support for this processor family is not provided in GCC version 4.0. These options are therefore only of interest if you are using a version of GCC that is earlier than 4.x and that you are certain provides support for this processor family.

GCC options available when compiling code for Intel 960 systems are the following:

`-masm-compat` | `-mintel-asm`: Specifying either of these options tells GCC to enable compatibility with the iC960 assembler.

`-mcode-align`: Specifying this option tells GCC to align code to 8-byte boundaries in order to support faster fetching. This option is the default for C-series processors (as specified using the `-mcpu-CPU-type` option).

`-mcomplex-addr`: Specifying this option tells GCC to use a complex addressing mode to provide performance improvements. Complex addressing modes may not be worthwhile on the K-series processors, but they definitely are on the C-series processors. This option is the default for all processors (as specified using the `-mcpu=CPU-type` option) except for the CB and CC processors.

`-mcpu=CPU-type`: Specifying this option tells GCC to use the defaults for the machine type `CPU-type`, affecting instruction scheduling, floating-point support, and addressing modes. Possible values for `CPU-type` are `ka`, `kb`, `mc`, `ca`, `cf`, `sa`, and `sb`, which are different generations and versions of the i960 processor. The default is `kb`.

`-mic-compat`: Specifying this option tells GCC to enable compatibility with the iC960 version 2.0 or version 3.0 C compiler from Intel.

`-mic2.0-compat`: Specifying this option tells GCC to enable compatibility with the iC960 version 2.0 C compiler from Intel.

`-mic3.0-compat`: Specifying this option tells GCC to enable compatibility with the iC960 version 3.0 C compiler from Intel.

`-mleaf-procedures`: Specifying this option tells GCC to attempt to alter leaf procedures to be callable with the `bal` instruction as well as call. This will result in more efficient code for explicit calls when the bal instruction can be substituted by the assembler or linker, but less efficient code in other cases, such as calls via function pointers or when using a linker that does not support this optimization.

`-mlong-double-64`: Specifying this option tells GCC to implement the `long double` type as 64-bit floating-point numbers. If you do not specify this option, `long double`s are implemented by 80-bit floating-point numbers. This option is present because there is currently no 128-bit `long double` support. This option should only be used when using the `-msoft-float` option, for example, for soft-float targets.

`-mno-code-align`: Specifying this option tells GCC not to align code to 8-byte boundaries for faster fetching. This option is the default for all non–C-series implementations (as specified using the `-mcpu-CPU-type` option).

`-mno-complex-addr`: Specifying this option tells GCC not to assume that the use of a complex addressing mode is a win on this implementation of the i960. Complex addressing modes may not be worthwhile on the K-series processors. This option is the default for the CB and CC processors (as specified using the `-mcpu-CPU-type` option).

`-mno-leaf-procedures`: Specifying this option tells GCC to always call leaf procedures with the call instruction.

`-mno-strict-align`: Specifying this option tells GCC to permit unaligned accesses.

`-mno-tail-call`: Specifying this option tells GCC not to make additional attempts (beyond those of the machine-independent portions of the compiler) to optimize tail-recursive calls into branches. This is the default.

`-mnumerics`: Specifying this option tells GCC that the processor supports floating-point instructions.

`-mold-align`: Specifying this option tells GCC to enable structure-alignment compatibility with Intel's GCC release version 1.3 (based on GCC 1.37). It would be really sad if this option was still getting much use. This option implies the `-mstrict-align` option.

`-msoft-float`: Specifying this option tells GCC that floating-point support should not be assumed, and to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system, or these function calls will be identified as unresolved.

`-mstrict-align`: Specifying this option tells GCC not to permit unaligned accesses.

`-mtail-call`: Specifying this option tells GCC to make additional attempts (beyond those of the machine-independent portions of the compiler) to optimize tail-recursive calls into branches. You may not want to do this because the detection of cases where this is not valid is not totally complete.

M32C Options

Renesas M32C processor family is a family of 32-bit RISC microprocessors designed for embedded systems. M32Cs are typically used in industrial, automotive, and communications systems.

GCC options available when compiling code for M32C systems are the following:

`-mcpu=`*name*: Specifying this option identifies the CPU for which GCC should generate code. Possible values for *name* are `r8c` for the R8C/Tiny series, `m16c` for the M16C (up to /60) series, `m32cm` for the M16C/80 series, and `m32c` for the M32C/80 series.

`-msim`: Specifying this option tells GCC that the code that it generates will be run on the simulator. This causes GCC to link in an alternate runtime library that supports capabilities such as file I/O. Do not use this option when generating programs that will run on real hardware; you must provide your own runtime library for any I/O functions that you need.

`-memregs=`*number*: Specifying this option tells GCC the number of memory-based pseudoregisters that it can use in generated code. This allows you to use more registers than are physically available but trades off the costs of juggling physical registers against the costs of using memory instead of registers. All of the object modules that you link together must be compiled with the same value for this option, which means that you must not use this option with the standard GCC runtime libraries unless you compiled them with the same option.

M32R Options

The Renesas M32R processor family is a family of 32-bit RISC microprocessors designed for embedded systems. M32Rs are designed for general industrial and car-mounted systems, digital AV equipment, digital imaging equipment, and portable consumer products.

GCC options available when compiling code for M32R systems are the following:

`-G num`: Specifying this option tells GCC to put global and static objects less than or equal to `num` bytes into the small data or bss sections instead of the normal data or bss sections. The default value of `num` is 8. The `-msdata` option must be set to either `sdata` or `use` for this option to have any effect.

■ **Note** All modules should be compiled with the same `-G num` value. Compiling with different values of `num` may or may not work. If it does not, the linker will display an error message and exit.

`-m32r`: Specifying this option tells GCC to generate code for the generic M32R. This is the default.

`-m32r2`: Specifying this option tells GCC to generate code for the M32R/2 processor.

`-m32rx`: Specifying this option tells GCC to generate code for the M32R/X processor.

`-malign-loops`: Specifying this option causes GCC to align all loops to a 32-byte boundary. This option is disabled by default.

`-mbranch-cost=`*number*: Specifying this option tells GCC how to value branches when generating code. If *number* is 1, branches will be preferred over conditional code. If *number* is 2, conditional code will be preferred over branches.

`-mcode-model=large`: Specifying this option tells GCC to assume that objects may be anywhere in the 32-bit address space (GCC generates seth/add3 instructions to load the addresses of those objects), and to assume that subroutines may not be reachable with the bl instruction (GCC generates the much slower seth/add3/jl instruction sequence).

■ **Tip** The addressability of a particular object can be set with the `model` attribute.

`-mcode-model=medium`: Specifying this option tells GCC to assume that objects may be anywhere in the 32-bit address space (GCC generates seth/add3 instructions to load their addresses), and to assume that all subroutines are reachable with the bl instruction.

`-mcode-model=small`: Specifying this option tells GCC to assume that all objects live in the lower 16MB of memory (so that their addresses can be loaded with the ld24 instruction), and to assume that all subroutines are reachable with the bl instruction. This is the default.

`-mdebug`: Specifying this option causes GCC to display diagnostic information about the M32R-specific portions of the compiler.

`-mflush-func=`*`name`*: Specifying this option tells GCC the name of the operating system function to call in order to flush the cache. The default is `__flush_cache_`. This option is only meaningful if the `-mno-flush-trap` option has been specified.

`-mflush-trap=`*`number`*: Specifying this option tells GCC which trap number to use in order to flush the cache. The default is 12. Possible values for *number* are 0 through 15, inclusive.

`-missue-rate=`*`number`*: Specifying this option causes GCC to produce code that issues *number* instructions per cycle. Possible values for *number* are 1 or 2.

`-mno-align-loops`: Specifying this option prevents GCC from aligning all loops of 32-byte boundaries.

`-mno-flush-func`: Specifying this option tells GCC that there is no operating system function that can be used to flush the cache.

`-mno-flush-trap`: Specifying this option tells GCC that the cache cannot be flushed by using a trap.

`-msdata=none`: Specifying this option tells GCC to disable the use of the small data area. The small data area consists of the sections sdata and sbss. Variables will be put into one of the data, bss, or rodata sections (unless the `section` attribute has been specified). This is the default.

■ **Tip** Objects may be explicitly put in the small data area with the `section` attribute.

`-msdata=sdata`: Specifying this option tells GCC to put small global and static data in the small data area, but not generate special code to reference them.

`-msdata=use`: Specifying this option tells GCC to put small global and static data in the small data area, and to generate special instructions to reference them.

M680x0 Options

The options in this section are used when employing GCC to compile applications for use on systems that rely on the Motorola 68000 family of processors. These are the 68000, 68010, 68020, 68030, 68040, and 68060 processors. These legendary processors were used in almost all early computer workstations before the advent of RISC processors and are still frequently used as microcontrollers.

GCC options available when compiling code for M680x0 systems are the following:

`-m5200`: Specifying this option tells GCC to generate code for a 520X ColdFire family CPU. This is the default when the compiler is configured for 520X-based systems. You should use this option when compiling code for microcontrollers with a 5200 core, including the MCF5202, MCF5203, and MCF5204 processors. The `-m5200` option implies the `-mnobitfield` option.

`-m68000` | `-mc68000`: Specifying either of these options tells GCC to generate code for a 68000 processor. These are the default when the compiler is configured for 68000-based systems. You should use these options when compiling for microcontrollers with a 68000 or EC000 core, including the 68008, 68302, 68306, 68307, 68322, 68328, and 68356 processors. The `-m68000` option implies the `-mnobitfield` option.

`-m68020` | `-mc68020`: Specifying either of these options tells GCC to generate code for a 68020 processor. This is the default when the compiler is configured for 68020-based systems. Specifying these options also sets the `-mbitfield` option.

`-m68020-40`: Specifying this option tells GCC to generate code for a 68040 processor, without using any instructions introduced since the 68020. This results in code that can run relatively efficiently on 68020/68881, 68030, or 68040 systems. The generated code uses the 68881 instructions that are emulated on the 68040 processor.

`-m68020-60`: Specifying this option tells GCC to generate code for a 68060 processor, without using any instructions introduced since the 68020 processor. This results in code that can run relatively efficiently on 68020/68881, 68030, 68040, or 68060 systems. The generated code uses the 68881 instructions that are emulated on the 68060 processor.

`-m68030`: Specifying this option tells GCC to generate code for a 68030 processor. This is the default when the compiler is configured for 68030-based systems.

`-m68040`: Specifying this option tells GCC to generate code for a 68040 processor. This is the default when the compiler is configured for 68040-based systems. Specifying this option inhibits the use of 68881/68882 instructions that have to be emulated by software on the 68040 processor. You should use this option if your 68040 system does not have code to emulate those instructions.

`-m68060`: Specifying this option tells GCC to generate code for a 68060 processor. This is the default when the compiler is configured for 68060-based systems. This option inhibits the use of 68020 and 68881/68882 instructions that have to be emulated by software on the 68060 processor. You should use this option if your 68060 system does not have code to emulate those instructions.

`-m68881`: Specifying this option tells GCC to generate code containing 68881 instructions for floating point. This is the default for most 68020 systems unless the `--nfp` option was specified when GCC was configured.

`-malign-int`: Specifying this option tells GCC to align `int`, `long`, `long long`, `float`, `double`, and `long double` variables on a 32-bit boundary. Aligning variables on 32-bit boundaries produces code that runs somewhat faster on processors with 32-bit buses at the expense of more memory. Specifying this option aligns structures containing these datatypes differently than most published application binary interface specifications for the 68000 processor.

`-mbitfield`: Specifying this option tells GCC to use the bit-field instructions. The `-m68020` option implies this option. This is the default if you use a configuration designed for a 68020 processor.

`-mcfv4e`: Specifying this option tells GCC to generate code for processors in the ColdFire V4e family (547X, 548X), including hardware floating-point instructions. (This option is not available in the GCC 4.x compilers.)

`-mcpu32`: Specifying this option tells GCC to generate code for a CPU32 processor core. This is the default when the compiler is configured for CPU32-based systems. You should use this option when compiling code for microcontrollers with a CPU32 or CPU32+ core, including the 68330, 68331, 68332, 68333, 68334, 68336, 68340, 68341, 68349, and 68360 processors. The `-mcpu32` option implies the `-mnobitfield` option.

`-mfpa`: Specifying this option tells GCC to generate code that uses the Sun FPA instructions for floating point.

`-mid-shared-library`: Specifying this option causes GCC to generate code that supports the shared library ID mechanism. This mechanism supports both execute in place and the use of shared libraries without requiring virtual memory management. Using this option implies the `-fPIC` option.

`-mno-align-int`: Specifying this option tells GCC to align `int`, `long`, `long long`, `float`, `double`, and `long double` variables on a 16-bit boundary. This is the default.

`-mno-id-shared-library`: Specifying this option causes GCC to generate code that does not use the shared library ID mechanism. This is the default.

`-mno-sep-data`: Specifying this option causes GCC to assume that the data segment follows the text segment. This is the default.

`-mno-strict-align`: Specifying this option tells GCC to assume that unaligned memory references will be handled by the system.

`-mnobitfield`: Specifying this option tells GCC not to use the bit-field instructions. The `-m68000`, `-mcpu32`, and `-m5200` options imply the `-mnobitfield` option.

`-mpcrel`: Specifying this option tells GCC to use the PC-relative addressing mode of the 68000 directly, instead of using a global offset table. This option implies the standard GCC `-fpic` option, which therefore allows, at most, a 16-bit offset for PC-relative addressing. The `-fpic` option is not presently supported with the `-mpcrel` option.

`-mrtd`: Specifying this option tells GCC to use a function-calling convention where functions that take a fixed number of arguments return with the rtd instruction, which pops their arguments during the return. The rtd instruction is supported by the 68010, 68020, 68030, 68040, 68060, and CPU32 processors, but not by the 68000 or 5200 processors. Using the rtd instruction saves one instruction in the caller since there is no need to pop the arguments there. This calling convention is incompatible with the one normally used on Unix and therefore cannot be used if you need to call libraries that have been compiled with generic Unix compilers. When using this option, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`). If you do not, incorrect code will be generated for calls to those functions. You must also be careful to never call functions with extra arguments, which would result in seriously incorrect code.

■ **Tip** To optimize heavily used functions, you can specify that an individual function is called with the calling sequence specified by the `-mrtd option` with the function attribute `stdcall`. You can also override the `-mrtd` option for specific functions by using the function attribute `cdecl`.

`-msep-data`: Specifying this option tells GCC to generate code that allows the data segment to be located in a different memory area than the text segment. This option implies `-fPIC`, and enables support for execute in place without requiring virtual memory management.

`-mshared-library-id=`*n*: Specifying this option enables you to indicate the ID number (*n*) of an ID-based shared library that is being compiled.

`-mshort`: Specifying this option tells GCC to consider type `int` to be 16 bits wide, like `short int`.

`-msoft-float`: Specifying this option tells GCC that floating-point support should not be assumed and to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC (except for the embedded GCC build targets m68k-*-aout and m68k-*-coff), but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system, or these function calls will be identified as unresolved.

`-mstrict-align`: Specifying this option tells GCC not to assume that unaligned memory references will be handled by the system and to perform the alignment itself.

M68HC1x Options

The M68HC1x is a 16-bit microprocessor that is typically used as a microcontroller in embedded applications.

GCC options available when compiling code for M68HC1x systems are the following:

`-m6811` | `-m68hc11`: Specifying either of these options tells GCC to generate code for a 68HC11 microcontroller. This is the default when the compiler is configured for 68HC11-based systems.

`-m6812` | `-m68hc12`: Specifying either of these options tells GCC to generate code for a 68HC12 microcontroller. This is the default when the compiler is configured for 68HC12-based systems.

`-m68S12` | `-m68hcs12`: Specifying this option tells GCC to generate code for a 68HC12 microcontroller.

`-mauto-incdec`: Specifying this option tells GCC to use 68HC12 pre- and post-autoincrement and autodecrement addressing modes.

`-mlong-calls`: Specifying this option tells GCC to treat all calls as being far away, which causes GCC to use the call instruction to call a function, and the rtc instruction to return from those calls.

`-minmax`: Specifying this option enables GCC to use the 68HC12 min and max instructions.

`-mno-long-calls`: Specifying this option tells GCC to treat all calls as being near.

`-mno-minmax`: Specifying this option prevents GCC from using the 68HC12 min and max instructions.

`-mshort`: Specifying this option tells GCC to consider the `int` datatype to be 16 bits wide, like `short int`.

`-msoft-reg-count=`*count*: Specifying this option enables you to tell GCC the number of pseudosoft registers (*count*) that are used for the code generation. The maximum number is 32. Depending on the program, using more pseudosoft registers may or may not result in improved performance. The default is `4` for 68HC11, and `2` for 68HC12.

M88K Options

The Motorola 88000 family of RISC processors were primarily designed for use in workstations such as the Data General AViiON.

■ **Note** Support for this processor family is not provided in GCC version 4.0. These options are therefore only of interest if you are using a version of GCC that is earlier than 4.x and that you are certain provides support for this processor family.

GCC options available when compiling code for M88K systems are the following:

`-m88000`: Specifying this option tells GCC to generate code that works well on both the M88100 and the M88110 processors.

`-m88100`: Specifying this option tells GCC to generate code that works best for the M88100 processor, but that also runs on the M88110 processor.

`-m88110`: Specifying this option tells GCC to generate code that works best for the M88110 processor and may not run on the M88100 processor.

`-mbig-pic`: This deprecated option has the same effect as the `-fPIC` option on M88000 systems.

`-mcheck-zero-division`: Specifying this option tells GCC to generate code that guarantees that integer division by zero will be detected. This is the default. Some M88100 processors do not correctly detect integer division by zero, though all M88110 processors do. Using this option as a default generates code that will execute correctly on systems using either type of processor. This option is ignored if the `-m88110` option is specified.

`-mhandle-large-shift`: Specifying this option tells GCC to generate code that detects bit shifts of more than 31 bits and emits code to handle them properly.

`-midentify-revision`: Specifying this option tells GCC to include an `ident` directive in the assembler output that identifies the name of the source file, identifies the name and version of GCC, provides a time stamp, and records the GCC options used.

`-mno-check-zero-division`: Specifying this option tells GCC not to generate code that guarantees that integer division by zero will be detected. The MC88100 processor does not always trap on integer division by zero, so GCC generates additional code to explicitly check for zero divisors and trap with exception 503 when this is detected. This option is useful to reduce code size and possibly increase performance when running on systems with an MC88110 processor, which correctly detects all instances of integer division by zero.

`-mno-ocs-debug-info`: Specifying this option tells GCC not to include additional debugging information (about registers used in each stack frame) as specified in the 88open Object Compatibility Standard. This is the default on M88K systems running operating systems other than DG/UX, SVr4, and Delta 88 SVr3.2.

`-mno-ocs-frame-position`: Specifying this option tells GCC to use the offset from the frame pointer register (register `30`) when emitting COFF debugging information for automatic variables and parameters stored on the stack. When this option is in effect, the frame pointer is not eliminated when debugging information is selected by the `-g` switch. This is the default on M88K systems running operating systems other than DG/UX, SVr4, Delta 88 SVr3.2, and BCS.

`-mno-optimize-arg-area`: Specifying this option tells GCC not to reorganize the stack frame to save space, which results in increased memory use by the application. This is the default. The generated code conforms to the 88open Object Compatibility Standard specification.

`-mno-serialize-volatile`: Specifying this option tells GCC not to generate code to guarantee the sequential consistency of volatile memory references. This option is useful because by default GCC generates code to guarantee serial consistency, even on the M88100 processor, where serial consistency is guaranteed. This is done to enable the same code to run on M88110 systems, where the order of memory references does not always match the order of the instructions requesting those references. For example, on an M881100 processor, a load instruction may execute before a preceding store instruction. If you intend to run your code only on the M88100 processor, using the `-mno-serialize-volatile` option will produce smaller, faster code.

`-mno-underscores`: Specifying this option tells GCC to emit symbol names in assembler output without adding an underscore character at the beginning of each name. This is used for integration with linkers that do not follow the standard name mangling conventions. The default is to use an underscore as a prefix on each name.

`-mocs-debug-info`: Specifying this option tells GCC to include additional debugging information (about registers used in each stack frame) as specified in the 88open Object Compatibility Standard. This extra information allows debugging of code that has had the frame pointer eliminated. This is the default for M88K systems running DG/UX, SVr4, and Delta 88 SVr3.2.

`-mocs-frame-position`: Specifying this option tells GCC to use the offset from the canonical frame address when emitting COFF debugging information for automatic variables and parameters stored on the stack. The canonical frame address is the stack pointer (register `31`) on entry to the function. This is the default on M88K systems running DG/UX, SVr4, Delta 88 SVr3.2, and BCS.

`-moptimize-arg-area`: Specifying this option tells GCC to save space by reorganizing the stack frame, reducing memory consumption. This option generates code that does not agree with the 88open Object Compatibility Standard specification.

`-mserialize-volatile`: Specifying this option tells GCC to generate code that guarantees the serial consistency of volatile memory references. This is the default.

`-mshort-data-`*`num`*: Specifying this option tells GCC to generate smaller data references by making them relative to `r0`, which allows loading a value using a single instruction (rather than the usual two). The value specified for *`num`* enables you to control which data references are affected by this option by identifying the maximum displacement of short references that will be handled in this fashion. For example, specifying the `-mshort-data-512` option limits affected data references to those involving displacements of less than 512 bytes. The maximum value for `num` is 64K.

`-msvr3`: Specifying this option tells GCC to turn off compiler extensions related to System V Release 4 (SVR4). This option is the default for all GCC M88K build configurations other than the m88k-motorola-sysv4 and m88k-dg-dgux m88k GCC build targets.

`-msvr4`: Specifying this option tells GCC to turn on compiler extensions related to SVR4. Using this option makes the C preprocessor recognize `#pragma weak` and causes GCC to issue additional declaration directives that are used in SVR4. This option is the default for the m88k-motorola-sysv4 and m88k-dg-dgux m88k GCC build targets.

`-mtrap-large-shift`: Specifying this option tells GCC to generate code that traps on bit shifts of more than 31 bits. This is the default.

`-muse-div-instruction`: Specifying this option tells GCC to use the div instruction for signed integer division on the M88100 processor. By default, the div instruction is not used. On the M88100 processor, the signed integer division instruction traps to the operating system on a negative operand. The operating system transparently completes the operation, but at a large cost in execution time. On the M88110 processor, the div instruction (also known as the divs instruction) processes negative operands without trapping to the operating system. Using this option causes GCC to generate code that will run correctly on either type of processor. This option is ignored if the `-m88110` option is specified.

■ **Note** The result of dividing `int_min()` by -1 is undefined. In particular, the behavior of such a division with and without using the `-muse-div-instruction` option may differ.

`-mversion-03.00`: This option is obsolete and is ignored.

`-mwarn-passed-structs`: Specifying this option tells GCC to display a warning when a function passes a `struct` as an argument or a result. Structure-passing conventions have changed during the evolution of the C language and are often the source of portability problems. By default, GCC does not issue a warning.

MCore Options

Motorola's MCore processor family is a family of general-purpose 32-bit microcontrollers. Also known as M-Core or M*Core processors, the MCore family of processors is often used in embedded devices such as those employed for industrial control and measurement, health care and scientific equipment, and security systems.

GCC options available when compiling code for MCore-based systems are the following:

`-m210`: Specifying this option tells GCC to generate code for the 210 processor.

`-m340`: Specifying this option tells GCC to generate code for the 340 processor.

`-m4byte-functions`: Specifying this option tells GCC to force all functions to be aligned to a 4-byte boundary.

`-mbig-endian`: Specifying this option tells GCC to generate code for a big endian target.

`-mcallgraph-data`: Specifying this option tells GCC to emit callgraph information.

`-mdiv`: Specifying this option tells GCC to use the divide instruction. This is the default.

`-mhardlit`: Specifying this option tells GCC to inline constants in the code stream if it can be done in two instructions or less.

`-mlittle-endian`: Specifying this option tells GCC to generate code for a little endian target.

`-mno-4byte-functions`: Specifying this option tells GCC not to force all functions to be aligned to a 4-byte boundary.

`-mno-callgraph-data`: Specifying this option tells GCC not to emit callgraph information.

`-mno-div`: Specifying this option tells GCC not to use the divide instruction, replacing it with subtraction/remainder operations.

`-mno-hardlit`: Specifying this option tells GCC not to inline constants in the code stream.

`-mno-relax-immediate`: Specifying this option tells GCC not to allow arbitrarily sized immediates in bit operations.

`-mno-slow-bytes`: Specifying this option tells GCC not to prefer word access when reading byte quantities.

`-mno-wide-bitfields`: Specifying this option tells GCC not to treat bit fields as `int`-sized.

`-mrelax-immediate`: Specifying this option tells GCC to allow arbitrarily sized immediates in bit operations.

`-mslow-bytes`: Specifying this option tells GCC to prefer word access when reading byte quantities.

`-mwide-bitfields`: Specifying this option tells GCC to treat bit fields as `int`-sized.

MIPS Options

MIPS, an acronym for Microprocessor without Interlocked Pipeline Stages, is a microprocessor architecture developed by MIPS Computer Systems Inc. based on research and development done at Stanford University. The MIPS R2000 and R3000 processors were 32-bit processors, while later processors such as the R5000, R8000, R10000, R12000, and R16000 are all 64-bit processors. The R6000 processor was a third-party R3000 that quickly vanished, while the R7000 was targeted for embedded use and never saw wide deployment. Processors based on various MIPS cores are widely used in embedded systems.

GCC options available when compiling code for MIPS-based systems are the following:

`-EB`: Specifying this option tells GCC to compile code for the processor in big endian mode. The required libraries are assumed to exist.

`-EL`: Specifying this option tells GCC to generate code for the processor in little endian mode. The required libraries are assumed to exist.

`-G num`: Specifying this option tells GCC to put global and static objects less than or equal to `num` bytes into the small data or bss sections instead of the normal data or bss sections. Using this option enables the assembler to emit one-word memory reference instructions based on the global pointer (`GP` or `$28`), instead of using the normal two words. By default, `num` is `8` when the MIPS assembler is used, and `0` when the GNU assembler is used. The `-G num` switch is also passed to the assembler and linker.

■ **Note** All modules should be compiled with the same `-G num` value. Compiling with different values of `num` may or may not work. If it does not, the linker will display an error message and exit.

`-m4650`: Specifying this option is a convenient shortcut for the `-msingle-float`, `-mmad`, and `-march=r4650` options. (This option is not available in the GCC 4.x compilers.)

`-mabi=32`: Specifying this option tells GCC to generate code for the 32-bit ABI. The default instruction level is `-mips1` when using this option.

`-mabi=n32`: Specifying this option tells GCC to generate code for the new 32-bit ABI.

`-mabi=64`: Specifying this option tells GCC to generate code for the 64-bit application ABI. The default instruction level is `-mips4` when using this option.

`-mabi=o64`: Specifying this option tells GCC to generate code for the old 64-bit ABI.

`-mabi=eabi`: Specifying this option tells GCC to generate code for the EABI. The default instruction level is `-mips4` when using this option. By default, GCC generates 64-bit code for any 64-bit architecture, but you can specify the `-mgp32` option to force GCC to generate 32-bit code.

`-mabi=n32`: Specifying this option tells GCC to generate code for the new 32-bit ABI. The default instruction level is `-mips3` when using this option.

`-mabi=o64`: Specifying this option tells GCC to generate code for the old 64-bit ABI. The default instruction level is `-mips4` when using this option.

`-mabicalls`: Specifying this option tells GCC to generate code containing the pseudo-operations `.abicalls`, `.cpload`, and `.cprestore` that some SVR4 ports use for position-independent code. All code generated using the `-mabicalls` option is position-independent; this behavior can be changed to support absolute addresses for locally bound symbols and make direct calls to locally defined functions by specifying the `-mno-shared` option.

`-march=`*arch*: Specifying this option tells GCC to generate code for *arch*, which is either a MIPS ISA (Instruction Set Architecture) or a specific type of processor. Valid ISA values for *arch* are `mips1`, `mips2`, `mips3`, `mips4`, `mips32`, `mips32r2`, and `mips64`. Valid processor names are `4kc`, `4km`, `4kp`, `5kc`, `5kf`, `20kc`, `24k`, `24kc`, `24kf`, `24kx`, `m4k`, `orion`, `r2000`, `r3000`, `r3900`, `r4000`, `vr4100`, `vr4111`, `vr4120`, `vr4130`, `vr4300`, `r4400`, `r4600`, `r4650`, `vr5000`, `vr5400`, `vr5500`, `r6000`, `rm7000`, `r8000`, `rm9000`, `sr71000`, and `sb1`. The `r2000`, `r3000`, `r4000`, `r5000`, and `r6000` values can be abbreviated as `r2k` (or `r2K`), `r3k`, and so on, and the leading `vr` prefix can simply be abbreviated as `r`. When specifying an ISA, you can also abbreviate the entire `-march=`*arch* option as `-`*arch*. For example, `-mips32` is a valid synonym for `-march=mips32`.

■ **Tip** GCC uses the `arch` value specified using `-march=`*arch* to define two macros: `_MIPS_ARCH`, which provides the name of the target architecture as a string, and `_MIPS_ARCH_`*FOO*, where *FOO* is the name of the uppercase value of `_MIPS_ARCH`. The full numeric value for a specified processor is always used in these macros. For example, if you specify the argument `-march=r4k`, `_MIPS_ARCH` will be set to `r4000`, and the macro `_MIPS_ARCH_R4000` will be defined.

`-mbranch-likely`: Specifying this option tells GCC to use branch likely instructions on all architectures and processors where they are supported. This is the default for architectures other than MIPS32 and MIPS64, where the branch likely instructions are officially deprecated.

`-mcheck-zero-division`: Specifying this option tells GCC to trap on integer division by zero. This is the default.

`-mdivide-breaks`: Specifying this option tells GCC to check for integer division by zero by generating a break instruction. Checking for integer division by zero can be disabled entirely by specifying the `-mno-check-zero-division` option.

`-mdivide-traps`: Specifying this option tells GCC to check for integer division by zero by generating a conditional trap. This is only supported on MIPS 2 or later processors. This option is the default unless GCC was configured using the `--with-divide=breaks` option. Checking for integer division by zero can be disabled entirely by specifying the `-mno-check-zero-division` option.

`-mdouble-float`: Specifying this option tells GCC to assume that the floating-point coprocessor supports double-precision operations. This is the default.

`-mdsp`: Specifying this option enables GCC to generate code that uses the MIPS DSP ASE (digital signal processor application-specific extension).

`-membedded-data`: Specifying this option tells GCC to allocate variables in the read-only data section first, if possible, then in the small data section, if possible, or finally in the data section. This results in slightly slower code than the default, but reduces the amount of RAM required when executing, and thus may be preferred for some embedded systems.

`-membedded-pic`: Specifying this option tells GCC to generate PIC code that is suitable for embedded systems. All calls are made using PC-relative addresses, and all data is addressed using the `$gp` register. No more than 65,536 (64K) bytes of global data may be used. Using this option requires GNU as and GNU ld, which do most of the work. This option currently only works on targets that use the ECOFF binary output format. It does not work with the ELF binary output format.

`-mentry`: Specifying this option tells GCC to use the entry and exit pseudo-ops. This option can only be used with the `-mips16` option.

`-mexplicit-relocs`: Specifying this option tells GCC to use assembler relocation operators when resolving symbolic addresses. This option is the default if GCC was configured to use an assembler that supports relocation operators, such as the GNU assembler.

`-mfix-r4000`: Specifying this option tells GCC to use workarounds for various problems with the R4000 processor.

`-mfix-r4400`: Specifying this option tells GCC to use workarounds for various problems with the R4400 processor.

`-mfix-sb1`: Specifying this option tells GCC to use workarounds for various problems with the SB-1 CPU core.

`-mfix-vr4120`: Specifying this option tells GCC to use workarounds for various problems with the VR4120 processor.

`-mfix-vr4130`: Specifying this option tells GCC to use workarounds for various problems with the VR4130 processor.

`-mflush-func=func`: Specifying this option tells GCC the function to call in order to flush the I and D caches. The function must take the same arguments as the common `_flush_func()` function, which are the address of the memory range for which the cache is being flushed, the size of the memory range, and the number 3 (to flush both caches). The default is usually `_flush_func` or `__cpu_flush`, and is defined by the macro `TARGET_SWITCHES` in the machine description file.

`-mfp-exceptions`: Specifying this option enables GCC to use floating-point exceptions, which affects the scheduling of floating-point instructions on some processors. This option is the default.

`-mfp32`: Specifying this option tells GCC to assume that 32 32-bit floating-point registers are available. This is the default.

`-mfp64`: Specifying this option tells GCC to assume that 32 64-bit floating-point registers are available. This is the default when the `-mips3` option is used.

`-mfused-madd`: Specifying this option tells GCC to generate code that uses the floating-point multiply and accumulate instructions when they are available. These instructions are generated by default if they are available.

`-mgas`: Specifying this option tells GCC to generate code for the GNU assembler. This is the default on the OSF/1 reference platform, which uses the OSF/rose object format. More significantly, this is the default if the GCC configuration option `--with-gnu-as` is used, which is the GCC default. (This option is not available in the GCC 4.x compilers.)

`-mgp32`: Specifying this option tells GCC to assume that 32 32-bit general-purpose registers are available. This is the default.

`-mgp64`: Specifying this option tells GCC to assume that 32 64-bit general-purpose registers are available. This is the default when the `-mips3` option is used.

`-mgpopt`: Specifying this option tells GCC to write all of the data declarations before the instructions in the text section. This allows the MIPS assembler to generate one-word memory references instead of using two words for short global or static data items. This is the default if optimization is selected. (This option is not available in the GCC 4.x compilers.)

`-mhalf-pic`: Specifying this option tells GCC to put pointers to `extern` references into the data section, rather than putting them in the text section. (This option is not available in the GCC 4.x compilers.)

`-mhard-float`: Specifying this option tells GCC to generate code that contains floating-point instructions. This is the default.

`-mint64`: Specifying this option tells GCC to force `int` and `long` types to be 64 bits wide. See the discussion of the `-mlong32` option for an explanation of the default and the width of pointers. (This option is not available in the GCC 4.x compilers.)

`-mips1`: Specifying this option tells GCC to generate instructions that are conformant to level 1 of the MIPS ISA. This is the default. `r3000` is the default `CPU-type` at this ISA level. The default ABI is 32 (`-mabi=32`) when using this option.

`-mips16`: Specifying this option tells GCC to enable the use of 16-bit instructions.

`-mips2`: Specifying this option tells GCC to generate instructions that are conformant to level 2 of the MIPS ISA (branch likely, square root instructions, etc.). `r6000` is the default `CPU-type` at this ISA level. The default ABI is 32 (`-mabi=32`) when using this option.

`-mips3`: Specifying this option tells GCC to generate instructions that are conformant to level 3 of the MIPS ISA (64-bit instructions). `r4000` is the default `CPU-type` at this ISA level. The default ABI is 64 (`-mabi=64`) when using this option.

`-mips3d`: Specifying this option enables GCC to generate code that uses the MIPS 3D ASE (three-dimensional application-specific extensions).

`-mips4`: Specifying this option tells GCC to generate instructions that are conformant to level 4 of the MIPS ISA (conditional move, prefetch, enhanced FPU instructions). `r8000` is the default `CPU-type` at this ISA level. The default ABI is 64 (`-mabi=64`) when using this option.

`-mlong32`: Specifying this option tells GCC to force `long`, `int`, and `pointer` types to be 32 bits wide.

Note If none of the options `-mlong32`, `-mlong64`, or `-mint64` are set, the size of `int`s, `long`s, and `pointer`s depends on the ABI and ISA chosen. For `-mabi=32` and `-mabi=n32`, `int`s and `long`s are 32 bits wide. For `-mabi=64`, `int`s are 32 bits wide, and `long`s are 64 bits wide. For `-mabi=eabi` and either `-mips1` or `-mips2`, `int`s and `long`s are 32 bits wide. For `-mabi=eabi` and higher ISAs, `int`s are 32 bits, and `long`s are 64 bits wide. The width of pointer types is the smaller of the width of `long`s or the width of general-purpose registers (which in turn depends on the ISA).

`-mlong64`: Specifying this option tells GCC to force `long` types to be 64 bits wide. See the discussion of the `-mlong32` option for an explanation of the default and the width of pointers.

`-mlong-calls`: Specifying this option tells GCC to make all function calls using the JALR instruction, which requires loading a function's address into a register before making the call. You must use this option if you call outside of the current 512-megabyte segment to functions that are not called through pointers.

`-mmad`: Specifying this option tells GCC to permit the use of the mad, madu, and mul instructions, as on the r4650 chip.

`-mmemcpy`: Specifying this option tells GCC to make all block moves call the appropriate string function (`memcpy()` or `bcopy()`) instead of possibly generating inline code.

`-mno-fp-exceptions`: Specifying this option prevents GCC from using floating-point exceptions.

`-mno-fused-madd`: Specifying this option prevents GCC from generating code that uses the floating-point multiply and accumulate instructions, even when they are available.

`-mno-mips3d`: Specifying this option prevents GCC from generating code that uses the MIPS 3D ASE. This option is the default.

`-mno-abicalls`: Specifying this option tells GCC not to generate code containing the pseudo-operations `.abicalls`, `.cpload`, and `.cprestore` that some System V.4 ports use for position-independent code.

`-mno-branch-likely`: Specifying this option prevents GCC from using the branch likely instruction.

`-mno-check-zero-division`: Specifying this option tells GCC not to check for attempts to perform integer division by zero.

`-mno-dsp`: Specifying this option prevents GCC from generating code that uses the MIPS DSP ASE. This option is the default.

`-mno-explicit-relocs`: Specifying this option tells GCC to use macros rather than assembler relocation operators when resolving symbolic addresses.

`-mno-flush-func`: Specifying this option tells GCC not to call any function to flush the I and D caches.

`-mno-fused-madd`: Specifying this option tells GCC not to generate code that uses the floating-point multiply and accumulate instructions, even if they are available. These instructions may be undesirable if the extra precision causes problems or on certain chips in the modes where denormals are rounded to zero and where denormals generated by multiply and accumulate instructions cause exceptions anyway.

`-mno-gpopt`: Specifying this option tells GCC not to write all of the data declarations before the instructions in the text section, preventing some optimizations but resulting in more readable assembly code.

`-mno-long-calls`: Specifying this option tells GCC not to use the JALR instruction when making function calls.

`-mno-mad`: Specifying this option tells GCC not to permit the use of the mad, madu, and mul instructions.

`-mno-memcpy`: Specifying this option tells GCC to possibly generate inline code for all block moves rather than calling the appropriate string function (`memcpy()` or `bcopy()`).

`-mno-mips-tfile`: Specifying this option tells GCC not to post-process the object file with the `mips-tfile` program, which adds debugging support after the MIPS assembler has generated it. If the `mips-tfile` program is not run, no local variables will be available to the debugger. In addition, `stage2` and `stage3` objects will have the temporary filenames passed to the assembler embedded in the object file, which means the objects will not compare the same. The `-mno-mips-tfile` switch should only be used when there are bugs in the `mips-tfile` program that prevents compilation. This option is the default in modern versions of GCC, which all use the GNU assembler.

`-mno-mips16`: Specifying this option tells GCC not to use 16-bit instructions.

`-mno-embedded-data`: Specifying this option tells GCC to allocate variables in the data section, as usual. This may result in faster code but increases the amount of RAM required to run an application.

`-mno-embedded-pic`: Specifying this option tells GCC not to make all function calls using PC-relative addresses, and not to address all data using the `$gp` register. This is the default on systems that use the ELF binary output format, or use an assembler or linker/loader other than the GNU tools. (This option is not available in the GCC 4.x compilers.)

`-mno-half-pic`: Specifying this option tells GCC to put pointers to `extern` references in the text section. This option is the default. (This option is not available in the GCC 4.x compilers.)

`-mno-rnames`: Specifying this option tells GCC to generate code that uses the hardware names for the registers (i.e., `$4`). This is the default. (This option is not available in the GCC 4.x compilers.)

`-mno-shared`: Specifying this option prevents GCC from generating position-independent code. Specifying this option also supports absolute addresses for locally bound symbols and direct calls to locally defined functions. (This option is not available in the GCC 4.x compilers.)

`-mno-split-addresses`: Specifying this option tells GCC not to generate code that loads the high and low parts of address constants separately, which prevents some optimizations but may be necessary if using a non-GNU assembler or linker/loader.

`-mno-stats`: Specifying this option tells GCC not to emit statistical information when processing noninline functions. This is the default. (This option is not available in the GCC 4.x compilers.)

`-mno-sym32`: Specifying this option tells GCC not to assume that all symbols have 32-bit values.

`-mno-uninit-const-in-rodata`: Specifying this option tells GCC to store uninitialized `const` variables in the data section, as usual. This is the default.

`-mno-vr4130-align`: Specifying this option prevents GCC from aligning pairs of instructions that might be able to execute in parallel. This produces smaller code, but doesn't take advantage of the VR4130's two-way superscalar pipeline.

`-mno-xgot`: Specifying this option tells GCC to impose a 64K limitation on the size of the GOT, which therefore only requires a single instruction to fetch the value of a global symbol. This option is the default.

`-mshared`: Specifying this option causes GCC to generate position-independent code that can be linked into shared libraries. All code generated using the `-mabicalls` option is position-independent—this behavior can be changed to support absolute addresses for locally bound symbols by specifying the `-mno-shared` option.

`-msingle-float`: Specifying this option tells GCC to assume that the floating-point coprocessor only supports single precision operations, as on the r4650 chip.

`-msoft-float`: Specifying this option tells GCC that floating-point support should not be assumed, and to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC, but are normally found on the target system, and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system or these function calls will be identified as unresolved.

`-msplit-addresses`: Specifying this option tells GCC to generate code that loads the high and low parts of address constants separately. This allows GCC to optimize away redundant loads of the high order bits of addresses. This optimization requires GNU as and GNU ld. This optimization is enabled by default for some embedded targets where GNU as and GNU ld are standard.

`-mstats`: Specifying this option tells GCC to emit one line to the standard error output for each noninline function processed. This line provides statistics about the program (number of registers saved, stack size, and so on).

`-msym32`: Specifying this option tells GCC to assume that all symbols have 32-bit values, regardless of the ABI that you are using. This option can provide faster access to symbolic addresses when used in conjunction with the `-mabi=64` and `-no-abicalls` options.

`-mtune=`*`arch`*: Specifying this option tells GCC to use the defaults for the specified machine type *`arch`* when scheduling instructions. See the `-march=`*`arch`* option for a list of possible values for *`arch`*. Supplying different values for *`arch`* to the `-march=`*`arch`* and `-mtune=`*`arch`* options enables GCC to generate code that runs on an entire family of processors, but is tuned for optimal performance on a specific CPU. If this option is not specified, the code produced by GCC is tuned for the *`arch`* specified using any `-march=`*`arch`* option.

■ **Tip** GCC uses the `arch` value specified using `-mtune=`*`arch`* to define two macros: `_MIPS_TUNE`, which provides the name of the target architecture as a string, and `_MIPS_TUNE_`*`FOO`*, where *`FOO`* is the name of the uppercase value of `_MIPS_TUNE`. The full numeric value for a specified processor is always used in these macros. For example, if you specify the argument `-march=r4k`, `_MIPS_TUNE` will be set to `r4000`, and the macro `_MIPS_TUNE_R4000` will be defined.

`-muninit-const-in-rodata`: Specifying this option tells GCC to store uninitialized `const` variables in the read-only data section. This option must be used with the `-membedded-data` option.

`-mvr4130-align`: Specifying this option tells GCC to align pairs of instructions that it believes can execute in parallel in order to take advantage of the VR4130's two-way superscalar pipeline. This option is used when optimizing code for the VR4130, produces faster code at the expense of code size, and is active by default at optimization levels `-O3` or greater.

`-mxgot`: Specifying this option tells GCC to remove limitations on the size of the GOT, which therefore requires multiple instructions to fetch the value of a global symbol. This is necessary when the size of the GOT table exceeds 64K.

`-nocpp`: Specifying this option instructs the MIPS assembler not to run its preprocessor over user assembler files (files with an `.s` suffix) when assembling them.

`-no-crt0`: Specifying this option tells GCC not to include the default crt0 C runtime initialization library.

MMIX Options

MIX was a virtual computer system designed for use as an example in Donald Knuth's legendary *The Art of Computer Programming* books. MMIX is a 64-bit RISC-oriented follow-up to MIX.

GCC options available when compiling code for MMIX-based systems are the following:

`-mabi=gnu`: Specifying this option tells GCC to generate code that is conformant to the GNU ABI and therefore passes function parameters and return values in global registers `$231` and up.

`-mabi=mmixware`: Specifying this option tells GCC to generate code that passes function parameters and return values that (in the called function) are seen as registers `$0` and up.

`-mbase-addresses`: Specifying this option tells GCC to generate code that uses `_base addresses_`. Using a base address automatically generates a request (handled by the assembler and the linker) for a constant to be set up in a global register. The register is used for one or more base address requests within the range 0 to 255 from the value held in the register. This generally leads to short and fast code, but the number of different data items that can be addressed is limited. This means that a program that uses a lot of static data may require `-mno-base-addresses`.

`-mbranch-predict`: Specifying this option tells GCC to use the probable-branch instructions when static branch prediction indicates a probable branch.

`-melf`: Specifying this option tells GCC to generate code in ELF format, rather than in the default mmo format used by the MMIX simulator.

`-mepsilon`: Specifying this option tells GCC to generate floating-point comparison instructions that compare with respect to the `rE` epsilon register.

`-mknuthdiv`: Specifying this option tells GCC to make the result of a division yielding a remainder have the same sign as the divisor.

`-mlibfuncs`: Specifying this option tells GCC that all intrinsic library functions are being compiled, passing all values in registers, no matter the size.

`-mno-base-addresses`: Specifying this option tells GCC not to generate code that uses `_base addresses_`. Using a base address generally leads to short and fast code but limits the number of different data items that can be addressed. Programs that use significant amounts of static data may require `-mno-base-addresses`.

`-mno-branch-predict`: Specifying this option tells GCC not to use the probable-branch instructions.

`-mno-epsilon`: Specifying this option tells GCC not to generate floating-point comparison instructions that use the `rE` epsilon register.

`-mno-knuthdiv`: Specifying this option tells GCC to make the result of a division yielding a remainder have the same sign as the dividend. This is the default.

`-mno-libfuncs`: Specifying this option tells GCC that all intrinsic library functions are not being compiled and that values should therefore not be passed in registers.

`-mno-single-exit`: Specifying this option enables GCC to support multiple exit points in a function.

`-mno-toplevel-symbols`: Specifying this option tells GCC not to insert a colon (:) at the beginning of all global symbols. This disallows the use of the `PREFIX` assembly directive with the resulting assembly code.

`-mno-zero-extend`: Specifying this option tells GCC to use sign-extending load instructions when reading data from memory in sizes shorter than 64 bits.

`-msingle-exit`: Specifying this option tells GCC to force generated code to have a single exit point in each function.

`-mtoplevel-symbols`: Specifying this option tells GCC to insert a colon (:) at the beginning of all global symbols, so that the assembly code can be used with the `PREFIX` assembly directive.

`-mzero-extend`: Specifying this option tells GCC to use zero-extending load instructions when reading data from memory in sizes shorter than 64 bits.

MN10200 Options

The MN10200 series of 16-bit single-chip microcontrollers from Panasonic are low-power processors with a 16MB address space and fast instruction execution time complemented by a three-stage pipeline.

■ **Note** Support for this processor family is not provided in GCC version 4.0. This option is therefore only of interest if you are using a version of GCC that is earlier than 4.x and which you are certain provides support for this processor family.

The GCC option available when compiling code for MN10200-based systems is the following:

`-mrelax`: Specifying this option tells GCC to tell the linker that it should perform a relaxation optimization pass to shorten branches, calls, and absolute memory addresses. This option is only useful when specified on the command line for the final link step. Using this option makes symbolic debugging impossible.

MN10300 Options

The MN10300 series of 32-bit single-chip microcontrollers are the descendents of the MN10200 processors, and add built-in support for multimedia applications to the core capabilities of the MN10200.

GCC options available when compiling code for MN10300-based systems are the following:

`-mam33`: Specifying this option tells GCC to generate code that uses features specific to the AM33 processor.

`-mmult-bug`: Specifying this option tells GCC to generate code that avoids bugs in the multiply instructions for the MN10300 processors. This is the default.

`-mno-am33`: Specifying this option tells GCC not to generate code that uses features specific to the AM33 processor. This is the default.

`-mno-crt0`: Specifying this option tells GCC not to link in the C runtime initialization object file.

`-mno-mult-bug`: Specifying this option tells GCC not to generate code that avoids bugs in the multiply instructions for the MN10300 processors. This may result in smaller, faster code if your application does not trigger these bugs.

`-mno-return-pointer-on-d0`: Specifying this option tells GCC to only return a pointer in `a0` when generating code for functions that return pointers.

`-mrelax`: Specifying this option tells GCC to tell the linker that it should perform a relaxation optimization pass to shorten branches, calls, and absolute memory addresses. This option is only useful when specified on the command line for the final link step. Using this option makes symbolic debugging impossible.

`-mreturn-pointer-on-d0`: Specifying this option tells GCC to return pointers in both `a0` and `d0` when generating code for functions that return pointers. This option is the default.

MT Options

The Morpho Technologies MS1 and MS2 processors are reconfigurable digital signal processors that are popular in wireless and handheld devices.

GCC options available when compiling code for MT systems are the following:

`-march=`*`cpu-type`*: Specifying this option tells GCC to generate code that will run on *`cpu-type`*, which is a specific type of MT processor. Possible values for *`cpu-type`* are `ms1-64-001`, `ms1-16-002`, `ms1-16-003`, and `ms2`. If no `-march=`*`cpu-type`* option is specified, the default value is `-march=ms1-16-002`.

`-mbacc`: Specifying this option tells GCC to use byte loads and stores when generating code.

`-mno-bacc`: Specifying this option prevents GCC from using byte loads and stores when generating code.

`-mno-crt0`: Specifying this option prevents GCC from linking the object code that it generates with the C runtime initialization object file crti.o. Other runtime initialization and termination files, such as startup.o and exit.o, will still be used.

`-msim`: Specifying this option tells GCC to use a special runtime with the MT simulator.

NS32K Options

The National Semiconductor NS32000 (a.k.a. NS32K) family of processors was used in a variety of older computer systems and has also been used in embedded and signal processing applications.

■ **Note** Support for this processor family is not provided in GCC version 4.0. These options are therefore only of interest if you are using a version of GCC that is earlier than 4.x and that you are certain provides support for this processor family.

GCC options available when compiling code for NS32000-based systems are the following:

`-m32032`: Specifying this option tells GCC to generate code for a 32032 processor. This is the default when the compiler is configured for 32032- and 32016-based systems.

`-m32081`: Specifying this option tells GCC to generate code containing 32081 instructions for floating point. This is the default for all systems.

`-m32332`: Specifying this option tells GCC to generate code for a 32332 processor. This is the default when the compiler is configured for 32332-based systems.

`-m32381`: Specifying this option tells GCC to generate code containing 32381 instructions for floating point. Specifying this option also implies the `-m32081` option. The 32381 processor is only compatible with the 32332 and 32532 CPUs. This is the default for GCC's `pc532-netbsd` build configuration.

`-m32532`: Specifying this option tells GCC to generate code for a 32532 processor. This is the default when the compiler is configured for 32532-based systems.

`-mbitfield`: Specifying this option tells GCC to use bit-field instructions. This is the default for all platforms except the pc532.

`-mhimem`: Specifying this option tells GCC to generate code that can be loaded above 512MB. Many NS32000 series addressing modes use displacements of up to 512MB. If an address is above 512MB, then displacements from zero cannot be used. This option is often useful for operating systems or ROM code.

`-mmulti-add`: Specifying this option tells GCC to attempt to generate the multiply-add floating-point instructions polyF and dotF. This option is only available if the `-m32381` option is also specified. Using these instructions requires changes to register allocation that generally have a negative impact on performance. This option should only be enabled when compiling code that makes heavy use of multiply-add instructions.

`-mnobitfield`: Specifying this option tells GCC not to use the bit-field instructions. On some machines, such as the pc532, it is faster to use shifting and masking operations. This is the default for the pc532.

`-mnohimem`: Specifying this option tells GCC to assume that code will be loaded in the first 512MB of virtual address space. This is the default for all platforms.

`-mnomulti-add`: Specifying this option tells GCC not to generate code containing the multiply-add floating-point instructions polyF and dotF. This is the default on all platforms.

`-mnoregparam`: Specifying this option tells GCC not to pass any arguments in registers. This is the default for all targets.

`-mnosb`: Specifying this option tells GCC not to use the `sb` register as an index register. This is usually because it is not present or is not guaranteed to have been initialized. This option is the default for all targets except the pc532-netbsd. This option is also implied whenever the `-mhimem` or `-fpic` options are specified.

`-mregparam`: Specifying this option tells GCC to use a different function-calling convention where the first two arguments are passed in registers. This calling convention is incompatible with the one normally used on Unix, and therefore should not be used if your application calls libraries that have been compiled with the Unix compiler.

`-mrtd`: Specifying this option tells GCC to use a function-calling convention where functions that take a fixed number of arguments pop their arguments during the return. This calling convention is incompatible with the one normally used on Unix, and therefore cannot be used if you need to call libraries that have been compiled with generic Unix compilers. When using this option, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`); if you do not, incorrect code will be generated for calls to those functions. You must also be careful to never call functions with extra arguments, which would result in seriously incorrect code. This option takes its name from the 680x0 rtd instruction.

`-msoft-float`: Specifying this option tells GCC that floating-point support should not be assumed and to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC, but are normally found on the target system and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system or these function calls will be identified as unresolved.

`-msb`: Specifying this option tells GCC that the `sb` can be used as an index register that is always loaded with zero. This is the default for the `pc532-netbsd` target.

PDP-11 Options

The options in this section are specific to using GCC to compile applications for the former Digital Equipment Corporation's PDP-11 minicomputers. Few of these systems are still in use, aside from some that are still used in older process control applications. However, these options can still be useful if you happen to have PDP-11 in the basement "just in case." I know I do.

GCC options available when compiling code for PDP-11 systems are the following:

`-m10`: Specifying this option tells GCC to generate code for a PDP-11/10.

`-m40`: Specifying this option tells GCC to generate code for a PDP-11/40.

`-m45`: Specifying this option tells GCC to generate code for a PDP-11/45. This is the default.

`-mabshi`: Specifying this option tells GCC to use the abshi2 pattern. This is the default.

`-mac0`: Specifying this option tells GCC to return floating-point results in `ac0` (`fr0` in Unix assembler syntax).

`-mbcopy`: Specifying this option tells GCC not to use inline movstrhi patterns for copying memory.

`-mbcopy-builtin`: Specifying this option tells GCC to use inline movstrhi patterns for copying memory. This is the default.

`-mbranch-cheap`: Specifying this option tells GCC not to assume that branches are expensive. This is the default.

`-mbranch-expensive`: Specifying this option tells GCC to assume that branches are expensive. This option is designed for experimenting with code generation and is not intended for production use.

`-mdec-asm`: Specifying this option tells GCC to use DEC (Compaq? HP?) assembler syntax. This is the default when GCC is configured for any PDP-11 build target other than `pdp11-*-bsd`.

`-mfloat32` | `-mno-float64`: Specifying either of these options tells GCC to use 32-bit floats.

`-mfloat64` | `-mno-float32`: Specifying either of these options tells GCC to use 64-bit floats. This is the default.

`-mfpu`: Specifying this option tells GCC to use hardware FPP floating point. This is the default. (FIS floating point on the PDP-11/40 is not supported.)

`-mint16` | `-mno-int32`: Specifying either of these options tells GCC to use 16-bit `int`s. This is the default.

`-mint32` | `-mno-int16`: Specifying either of these options tells GCC to use 32-bit `int`s.

`-mno-abshi`: Specifying this option tells GCC not to use the abshi2 pattern.

`-mno-ac0`: Specifying this option tells GCC to return floating-point results in memory. This is the default.

`-mno-split`: Specifying this option tells GCC to generate code for a system without split instruction and data spaces. This is the default.

`-msoft-float`: Specifying this option tells GCC that floating-point support should not be assumed, and to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC, but are normally found on the target system, and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system or these function calls will be identified as unresolved.

`-msplit`: Specifying this option tells GCC to generate code for a system with split Instruction and Data spaces.

`-munix-asm`: Specifying this option tells GCC to use Unix assembler syntax. This is the default when GCC is configured for the `pdp11-*-bsd` build target.

PowerPC (PPC) Options

The PowerPC is a RISC microprocessor architecture that was originally created by the 1991 Apple-IBM-Motorola alliance, known as AIM. The PowerPC was the CPU portion of the overall AIM platform, and is the only surviving aspect of the platform. Performance Optimization With Enhanced RISC (POWER) had been introduced (and was doing well) as a multichip processor on the IBM RS/6000 workstation, but IBM was interested in a single-chip version as well as entering other markets. At the same time, Apple was looking for a new processor to replace the aging MC680x0 processors in its Macintosh computers.

GCC provides options that enable you to specify which instructions are available on the processor you are using. GCC supports two related instruction set architectures for the PowerPC and RS/6000. The POWER instruction set consists of those instructions that are supported by the RIOS chip set used in the original RS/6000 systems. The PowerPC instruction set is the architecture of the Motorola MPC5xx, MPC6xx, MPC8xx microprocessors, and the IBM 4xx microprocessors. Neither architecture is a subset of the other, but both support a large common subset of instructions. An `mq` register is included in processors supporting the POWER architecture.

■ **Tip** In general, it is easier to use the `-mcpu=CPU-type` option, which implies the appropriate instruction set, rather than trying to remember the appropriate `-mpower*` option.

IBM's RS64 processor family is a modified PowerPC architecture. These processors are used in the AS/400 computer family and in some RS/6000 systems. The latest generation of PowerPC processors (the G5) was used in Apple Macintosh computer systems prior to Apple's switch to Intel processors. PowerPC chips based on older cores are tremendously popular in embedded hardware.

GCC options available when compiling code for PowerPC systems are the following:

`-G num`: Specifying this option on embedded PowerPC systems tells GCC to put global and static objects less than or equal to `num` bytes into the small data or bss sections instead of the normal data or bss sections. By default, `num` is `8`. The `-G num` switch is also passed to the assembler and linker.

■ **Note** All modules should be compiled with the same `-G num` value. Compiling with different values of `num` may or may not work. If it does not, the linker will display an error message and exit.

`-m32`: Specifying this option tells GCC to generate code for a 32-bit environment. The 32-bit environment sets `int`, `long`, and `pointer` to 32 bits and generates code that will run on any PowerPC processor.

`-m64`: Specifying this option tells GCC to generate code for a 64-bit environment. The 64-bit environment sets `int` to 32 bits and `long` and `pointer` to 64 bits and generates code that will only run on PowerPC-64 processors.

`-mabi=`*`abi-type`*: Specifying this option tells GCC to extend the current ABI with a specific set of extensions or to remove a specified set of extensions. Possible values for *`abi-type`* are `altivec` (adds AltiVec vector-processing ABI extensions), `ibmlongdouble` (adds IBM extended precision `long double` extensions, a PowerPC 32-bit SYSV ABI option), `ieeelongdouble` (adds IEEE extended precision `long double` extensions, a PowerPC 32-bit Linux ABI option), `no-altivec` (disables AltiVec ABI extensions for the current ABI), `no-spe` (disables IBM's Synergistic Processing Elements [SPE] SIMD instructions for the current ABI), and `spe` (adds SPE SIMD instructions). Using this option does not change the current ABI except to add or remove the specified extensions.

`-mads`: Specifying this option on embedded PowerPC systems tells GCC to assume that the startup module is called crt0.o and the standard C libraries are libads.a and libc.a.

`-maix-struct-return`: Specifying this option tells GCC to return all structures in memory (as specified by the AIX ABI).

`-maix32`: Specifying this option tells GCC to use the 32-bit ABI, disabling the 64-bit AIX ABI and calling conventions. This is the default.

`-maix64`: Specifying this option tells GCC to enable the 64-bit AIX ABI and calling convention: 64-bit pointers, 64-bit `long` type, and the infrastructure needed to support them. Specifying `-maix64` implies the `-mpowerpc64` and `-mpowerpc` options.

`-malign-natural`: Specifying this option tells GCC to override the alignment of larger types on natural size-based boundaries, as defined by the ABI. This option can be used on AIX, 32-bit Darwin, and 64-bit PowerPC Linux systems. This is the default on 64-bit Darwin systems.

`-malign-power`: Specifying this option tells GCC to follow the alignment rules for larger types as specified in the ABI. This option is not supported on 64-bit Darwin systems.

`-maltivec`: Specifying this option tells GCC to enable the use of built-in functions that provide access to the AltiVec instruction set. You may also need to set `-mabi=altivec` to adjust the current ABI with AltiVec ABI enhancements.

`-mbig | -mbig-endian`: Specifying either of these options on SVR4 or embedded PowerPC systems tells GCC to compile code for the processor in big endian mode.

`-mbit-align`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to force structures and unions that contain bit fields to be aligned to the base type of the bit field. This option is the default. For example, by default a structure containing nothing but eight unsigned bit fields of length 1 would be aligned to a 4-byte boundary and have a size of 4 bytes.

`-mbss-plt`: Specifying this option enables GCC to use a bss PLT section that is filled in by the linker, and requires PLT and GOT sections that are both writable and executable. This is a PowerPC 32-bit SYSV ABI option.

`-mcall-aix`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to compile code using calling conventions that are similar to those used on AIX. This is the default if you configure GCC using the `powerpc-*-eabiaix` GCC build target. (This option is not available in the GCC 4.x compilers.)

`-mcall-gnu`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to compile code using calling conventions similar to those used by the HURD-based GNU system.

`-mcall-linux`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to compile code using calling conventions similar to those used by the Linux-based GNU system.

`-mcall-netbsd`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to compile code using calling conventions similar to those used by the NetBSD operating system.

`-mcall-solaris`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to compile code using calling conventions that are similar to those used by the Solaris operating system.

`-mcall-sysv`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to compile code using calling conventions that adhere to the March 1995 draft of the PowerPC processor supplement to the System V application binary interface. This is the default unless you configured GCC using the `powerpc-*-eabiaix` GCC build target.

`-mcall-sysv-eabi`: Specifying this option is the same as using both the `-mcall-sysv` and `-meabi` options.

`-mcall-sysv-noeabi`: Specifying this option is the same as using both the `-mcall-sysv` and `-mno-eabi` options.

`-mcpu=`*`cpu-type`*: Specifying this option tells GCC to set the architecture type, register usage, choice of mnemonics, and instruction scheduling parameters to values associated with the machine type *`CPU-type`*. Possible values for *`cpu-type`* are `401`, `403`, `505`, `405fp`, `440`, `440fp`, `505`, `601`, `602`, `603`, `603e`, `604`, `604e`, `620`, `630`, `740`, `7400`, `7450`, `750`, `801`, `821`, `823`, `860`, `970`, `common`, `power`, `power2`, `power3`, `power4`, `power5`, `power5+`, `powerpc`, `powerpc64`, `rios`, `rios1`, `rsc`, `rios2`, `rsc`, and `rs64a`. The `-mcpu=power`, `-mcpu=power2`, `-mcpu=powerpc`, and `-mcpu=powerpc64` options specify generic POWER, POWER2, pure 32-bit PowerPC, and 64-bit PowerPC architecture machine types, with an appropriate, generic processor model assumed for scheduling purposes. The other options specify a specific processor. Code generated under those options will run best on that processor and may not run at all on others.

■ **Tip** Specifying the `-mcpu=common` option selects a completely generic processor. Code generated under this option will run on any POWER or PowerPC processor. GCC will use only the instructions in the common subset of both architectures, and will not use the MQ register. GCC assumes a generic processor model for scheduling purposes.

The `-mcpu` options automatically enable or disable the `-maltivec`, `-mfprnd`, `-mhard-float`, `-mmfcrf`, `-mmultiple`, `-mnew-mnemonics`, `-mpopcntb`, `-mpower`, `-mpower2`, `-mpowerpc64`, `-mpowerpc-gpopt`, `-mpowerpc-gfxopt`, `-mstring`, `-mmulhw`, and `-mdlmzb` options to generate the best code possible for a specific processor. You can disable options explicitly if you are sure that you do not want to use them with your processor.

■ **Tip** On AIX systems, the `-maltivec` and `-mpowerpc64` options are not automatically enabled because they are not completely supported on the AIX platform. You can enable them manually if you are sure that they will work in your execution environment.

`-mdlmzb`: Specifying this option tells GCC to generate code that uses the string search dlmzb instructions, as needed. This option is active by default when compiling for the PowerPC 405 and 440 processors. (This option is not available in the GCC 4.x compilers.)

`-mdynamic-no-pic`: Specifying this option tells GCC to generate code that itself is not relocatable but in which all external references are relocatable. The resulting code is suitable for use in applications but not in shared libraries. This option is used on Darwin and Mac OS X systems.

`-meabi`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to adhere to the EABI, which is a set of modifications to the System V.4 specifications. This means that the stack is aligned to an 8-byte boundary, a function `__eabi` is called to from `main()` to set up the EABI environment, and the `-msdata` option can use both `r2` and `r13` to point to two separate small data areas. This option is the default if you configure GCC using one of the `powerpc*-*-eabi*` build targets.

`-memb`: Specifying this option on embedded PowerPC systems tells GCC to set the `PPC_EMB` bit in the ELF flags header to indicate that EABI-extended relocations are used.

`-mfloat-gprs | -mfloat-gprs=yes/single/double/no`: Specifying either of these options enables GCC to generate floating-point operations on the general-purpose registers for architectures that support them. (This option is currently only supported on the MPC854x processors.) You can optionally specify an argument to this option to refine the type of floating-point operations that can be generated: `yes` and `single` enable the use of single-precision floating-point operations; `double` enables the use of both single and double-precision floating-point operations, and `no` disables floating-point operations on general-purpose registers.

`-mfprnd`: Specifying this option enables GCC to generate the floating-point round to integer instructions that are implemented on the POWER5+ and other PowerPC v2.03-compliant processors.

`-mfull-toc`: Specifying this option modifies the generation of the table of contents (TOC) generated for every GCC executable. If the `-mfull-toc` option is specified, GCC allocates at least one TOC entry for each unique nonautomatic variable reference in a program and will also place floating-point constants in the TOC. A maximum of 16,384 entries are available in the TOC. The `-mfull-toc` option is the default.

`-mfused-madd`: Specifying this option tells GCC to generate code that uses the floating-point multiply and accumulate instructions. These instructions are generated by default if hardware floating point is used.

`-mhard-float`: Specifying this option tells GCC to generate code that uses the floating-point register set.

`-minsert-sched-nops=scheme`: Specifying this option identifies the *scheme* that GCC will use for inserting NOOPs during its second scheduling pass. Possible values for *scheme* are `NO` (doesn't insert NOOPs), *NUMBER* (inserts *NUMBER* NOOPs to force costly dependent instructions into separate groups), *PAD* (pads any dispatch groups that has vacant slots with NOOPs), and *REGROUP-EXACT* (inserts NOOPs to force costly dependent instructions into separate groups, based on the grouping for the target processor).

`-misel`: Specifying this option enables GCC to generate `isel` instructions. This option replaces the `-misel=YES` syntax.

`-mlittle | -mlittle-endian`: Specifying either of these options on SVR4 or embedded PowerPC systems tells GCC to compile code for the processor in little endian mode.

`--mlongcall`: Specifying this option tells GCC to translate direct calls to indirect calls unless it can determine that the target of a direct call is in the 32MB range allowed by the call instruction. This can generate slower code on systems whose linker can automatically generate and insert glue code for out-of-range calls, such as the AIX, Darwin, and PowerPC-64 linkers, though the Darwin discards this code if it is unnecessary.

`-mmfcrf`: Specifying this option enables GCC to generate the move from condition register field instructions that are implemented on POWER4 and other PowerPC v2.01-compliant processors.

`-mminimal-toc`: Specifying this option modifies the generation of the TOC that is generated for every PPC executable. The TOC provides a convenient way of looking up the address/entry point of specific functions. This option is a last resort if you see a linker error indicating that you have overflowed the TOC during final linking and have already tried using the `-no-fp-in-toc` and `-mno-sum-in-toc` options. The `-mminimal-toc` option causes GCC to make only one TOC entry for every file. When you specify this option, GCC produces code that is slower and larger but uses extremely little TOC space. You may wish to use this option only on files that contain code that is infrequently executed.

`-mmulhw`: Specifying this option tells GCC to generate code that uses the half-word multiply and multiply-accumulate instructions, as needed. This option is enabled by default when compiling for the PowerPC 405 and 440 processors.

`-mmultiple`: Specifying this option tells GCC to generate code that uses the load multiple word instructions and the store multiple word instructions. These instructions are generated by default on POWER systems, and not generated on PowerPC systems. Do not use `-mmultiple` on little endian PowerPC systems except for the PPC740 and PPC750, since those instructions do not usually work when the processor is in little endian mode. The PPC740 and the PPC750 permit the use of these instructions in little endian mode.

`-mmvme`: Specifying this option on embedded PowerPC systems tells GCC to assume that the startup module is called crt0.o and the standard C libraries are libmvme.a and libc.a.

`-mnew-mnemonics`: Specifying this option tells GCC to use the assembler mnemonics defined for the PowerPC architecture. Instructions defined in only one architecture have only one mnemonic. GCC uses that mnemonic irrespective of which of these options is specified. GCC defaults to the mnemonics appropriate for the architecture that is in use. Unless you are cross-compiling, you should generally accept the default.

`-mno-altivec`: Specifying this option tells GCC to disable the use of built-in functions that provide access to the AltiVec instruction set.

`-mno-bit-align`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC not to force structures and unions that contain bit fields to be aligned to the base type of the bit field. For example, by default, a structure containing nothing but eight unsigned bit fields of length 1 would be aligned to a 4-byte boundary and have a size of 4 bytes. When using the `-mno-bit-align` option, the structure would be aligned to a 1-byte boundary and be 1 byte in size.

`-mno-dlmzb`: Specifying this option prevents GCC from generating code that uses the string search dlmzb instructions that are otherwise used by default when compiling for the PowerPC 405 and 440 processors.

`-mno-eabi`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC not to adhere to the EABI, which is a set of modifications to the System V.4 specifications. This means that the stack is aligned to a 16-byte boundary, an initialization function is not called from `main`, and that the `-msdata` option will only use r13 to point to a single small data area. This is the default for all GCC build configurations other than the `powerpc*-*-eabi*` build targets.

`-mno-fp-in-toc`: Specifying this option tells GCC to generate the same TOC as specified by the `-mfull-toc` option, but not to store floating-point constants in the TOC. This option and the `-mno-sum-in-toc` option are typically used if you see a linker error indicating that you have overflowed the TOC during final linking.

`-mno-fprnd`: Specifying this option prevents GCC from generating the floating-point round to integer instructions that are implemented on the POWER5+ and other PowerPC V2.03-compliant processors.

`-mno-fused-madd`: Specifying this option tells GCC to generate code that does not use the floating-point multiply and accumulate instructions. These instructions are generated by default if hardware floating point is used.

`-mno-isel`: Specifying this option prevents GCC from generating `isel` instructions. This option replaces the `-misel=NO` syntax.

`-mno-longcalls`: Specifying this option tells GCC not to translate direct calls to indirect calls.

`-mno-mfcrf`: Specifying this option prevents GCC from generating the move from condition register field instructions that are implemented on POWER4 and other PowerPC v2.01-compliant processors.

`-mno-mulhw`: Specifying this option tells GCC to generate code that does not use the half-word multiply and multiply-accumulate instructions that are otherwise generated by default when compiling for the PowerPC 405 and 440 processors.

`-mno-multiple`: Specifying this option tells GCC to generate code that does not use the load multiple word instructions or the store multiple word instructions. This option should not be used on little endian systems, with the exception of the 740 and 750 systems. These instructions are generated by default on POWER systems, and are not generated on PowerPC systems.

`-mno-popcntb`: Specifying this option enables GCC to generate the popcount and double-precision floating-point reciprocal estimate instructions that are implemented on the POWER5 and other PowerPC v20.20-compliant processors.

`-mno-power`: Specifying this option prevents GCC from generating code that uses any of the instructions that are specific to the POWER architecture.

■ **Note** If you specify both the `-mno-power` and `-mno-powerpc` options, GCC will use only the instructions in the common subset of both architectures plus some special AIX common-mode calls, and will not use the MQ register.

`-mno-power2`: Specifying this option prevents GCC from generating code that uses any of the instructions that are specific to the POWER2 architecture.

`-mno-powerpc`: Specifying this option prevents GCC from generating code that uses any of the instructions that are specific to the PowerPC architecture.

■ **Note** If you specify both the `-mno-power` and `-mno-powerpc` options, GCC will use only the instructions in the common subset of both architectures plus some special AIX common-mode calls, and will not use the MQ register.

`-mno-powerpc64`: Specifying this option prevents GCC from generating code that uses any of the instructions that are specific to the PowerPC-64 architecture.

`-mno-powerpc-gpopt`: Specifying this option prevents GCC from generating code that uses any of the instructions that are specific to the PowerPC architecture, including floating-point square root and the optional PowerPC architecture instructions in the general purpose group.

`-mno-powerpc-gfxopt`: Specifying this option prevents GCC from generating code that uses any of the instructions that are specific to the PowerPC architecture, including floating-point select and the optional PowerPC architecture instructions in the Graphics group.

`-mno-prototype`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to assume that all calls to variable argument functions are properly prototyped. Only calls to prototyped variable argument functions will set or clear bit 6 of the condition code register (CR) to indicate whether floating-point values were passed in the floating-point registers.

`-mno-regnames`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC not to emit register names in the assembly language output using symbolic forms.

`-mno-relocatable`: Specifying this option on embedded PowerPC systems tells GCC not to generate code that enables the program to be relocated to a different address at runtime. This minimizes the size of the resulting executable, such as a boot monitor or kernel.

`-mno-relocatable-lib`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC not to generate code that enables the program to be relocated to a different address at runtime.

`-mno-secure-plt`: Specifying this option prevents GCC from generating code that allows the linker to build shared libraries with nonexecutable .plt and .got sections.

`-mno-spe`: Specifying this option prevents GCC from generating code that uses the SPE simd instructions. This option replaces the `-mspe=NO` syntax.

`-mno-strict-align`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC not to assume that unaligned memory references will be handled by the system.

`-mno-string`: Specifying this option tells GCC to generate code that does not use the load string instructions or the store string word instructions. These instructions are generated by default on POWER systems, and not generated on PowerPC systems.

`-mno-sum-in-toc`: Specifying this option tells GCC to generate the same TOC as specified by the `-mfull-toc` option, but to generate code to calculate the sum of an address and a constant at runtime instead of putting that sum into the TOC. This option and the `-no-fp-in-toc` option are typically used if you see a linker error indicating that you have overflowed the TOC during final linking.

`-mno-swdiv`: Specifying this option prevents GCC from generating code that performs division in software.

`-mno-toc`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC not to assume that register `2` contains a pointer to a global area pointing to the addresses used in the program.

`-mno-update`: Specifying this option tells GCC to generate code that does not use the load or store instructions that update the base register to the address of the calculated memory location. (These instructions are generated by default.) If you use the `-mno-updateg` option, there is a small window between the time that the stack pointer is updated and when the address of the previous frame is stored, which means that code that walks the stack frame across interrupts or signals may get corrupted data.

`-mno-vrsave`: Specifying this option prevents GCC from generating vrsave instructions when generating AltiVec code.

`-mno-xl-compat`: Specifying this option prevents GCC from performing the code gymnastics necessary to conform to the calling conventions of the IBM XL compilers on AIX systems. This is the default.

`-mold-mnemonics`: Specifying this option tells GCC to use the assembler mnemonics defined for the POWER architecture. Instructions defined in only one architecture have only one mnemonic. GCC uses that mnemonic irrespective of which of these options is specified. GCC defaults to the mnemonics appropriate for the architecture that is in use. Unless you are cross-compiling, you should generally accept the default.

`-mpe`: Specifying this option tells GCC to support the IBM RS/6000 SP Parallel Environment (PE). Applications written to use message passing must be linked with special startup code to enable the application to run. The system must have PE installed in the standard location (`/usr/lpp/ppe.poe/`), or GCC's specs file must be overridden by using the `-specs=` option to specify the appropriate directory location. The Parallel Environment does not support threads, so the `-mpe` option and the `-pthread` option are incompatible.

`-mpopcntb`: Specifying this option enables GCC to generate the popcount and double-precision floating-point reciprocal estimate instructions that are implemented on the POWER5 and other PowerPC v20.20-compliant processors.

`-mprioritize-restricted-insns=`*`priority`*: Specifying this option controls the priority assigned to dispatch-slot restricted instructions during GCC's second scheduling pass. Possible values for *`priority`* are `0` (no priority), `1` (highest priority), and `2` (second-highest priority).

`-mpower`: Specifying this option tells GCC to generate code using instructions that are found only in the POWER architecture and to use the MQ register.

Note Specifying both of the `-mpower` and `-mpowerpc` options permits GCC to use any instruction from either architecture and to allow use of the MQ register. Both of these options should be specified when generating code for the Motorola MPC601 processor.

`-mpower2`: Specifying this option implies the `-mpower` option and also enables GCC to generate instructions that are present in the POWER2 architecture but not the original POWER architecture.

`-mpowerpc`: Specifying this option tells GCC to generate instructions that are found only in the 32-bit subset of the PowerPC architecture.

Note Specifying both of the `-mpowerpc` and `-mpower` options permits GCC to use any instruction from either architecture and to allow use of the MQ register. Both of these options should be specified when generating code for the Motorola MPC601 processor.

`-mpowerpc64`: Specifying this option tells GCC to generate any PowerPC instructions as well as the additional 64-bit instructions that are found in the full PowerPC-64 architecture and to treat GPRs (general purpose registers) as 64-bit, double-word quantities. GCC defaults to `-mno-powerpc64`.

`-mpowerpc-gfxopt`: Specifying this option implies the `-mpowerpc` option and also enables GCC to use the optional PowerPC architecture instructions in the Graphics group, including floating-point select.

`-mpowerpc-gpopt`: Specifying this option implies the `-mpowerpc` option and also enables GCC to use the optional PowerPC architecture instructions in the General Purpose group, including floating-point square root.

`-mprototype`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC not to assume that all calls to variable argument functions are properly prototyped. The compiler will insert an instruction before every nonprototyped call to set or clear bit 6 of the CR to indicate whether floating-point values were passed in the floating-point registers in case the function takes a variable number of arguments.

`-mregnames`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to emit register names in the assembly language output using symbolic forms.

`-mrelocatable`: Specifying this option on embedded PowerPC systems tells GCC to generate code that enables the program to be relocated to a different address at runtime. If you use the `-mrelocatable` option on any module, all objects linked together must be compiled with the `-mrelocatable` or `-mrelocatable-lib` options.

`-mrelocatable-lib`: Specifying this option on embedded PowerPC systems tells GCC to generate code that enables the program to be relocated to a different address at runtime. Modules compiled with the `-mrelocatable-lib` option can be linked with modules compiled without the `-mrelocatable` option.

`-msched-costly-dep=`*`dependence-type`*: Specifying this option identifies the dependencies that are considered costly during instruction scheduling. Possible values for *`dependence-type`* are `NO` (no dependencies are especially costly), `ALL` (all dependencies are costly), `NUMBER` (any dependency with latency greater than `NUMBER` is costly), `STORE_TO_LOAD` (any dependency from store to load is costly), and `TRUE_STORE_TO_LOAD` (a true dependency from store to load is costly).

`-msdata=abi`: Specifying this option tells GCC where to put small global and static data based on the rules for various platforms. Possible values for *`abi`* are `default` (behaves as if the `-msdata=sysv` option were specified, unless the `-meabi` option was also specified, in which case it behaves as if the `-msdata=eabi` option were specified), `eabi` (puts small initialized global and static data in the sdata2 section, puts small initialized non-const global and static data in the sdata section pointed to by register `13`, and puts small uninitialized global and static data in the sbss section adjacent to the stat section), `none` (puts all initialized global and static data in the data section and all uninitialized data in the bss section), and `sysv` (puts small global and static data in the sdata section, which is pointed to by register `r13`, and small uninitialized global and static data in the sbss section adjacent to the sdata section). The `-msdata=none` option can also be specified as `-mno-sdata`. The `-msdata=eabi` and `-msdata=sysv` options are incompatible with the `-mrelocatable` option. The `-msdata=eabi` option also sets the `-memb` option.

`-msdata-data`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to put small global and static data in the sdata section. Small uninitialized global and static data are put in the sbss section, but register `r13` is not used to address small data. This is the default behavior unless other `-msdata` options are specified.

`-msecure-plt`: Specifying this option enables GCC to generate code that allows the linker to build shared libraries with nonexecutable .plt and .got sections. This is a PowerPC 32-bit SYSV ABI option.

`-msim`: Specifying this option on embedded PowerPC systems tells GCC to assume that the startup module is called sim-crt0.o and that the standard C libraries are libsim.a and libc.a. This is the default for the `powerpc-*-eabisim` GCC build configuration.

`-msoft-float`: Specifying this option tells GCC to generate code that does not use the floating-point register set. Software floating-point emulation is provided if you use this option and pass it to GCC when linking.

`-mspe`: Specifying this option tells GCC to generate code that uses the SPE simd instructions, as needed. This option replaces the `-mspe=YES` syntax.

`-mstrict-align`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC to assume that unaligned memory references will be handled by the system.

`-mstring`: Specifying this option tells GCC to generate code that uses the load string instructions and the store string word instructions to save multiple registers and do small block moves. These instructions are generated by default on POWER systems, and not generated on PowerPC systems. Do not use `-mstring` on little endian PowerPC systems except for PPC740 and PPC750 systems, since those instructions usually do not work when the processor is in little endian mode. PPC740 and PPC750 permit the use of these instructions in little endian mode.

`-msvr4-struct-return`: Specifying this option tells GCC to return structures smaller than 8 bytes in registers, as specified by the SVR4 ABI.

`-mswdiv`: Specifying this option tells GCC to generate code that performs division in software using reciprocal estimates and iterative refinement. This can provide increased throughput, but requires the PowerPC graphics instruction set for single precision calculations and the FRE instruction for double-precision calculations, assumes that division calculations cannot generate user-visible traps, and assumes that possible values can not include infinite values, denormals, or zero denominators.

`-mtoc`: Specifying this option tells GCC to assume that register `2` contains a pointer to a global area pointing to the addresses used in the program on SVR4 and embedded PowerPC systems.

`-mtune=`*`cpu-type`*: Specifying this option tells GCC to set the instruction scheduling parameters for the machine type specified by *`cpu-type`*, but not to set the architecture type, register usage, or choice of mnemonics, as `-mcpu=`*`cpu-type`* would. The same values for *`cpu-type`* are used for both the `-mtune` and `-mcpu` options. If both are specified, the code generated will use the architecture, registers, and mnemonics set by the argument to `-mcpu`, but the scheduling parameters set by the argument to `-mtune`.

`-mupdate`: Specifying this option tells GCC to generate code that uses the load or store instructions that update the base register to the address of the calculated memory location. These instructions are generated by default.

`-mvrsave`: Specifying this option tells GCC to generate vrsave instructions when generating AltiVec code.

`-mvxworks`: Specifying this option on SVR4 and embedded PowerPC systems tells GCC that you are compiling for a VxWorks system. You have my sympathy.

`-mwindiss`: Specifying this option tells GCC that you are compiling for the WindISS simulation environment.

`-mxl-compat`: Specifying this option when compiling using an AIX-compatible ABI tells GCC to pass floating-point arguments to prototyped functions beyond the register save area (RSA) on the stack in addition to argument floating-point registers. When a subroutine is compiled without optimization, IBM AIX XL compilers access floating-point arguments that do not fit in the RSA from the stack. Because always storing floating-point arguments on the stack is inefficient and rarely needed, this option is not enabled by default and is only necessary when calling subroutines compiled without optimization by IBM AIX XL compilers.

`-myellowknife`: Specifying this option on embedded PowerPC systems tells GCC to assume that the startup module is called crt0.o and the standard C libraries are libyk.a and libc.a. *Yellowknife* is the name of an old Motorola evaluation board.

`-pthread`: Specifying this option tells GCC to add support for multithreading through the pthreads library. This option sets flags for both the preprocessor and the linker. (This option is not available in the GCC 4.x compilers.)

RS/6000 Options

See the section "PowerPC (PPC) Options" of this appendix for options relevant to using GCC to compile applications for IBM's RS/6000 family of workstations.

RT Options

The IBM RT was IBM's first RISC workstation, and saw little use outside academia. It originally ran an operating system called Academic Operating System (AOS), which was loosely based on BSD Unix, but was later the original deployment platform for IBM's AIX operating system. The IBM RT was also a primary development platform for the Mach operating system at Carnegie Mellon University.

■ **Note** Support for this processor family is not provided in GCC version 4.0. These options are therefore only of interest if you are using a version of GCC that is earlier than 4.x and that you are certain provides support for this processor.

GCC options available when compiling code for IBM RT systems are the following:

`-mcall-lib-mul`: Specifying this option tells GCC to call `lmul$$` for integer multiples.

`-mfp-arg-in-fpregs`: Specifying this option tells GCC to use a calling sequence in which floating-point arguments are passed in floating-point registers. This calling sequence is incompatible with the IBM calling convention. Note that varargs.h and stdarg.h will not work with floating-point operands if this option is specified.

`-mfp-arg-in-gregs`: Specifying this option tells GCC to use the normal calling convention for floating-point arguments. This is the default.

`-mfull-fp-blocks`: Specifying this option tells GCC to generate full-size floating-point data blocks, including the minimum amount of scratch space recommended by IBM. This is the default.

`-mhc-struct-return`: Specifying this option tells GCC to return structures of more than one word in memory, rather than in a register. This provides compatibility with the MetaWare HighC compiler. See the `-fpcc-struct-return` option for similar compatibility with the Portable C Compiler (PCC).

`-min-line-mul`: Specifying this option tells GCC to use an inline code sequence for integer multiplies. This is the default.

`-mminimum-fp-blocks`: Specifying this option tells GCC not to include extra scratch space in floating-point data blocks. This results in smaller code but slower execution, since scratch space must be allocated dynamically.

`-mnohc-struct-return`: Specifying this option tells GCC to return some structures of more than one word in registers, when convenient. This is the default.

`-mpcc-struct-return`: Specifying this option tells GCC to use return conventions that are compatible with PCC.

S/390 and zSeries Options

The S/390 is the latest in IBM's family of 360 and 370 systems that stretches back to the 1960s. The zSeries is IBM's name for its latest series of high-powered servers. S/390 systems can run Linux in virtual machines, just as they run other operating systems. The zSeries machines can run Linux directly.

Linux supports the S/390 and zSeries processor architecture and some devices that are specific to S/390 and zSeries environments. Linux on these systems enables you to take advantage of the fast I/O and reliability that are traditional features of S/390 and zSeries mainframe hardware.

GCC options available when compiling code for S/390 and zSeries systems are the following:

`-m31`: Specifying this option tells GCC to generate code that is compliant to the Linux for S/390 ABI. This is the default for s390 targets.

`-m64`: Specifying this option tells GCC to generate code that is compliant to the Linux for zSeries ABI. This allows GCC to generate 64-bit instructions. This is the default for s390x build targets.

`-march=`*`CPU-type`*: Specifying this option tells GCC the instruction set to use for code generation. Possible values for *`CPU-type`* are `g5`, `g6`, `z900`, and `z990`. When generating code using the z/Architecture instructions, the default value for *`CPU-type`* is `z900`. Otherwise, the default value for `CPU-type` is `g5`.

`-mbackchain`: Specifying this option tells GCC to generate code that maintains an explicit backchain within the stack frame. This backchain points to the caller's frame, and is currently needed to allow debugging. This is the default.

`-mdebug`: Specifying this option tells GCC to print additional debug information when compiling.

`-mesa`: Specifying this option tells GCC to generate code using only the instructions that are available on the ESA/90. This option is the default when generating Linux code for the ESA/90 ABI.

`-mfused-madd`: Specifying this option tells GCC to generate code that uses the floating-point multiply and accumulate instructions when they are available. These instructions are generated by default if they are available.

`-mguard-size=`*`GUARD-SIZE`*: See the `-mstack-size` option for more information.

`-mhard-float`: Specifying this option tells GCC to use the hardware floating-point instructions and registers for floating-point operations. GCC generates the appropriate IEEE floating-point instructions. This is the default.

`-mlong-double-64`: Specifying this option tells GCC to use the default size of 64 bits for both the `long double` and `double` datatypes. This is the default.

`-mlong-double-128`: Specifying this option tells GCC to double the size of the `long double` datatype to 128 bits. By default it is 64 bits and is therefore internally equivalent to the `double` datatype.

`-mmvcle`: Specifying this option tells GCC to generate code using the mvcle instruction to perform block moves.

`-mno-backchain`: Specifying this option tells GCC not to generate code that maintains an explicit backchain within the stack frame that points to the caller's frame. Not maintaining a backchain prevents debugging.

`-mno-debug`: Specifying this option tells GCC not to print additional debug information when compiling. This is the default.

`-mno-fused-madd`: Specifying this option prevents GCC from generating code that uses the floating-point multiply and accumulate instructions, even when they are available.

`-mno-mvcle`: Specifying this option tells GCC not to generate code using the mvcle instruction to perform block moves, but to use an `mvc` loop instead. This is the default.

`-mno-packed-stack`: Specifying this option tells GCC not to use the packed stack layout, and to therefore only use fields in the register save area for their default purposes—unused space in this area is wasted. Code generated with this option is compatible with code that is not generated with this option as long as the `-mbackchain` option is not used.

`-mno-small-exec`: Specifying this option tells GCC not to generate code using the bras instruction to do subroutine calls. This is the default, causing programs compiled with GCC to use the basr instruction instead, which does not have a 64K limitation.

`-mno-tpf-trace`: Specifying this option prevents GCC from generating code that adds 390/TPF (Transaction Processing Facility) branches to trace routines in that operating system. This is the default, even when compiling for 390/TPF.

`-mpacked-stack`: Specifying this option tells GCC to use the packed stack layout where register save slots are densely packed so that unused space can be used for other purposes. Code generated with this option is compatible with code that is not generated with this option as long as the `-mbackchain` option is not used.

`-msmall-exec`: Specifying this option tells GCC to generate code using the bras instruction to do subroutine calls. This only works reliably if the total executable size does not exceed 64K.

`-msoft-float`: Specifying this option tells GCC not to use the hardware floating-point instructions and registers for floating-point operations. Functions in libgcc.a will be used to perform floating-point operations.

`-mstack-size=`*`stack-size`*: Specifying this option causes GCC to emit code in the function prologue that triggers a trap if the stack size is a specified number of bytes above `stack-size`. This option must be used with the `-mstack-guard=`*`guard-size`* option, whose `guard-size` argument identifies the number of bytes above `stack-size` at which a trap will be triggered. The value specified for *`stack-size`* must be greater than *`guard-size`* and not exceed 64K. Both `stack-size` and `guard-size` values must be powers of 2.

`-mtpf-trace`: Specifying this option tells GCC to generate code that adds 390/TPF (Transaction Processing Facility) branches to trace routines in that operating system. This option is off by default, even when compiling for 390/TPF.

`-mvcle`: Specifying this option tells GCC to generate using the mvcle instruction to perform block moves, as needed. This option is off by default.

`-mwarn-dynamicstack`: Specifying this option causes GCC to emit a warning if a function calls alloca or uses dynamically sized arrays. This option can be useful when compiling any executables that have limited stack size, such as the Linux kernel.

`-mwarn-framesize=`*`FRAMESIZE`*: Specifying this option causes GCC to emit a warning if a function exceeds the specified *`FRAMESIZE`*, and is intended to help identify function during compilation that might cause a stack overflow. This option can be useful when compiling any executables that have limited stack size, such as the Linux kernel.

`-mzarch`: Specifying this option tells GCC to generate code using only the instructions that are available on the z/Architecture. This option is the default when generating Linux code for the zSeries ABI.

SH Options

SuperH (SH) processors from Hitachi, Renesas, and others are powerful and popular processors for use in embedded systems. The SH1 and SH2 processors are 16-bit processors, the SH3, SH4, and SH4a are 32-bit processors, and the new SH5 is a fast 64-bit processor.

GCC options available when compiling code for systems using SH processors are the following:

`-m1`: Specifying this option tells GCC to generate code for the SH1 processor.

`-m2`: Specifying this option tells GCC to generate code for the SH2 processor.

`-m2e`: Specifying this option tells GCC to generate code for the SH2e processor.

`-m3`: Specifying this option tells GCC to generate code for the SH3 processor.

`-m3e`: Specifying this option tells GCC to generate code for the SH3e processor.

`-m4-nofpu`: Specifying this option tells GCC to generate code for SH4 processors without a floating-point unit.

`-m4-single-only`: Specifying this option tells GCC to generate code for SH4 processors with a floating-point unit that only supports single precision arithmetic.

`-m4-single`: Specifying this option tells GCC to generate code for SH4 processors, assuming the floating-point unit is in single precision mode by default.

`-m4`: Specifying this option tells GCC to generate code for the SH4 processor.

`-m4a-nofpu`: Specifying this option tells GCC to generate code for SH4a processors without a floating-point unit.

`-m4a-single-only`: Specifying this option tells GCC to generate code for SH4a processors with a floating-point unit that only supports single precision arithmetic.

`-m4a-single`: Specifying this option tells GCC to generate code for SH4a processors, assuming the floating-point unit is in single precision mode by default.

`-m4a`: Specifying this option tells GCC to generate code for the SH4a processor.

`-m4al`: Specifying this option tells GCC to generate code for SH4a1-DSP or for SH4a processors without a floating-point unit. This option is the same as the `-m4a-nofpu` except that it also specifies the `-dsp` option.

`-madjust-unroll`: Specifying this option tells GCC to limit loop unrolling in order to avoid thrashing on target registers. For this option to have any effect, the GCC code base must support the `TARGET_ADJUST_UNROLL_MAX` hook.

`-mb`: Specifying this option tells GCC to compile code for an SH processor in big endian mode.

`-mbigtable`: Specifying this option tells GCC to use 32-bit offsets in switch tables. The default is to use 16-bit offsets.

`-mdalign`: Specifying this option tells GCC to align `doubles` at 64-bit boundaries. This changes the calling conventions, and therefore some functions from the standard C library will not work unless you also recompile the standard C library with the `-mdalign` option.

`-mdiv=`*`strategy`*: Specifying this option tells GCC the division strategy that it should use when generating code using Super-Hitachi's SHmedia SIMD instruction set. Valid values for *`strategy`* are `call`, `call2`, `fp`, `inv`, `inv:minlat`, `inv20u`, `inv20l`, `inv:call`, `inv:call2`, and `inv:fp`.

`-mdivsi3_libfunc=`*name*: Specifying this option tells GCC the name of the library to use for 32-bit signed division. This only affects the actual function calls; GCC expects the same behavior and register-use conventions as if this option was not specified.

`-mfmovd`: Specifying this option tells GCC to enable the use of the fmovd instruction.

`-mgettrcost=`*number*: Specifying this option tells GCC the cost (*number*) to assume when scheduling the gettr instruction. The default value for *number* is `2` if the `-mpt-fixed` option is specified, and `100` otherwise.

`-mhitachi`: Specifying this option tells GCC to comply with the SH calling conventions defined by Hitachi.

`-mieee`: Specifying this option tells GCC to generate IEEE-compliant floating-point code.

`-mindexed-addressing`: Specifying this option enables GCC to use the indexed addressing mode for SHmedia32/SHcompact, which enables the implementation of 64-bit MMUs but is only safe when the hardware and/or OS implements 32-bit wraparound semantics, which is not currently supported on any hardware implementation.

`-minvalid-symbols`: Specifying this option tells GCC to assume that symbols may be invalid due to cross-basic-block common subexpression elimination or hoisting.

`-misize`: Specifying this option tells GCC to include instruction size and location in the assembly code.

`-ml`: Specifying this option tells GCC to compile code for an SH processor in little endian mode.

`-no-minvalid-symbols`: Specifying this option tells GCC to assume that all symbols are valid. This option is the default.

`-mno-pt-fixed`: Specifying this option tells GCC to assume that pt* instructions may generate a trap.

`-mno-renasas`: Specifying this option tells GCC to comply with the SH calling conventions used before the Renesas calling conventions were available. This option is the default for all SH toolchain targets except for sh-symbianelf.

`-mnomacsave`: Specifying this option tells GCC to mark the MAC register as call-clobbered, even if the `-mhitachi` option is also used.

`-mpadstruct`: This is a deprecated option that pads structures to multiples of 4 bytes, which is incompatible with the SH ABI.

`-mprefergot`: Specifying this option tells GCC to emit function calls using the global offset table instead of the procedure linkage table when generating position-independent code.

`-mpt-fixed`: Specifying this option tells GCC that no pt* instructions will generate a trap. This improves scheduling but is unsafe on current hardware.

`-mrelax`: Specifying this option tells GCC to shorten some address references at link time, when possible. Specifying this option passes the `-relax` option to the linker.

`-mrenasas`: Specifying this option tells GCC to comply with the SH calling conventions defined by Renesas.

`-mspace`: Specifying this option tells GCC to optimize for size instead of speed. This option is implied by the `-Os` optimization option.

`-multcost=`*`number`*: Specifying this option tells GCC the cost (an integer *number*) to assume for a multiply instruction when scheduling.

`-musermode`: Specifying this option tells GCC to generate a library function call that invalidates instruction cache entries after fixing up a trampoline. The function call does not assume that it can write to the whole memory address space. This is the default when GCC was built for the sh-*-linux* target.

SPARC Options

The SPARC is a fast 32-bit processor architecture originally developed by Sun Microsystems and used in all of its workstations, single-board computers (SBCs), and other embedded hardware since the late 1980s. The SPARC processor has also been licensed to a number of other workstation and embedded hardware vendors. The latest generation of SPARC processors, the UltraSPARC family, are 64-bit processors.

GCC options available when compiling code for systems using SPARC processors are the following:

`-m32`: Specifying this option tells GCC to generate code for a 32-bit environment. The 32-bit environment sets `int`, `long`, and `pointer` to 32 bits. This option is only supported when compiling for SPARC V9 processors in 64-bit environments.

`-m64`: Specifying this option tells GCC to generate code for a 64-bit environment. The 64-bit environment sets `int` to 32 bits and `long` and `pointer` to 64 bits.

`-mapp-regs`: Specifying this option tells GCC to be fully SVR4 ABI compliant at the cost of some performance loss. Libraries and system software should be compiled with this option to maximize compatibility.

`-mcmodel=`*`code-model`*: Specifying this option identifies the code model that GCC should use when generating code. Valid values for *code-model* are `embmedany`, `medany`, `medlow`, and `medmid`. See the explanations of each of these option/*`code-model`* pairs for detailed information about the code model that they implement.

`-mcmodel=embmedany`: Specifying this option tells GCC to generate code for the Medium/Anywhere code model for embedded systems, which assumes a 32-bit text and a 32-bit data segment, both starting anywhere (determined at link time). Register `%g4` points to the base of the data segment. Pointers are still 64 bits. Programs are statically linked; PIC is not supported. This option is only supported when compiling for SPARC V9 processors in 64-bit environments.

`-mcmodel=medany`: Specifying this option tells GCC to generate code for the Medium/Anywhere code model, where the program may be linked anywhere in the address space, the text segment must be less than 2GB, and the data segment must be within 2GB of the text segment. Pointers are 64 bits. This option is only supported when compiling for SPARC V9 processors in 64-bit environments.

`-mcmodel=medlow`: Specifying this option tells GCC to generate code for the Medium/Low code model, where a program must be linked in the low 32 bits of the address space. Pointers are 64 bits. Programs can be statically or dynamically linked. This option is only supported when compiling for SPARC V9 processors in 64-bit environments.

`-mcmodel=medmid`: Specifying this option tells GCC to generate code for the Medium/Middle code model, where the program must be linked in the low 44 bits of the address space, the text segment must be less than 2GB, and the data segment must be within 2GB of the text segment. Pointers are 64 bits. This option is only supported when compiling for SPARC V9 processors in 64-bit environments.

`-mcpu=CPU-type`: Specifying this option tells GCC to set the instruction set, register set, and instruction scheduling parameters for machine type `CPU-type`. Supported values for *`CPU-type`* are `cypress`, `f930`, `f934`, `hypersparc`, `niagra`, `sparclet`, `sparclite`, `sparclite86x`, `supersparc`, `tsc701`, `ultrasparc`, `ultrasparc3`, `v6`, `v7`, `v8`, and `v9`. Default instruction scheduling parameters are used for values that select an architecture and not an implementation. These are `sparclet`, `sparclite`, `v7`, `v8`, and `v9`. By default, GCC generates code for the `v7` `CPU-type`.

Table B-1 shows each supported architecture and its supported implementations.

Table B-1. *SPARC Architectures and Associated CPU-type Values*

Values	Architecture
`v7`	cypress
`v8`	supersparc, hypersparc
`sparclite`	f930, f934, sparclite86x
`sparclet`	tsc701
`v9`	niagra, ultrasparc, ultrasparc3

`-mcypress`: Specifying this option (or the `-mcpu=cypress` option) tells GCC to optimize code for the Cypress CY7C602 chip, as used in the SPARCStation/SPARCServer 3xx series. This is also appropriate for the older SPARCStation 1, 2, IPX, and so on. This is the default. This option is deprecated—the more general `-mCPU=CPU-type` option should be used instead. (This option is not available in the GCC 4.x compilers.)

`-mfaster-structs`: Specifying this option tells GCC to assume that structures should have 8-byte alignment. This enables the use of pairs of ldd and std instructions for copies in structure assignment, instead of twice as many ld and st pairs. However, the use of this changed alignment directly violates the SPARC ABI and is therefore intended only for use on targets where developers acknowledge that their resulting code will not be directly in line with the rules of the ABI.

`-mflat`: Specifying this option tells GCC not to generate save/restore instructions and instead to use a flat, or single-register window, calling convention. This model uses `%i7` as the frame pointer and is compatible with code that does not use this calling convention. Regardless of the calling conventions used, the local registers and the input registers (`0`–`5`) are still treated as "call-saved" registers and will be saved on the stack as necessary.

`-mhard-float` | `-mfpu`: Specifying either of these options tells GCC to generate output containing floating-point instructions. This is the default.

`-mhard-quad-float`: Specifying this option tells GCC to generate output that contains quad-word (`long double`) floating-point instructions.

■ **Note** No current SPARC implementations provide hardware support for the quad-word floating-point instructions. They all invoke a trap handler for one of these instructions, where the trap handler then emulates the effect of the instruction. Because of the overhead of the trap handler, this is much slower than calling the ABI library routines. For this reason, the `-msoft-quad-float` option is the default.

`-mimpure-text`: Specifying this option along with the standard GCC `-shared` option tells GCC not to pass `-z text` to the linker when linking a shared object, which enables you to link position-independent code into shared objects. This option is only available when compiling for SunOS or Solaris.

`-mlittle-endian`: Specifying this option tells GCC to generate code for a processor running in little endian mode. This option is only supported when compiling for SPARC V9 processors in 64-bit environments and is not supported in most standard GCC configurations, such as Linux and Solaris.

`-mno-app-regs`: Specifying this option tells GCC to generate output using the global registers `2` through `4`, which the SPARC SVR4 ABI reserves for applications. This is the default.

`-mno-faster-structs`: Specifying this option tells GCC not to make any assumptions about structure alignment, and to use the ld and st instructions when making copies during structure assignment.

`-mno-flat`: Specifying this option tells GCC to use save/restore instructions as its calling convention (except for leaf functions). This option is the default.

`-mno-stack-bias`: Specifying this option tells GCC to assume that no offset is used when making stack frame references. This option is only supported when compiling for SPARC V9 processors in 64-bit environments.

`-mno-unaligned-doubles`: Specifying this option tells GCC to assume that `doubles` have 8-byte alignment. This is the default.

`-mno-v8plus`: Specifying this option tells GCC not to assume that in and out registers are 6 bits when generating code which essentially means that code generation conforms to the V8 ABI.

`-mno-vis`: Specifying this option prevents GCC from generating code that uses the UltraSPARC VIS (Visual Instruction Set) instructions. This is the default.

`-msoft-float` | `-mno-fpu`: Specifying either of these options tells GCC to generate output that contains library calls for floating-point operations. These libraries are not provided as part of GCC (except for the embedded GCC build targets sparc-*-aout and sparclite-*-*), but are normally found on the target system, and should be resolved by the C loader on the target machine if they are available. When using this option and cross-compiling, you must provide suitable libraries containing at least function stubs on the host system, or these function calls will be identified as unresolved.

■ **Tip** Specifying this option also changes the calling convention used in the output file. You must therefore compile all of the modules of your program with this option, including any libraries that you reference. You must also compile libgcc.a, the library that comes with GCC, with this option in order to be able to use this option.

`-msoft-quad-float`: Specifying this option tells GCC to generate output containing library calls for quad-word (`long double`) floating-point instructions. The functions called are those specified in the SPARC ABI. This is the default.

`-msparclite`: Specifying this option (or the `-mcpu=sparclite` option) tells GCC to generate code for SPARClite processors. This adds the integer multiply and integer divide step and scan (ffs) instructions that exist in SPARClite but not in SPARC V7. This option is deprecated; the more general `-mCPU=`*`cpu-type`* option should be used instead.

`-mstack-bias`: Specifying this option tells GCC to assume that the stack pointer and the frame pointer (if present) are offset by -2047, which must be added back when making stack frame references. This option is only supported when compiling for SPARC V9 processors in 64-bit environments.

`-msupersparc`: Specifying this option (or the `-mcpu=supersparc` option) tells GCC to optimize code for the SuperSPARC processor, as used in the SPARCStation 10, 1000, and 2000 series. This option also enables use of the full SPARC V8 instruction set. This option is deprecated; the more general `-mCPU=CPU-type` option should be used instead.

`-mtune=`*`cpu-type`*: Specifying this option tells GCC to set the instruction scheduling parameters for machine type *`cpu-type`*, but not to set the instruction set or register set as would the option `-mcpu=CPU-type`. The same values for `-mcpu=`*`cpu-type`* can be used with the `-mtune=`*`cpu-type`* option, but the only useful values are those that select a particular processor implementation: `cypress`, `f930`, `f934`, `hypersparc`, `sparclite86x`, `supersparc`, `tsc701`, and `ultrasparc`.

`-munaligned-doubles`: Specifying this option causes GCC not to assume that `doubles` are 8-byte aligned. GCC assumes that `doubles` have 8-byte alignment only if they are contained in another type, or if they have an absolute address. Otherwise, it assumes they have 4-byte alignment. Specifying this option avoids some rare compatibility problems with code generated by other compilers, but results in a performance loss, especially for floating-point code.

`-mv8plus`: Specifying this option tells GCC to generate code for the SPARC V8+ ABI. The only difference from the V8 ABI is that the global in and out registers are 64 bits wide. This option is enabled by default when compiling for SPARC V9 processors in 32-bit mode.

`-mvis`: Specifying this option tells GCC to generate code that uses the UltraSPARC VIS instructions, as needed.

`-pthread | -pthreads`: Specifying either of these options causes GCC to add support for multithreading using the POSIX thread library. Specifying this option sets other options for both the preprocessor and the linker.

`-threads`: Specifying this option causes GCC to add support for multithreading using the Solaris thread library. Specifying this option sets other options for both the preprocessor and the linker.

System V Options

System V Release 4, or SVR4, was a landmark release of the Unix operating system from AT&T and is the conceptual foundation for Unix-like operating systems such as Solaris and Linux. Alas, poor BSD, I knew thee well . . .

GCC options specific to compiling code for systems running SVR4 are the following:

`-G`: Specifying this option tells GCC to create a shared object. This option is deprecated, as it is easily confused with other `-G` options. You should use the more comprehensible `-symbolic` or `-shared` options instead.

`-Qn`: Specifying this option tells GCC not to add `.ident` directives identifying tool versions in the output assembly file. This is the default.

`-Qy`: Specifying this option tells GCC to identify the versions of each tool used by the compiler by adding an `.ident` assembler directive in the assembler output.

`-Ym,`*`dir`*: Specifying this option tells GCC to look in the directory *`dir`* to find the m4 preprocessor. The assembler uses this option.

`-YP,`*`dir`*: Specifying this option tells GCC to search the directory *`dir`*, and no others, for libraries specified with -l.

TMS320C3x/C4x Options

The TMS320C3x and TMS320C4x processors are digital signal processors from Texas Instruments.

GCC options specific to compiling code for systems running TMS320C3x and TMS320C4x processors are the following:

`-mbig` | `-mbig-memory`: Specifying either of these options tells GCC to generate code for the big memory model. Using the big memory model makes no assumptions about code data size and requires reloading of the `dp` register for every direct memory access. This is the default.

`-mbk`: Specifying this option tells GCC to allow allocation of general integer operands into the block count register `bk`.

`-mcpu=`*`cpu-type`*: Specifying this option tells GCC to set the instruction set, register set, and instruction scheduling parameters for machine type `CPU-type`. Supported values for `CPU-type` are `c30`, `c31`, `c32`, `c40`, and `c44`. The default is `c40`, which generates code for the TMS320C40.

`-mdb`: Specifying this option tells GCC to generate code that uses the decrement and branch, DBcond(D), instruction. This is the default for TMS320C4x processors.

■ **Note** GCC will try to reverse a loop so that it can utilize the decrement and branch instruction, but will give up if there is more than one memory reference in the loop. Thus a loop where the loop counter is decremented can generate slightly more efficient code in cases where the RPTB instruction cannot be utilized.

`-mdp-isr-reload` | `-mparanoid`: Specifying either of these options tells GCC to force the DP register to be saved on entry to an interrupt service routine (ISR), reloaded to point to the data section, and restored on exit from the ISR. This should not be necessary unless someone has violated the small memory model by modifying the DP register, such as within an object library.

`-mfast-fix`: Specifying this option tells GCC to accept the results of the C3x/C4x FIX instruction which, when converting a floating-point value to an integer value, chooses the nearest integer less than or equal to the floating-point value rather than to the nearest integer. Thus, if the floating-point number is negative, the result will be incorrectly truncated and additional code will be necessary to detect and correct this case.

`-mloop-unsigned`: Specifying this option tells GCC to use an unsigned iteration count. This limits loop iterations to 2**31 + 1, since these instructions test if the iteration count is negative in order to terminate a loop.

`-mmemparm`: Specifying this option tells GCC to generate code that uses the stack for passing arguments to functions.

`-mmpyi`: Specifying this option when compiling code for TMS320C3x processors tells GCC to use the 24-bit MPYI instruction for integer multiplies instead of a library call to guarantee 32-bit results. Note that if one of the operands is a constant, the multiplication will be performed using shifts and adds.

`-mno-bk`: Specifying this option tells GCC not to allow allocation of general integer operands into the block count register `bk`.

`-mno-db`: Specifying this option tells GCC not to generate code that uses the decrement and branch, DBcond(D), instruction. This is the default for TMS320C3x processors.

`-mno-fast-fix`: Specifying this option tells GCC to generate additional code to correct the results of the C3x/C4x FIX instruction which, when converting a floating-point value to an integer value, chooses the nearest integer less than or equal to the floating-point value rather than to the nearest integer. Thus, if the floating-point number is negative, the result will be incorrectly truncated. Specifying this option generates the additional code necessary to detect and correct this case.

`-mno-loop-unsigned`: Specifying this option tells GCC not to use an unsigned iteration count, which might artificially limit loop iterations to 2**31 + 1, since these instructions test if the iteration count is negative in order to terminate a loop.

`-mno-mpyi`: Specifying this option tells GCC to use a library call for integer multiplies. When compiling code for the TMS320C3x processor, squaring operations are performed inline instead of through a library call.

`-mno-parallel-insns`: Specifying this option tells GCC not to generate parallel instructions.

`-mno-parallel-mpy`: Specifying this option tells GCC not to generate MPY||ADD and MPY||SUB parallel instructions. This generally minimizes code size, though the lack of parallelism may negatively impact performance.

`-mno-rptb`: Specifying this option tells GCC not to generate repeat block sequences using the `RPTB` instruction for zero overhead looping.

`-mno-rpts`: Specifying this option tells GCC not to use the single-instruction repeat instruction RPTS. This is the default, because interrupts are disabled by the RPTS instruction.

`-mparallel-insns`: Specifying this option tells GCC to enable generating parallel instructions. This option is implied when the `-O2` optimization option is specified.

`-mparallel-mpy`: Specifying this option tells GCC to generate MPY||ADD and MPY||SUB parallel instructions if the `-mparallel-insns` option is also specified. These instructions have tight register constraints that can increase code size for large functions.

`-mregparm`: Specifying this option tells GCC to generate code that uses registers (whenever possible) for passing arguments to functions. This is the default.

`-mrptb`: Specifying this option tells GCC to enable the generation of repeat block sequences using the RPTB instruction for zero overhead looping. The RPTB construct is only used for innermost loops that do not call functions or jump across the loop boundaries. There is no advantage to having nested RPTB loops due to the overhead required to save and restore the `RC`, `RS`, and `RE` registers. This is option enabled by default with the `-O2` optimization option.

`-mrpts=`*`count`*: Specifying this option tells GCC to enable the use of the single instruction repeat instruction RPTS. If a repeat block contains a single instruction, and the loop count can be guaranteed to be less than the value *`count`*, GCC will emit an RPTS instruction instead of an RPTB instruction. If no value is specified, an RPTS will be emitted even if the loop count cannot be determined at compile time. Note that the repeated instruction following the RPTS instruction does not have to be reloaded from memory during each iteration, thus freeing up the CPU buses for operands.

`-msmall` | `-msmall-memory`: Specifying either of these options tells GCC to generate code for the small memory model. The small memory model assumes that all data fits into one 64K word page. At runtime, when using the small memory model, the DP register must be set to point to the 64K page containing the bss and data program sections.

`-mti`: Specifying this option tells GCC to try to emit an assembler syntax that the Texas Instruments assembler (asm30) is happy with. This also enforces compatibility with the ABI employed by the TI C3x C compiler. For example, `long double`s are passed as structures rather than in floating-point registers.

V850 Options

NEC's V850 family of 32-bit RISC microcontrollers is designed for embedded real-time applications such as motor control, process control, industrial measuring equipment, automotive systems, and so on.

GCC options specific to compiling code for systems running V850 processors are the following:

`-mapp-regs`: Specifying this option tells GCC to use `r2` and `r5` in the code that it generates. This is the default.

`-mbig-switch`: Specifying this option tells GCC to generate code suitable for big switch tables. You should only use this option if the assembler or linker complains about out-of-range branches within a switch table.

`-mdisable-callt`: Specifying this option prevents GCC from using the callt instruction in the code that it generates for the V850e and V850e1 processors.

`-mep`: Specifying this option tells GCC to optimize basic blocks that use the same index pointer four or more times to copy pointers into the `ep` register, using the shorter sld and sst instructions instead. This option is on by default if you specify any of the optimization options.

`-mlong-calls`: Specifying this option tells GCC to treat all calls as being far. If calls are assumed to be far, GCC will always load the function's address into a register and make an indirect call through the pointer.

`-mno-app-regs`: Specifying this option tells GCC to treat `r2` and `r5` as fixed registers in the code that it generates.

`-mno-disable-callt`: Specifying this option enables GCC to use the callt instruction in the code that it generates for the V850e and V850e1 processors as needed. This is the default.

`-mno-ep`: Specifying this option tells GCC not to optimize basic blocks that use the same index pointer four or more times to copy pointers into the `ep` register.

`-mno-long-calls`: Specifying this option tells GCC to treat all calls as being near, requiring no special handling.

`-mno-prolog-function`: Specifying this option tells GCC not to use external functions to save and restore registers at the prologue and epilogue of a function.

`-mprolog-function`: Specifying this option tells GCC to use external functions to save and restore registers at the prologue and epilogue of a function. These external functions are slower but use less code space if more than one function saves the same number of registers. This option is enabled by default if you specify any of the optimization options.

`-msda=`*n*: Specifying this option tells GCC to put static or global variables whose size is *n* bytes or less into the small data area that register `gp` points to. The small data area can hold up to 64K.

`-mspace`: Specifying this option tells GCC to try to make the code as small as possible by activating the `-mep` and `-mprolog-function` options.

`-mtda=`*n*: Specifying this option tells GCC to put static or global variables whose size is *n* bytes or less into the tiny data area that register `ep` points to. The tiny data area can hold up to 256 bytes in total (128 bytes for byte references).

`-mv850`: Specifying this option tells GCC that the target processor is the V850 and defines the preprocessor symbols `__v850` and `__v8501__`.

`-mv850e`: Specifying this option tells GCC that the target processor is the V850e, and defines the preprocessor symbol `__v850e__` as well as the standard symbols `__v850` and `__v8501__`.

`-mv850e1`: Specifying this option tells GCC that the target processor is the V850E1, and defines the preprocessor symbols `__v850e1__` and `__v850e__`, as well as the standard symbols `__v850` and `__v8501__`.

■ **Note** If none of the `-mv850`, `-mv850e`, or `-mv850e1` options are specified, GCC will select a default processor depending on how it was built and define the appropriate symbols. The standard symbols `__v850` and `__v8501__` are always defined.

`-mzda=`*n*: Specifying this option tells GCC to put static or global variables whose size is *n* bytes or less into the first 32K of memory.

VAX Options

VAX was Digital Equipment Corporation's (DEC) workstations and minicomputers for years, and shipped with a quaint operating system called VMS.

GCC options available when compiling code for VAX systems are the following:

`-mg`: Specifying this option tells GCC to generate code for g-format floating-point numbers instead of d-format floating-point numbers.

`-mgnu`: Specifying this option tells GCC to generate all possible jump instructions. This option assumes that you will assemble your code using the GNU assembler.

`-munix`: Specifying this option tells GCC not to generate certain jump instructions (aobleq and so on) that the Unix assembler for the VAX cannot handle across long ranges.

Xstormy16 Options

Sanyo's Xstormy16 processor is designed for memory-constrained applications, and is often used in home appliances and audio systems.

The sole GCC option available when compiling code for Xstormy16 processors is the following:

`-msim`: Specifying this option tells GCC to choose startup files and linker scripts that are suitable for an Xstormy16 simulator.

Xtensa Options

The Xtensa architecture is designed to support many different configurations—there is no such thing as a generic Xtensa processor. Each instance of the Xtensa processor architecture is uniquely tuned by the system designer to ideally fit the application targeted by a specific system-on-a-chip (SoC) implementation, by using the Xtensa Processor Generator or by selecting from a broad selection of predefined standard RISC microprocessor features.

GCC's default options can be set to match a particular Xtensa configuration by copying a configuration file into the GCC sources when building GCC.

GCC options available to override the default values specified in the configuration file when compiling code for Xtensa processors are the following:

`-mbig-endian`: Specifying this option tells GCC to generate code using big endian byte ordering. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mbooleans`: Specifying this option tells GCC to enable support for the Boolean register file used by Xtensa coprocessors. This is not typically useful by itself but may be required for other options that make use of the Boolean registers (such as the floating-point options). This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mconst16`: Specifying this option enables GCC to generate code that uses the const16 instruction to load values as needed. The const16 instruction is not a standard Tensilica instruction but replaces the standard l32r instruction when this option is specified. This option is the default if the l32r instruction is not available.

`-mdensity`: Specifying this option tells GCC to enable the use of the optional Xtensa code density instructions. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mfused-madd`: Specifying this option tells GCC to enable the use of fused multiply/add and multiply/subtract instructions in the floating-point option. This has no effect if the floating-point option is not also enabled. This option should not be used when strict IEEE 754–compliant results are required, since the fused multiply/add and multiply/subtract instructions do not round the intermediate result, and therefore produce results with more precision than is specified by the IEEE standard.

`-mhard-float`: Specifying this option tells GCC to enable the use of the floating-point option. GCC generates floating-point instructions for 32-bit float operations. 64-bit `double` operations are always emulated with calls to library functions. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mlittle-endian`: Specifying this option tells GCC to generate code using little endian byte ordering. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mlongcalls`: Specifying this option tells GCC to instruct the assembler to translate direct calls to indirect calls unless it can determine that the target of a direct call is in the range allowed by the call instruction. This translation typically occurs for calls to functions in other source files. This option should be used in programs where the call target can potentially be out of range. This option is implemented in the assembler, not the compiler, so the assembly code generated by GCC will still show direct call instructions; you will have to examine disassembled object code in order to see the actual instructions. This option also causes the assembler to use an indirect call for every cross-file call, not just those that really will be out of range.

`-mmac16`: Specifying this option tells GCC to enable the use of the Xtensa MAC16 option. GCC will generate MAC16 instructions from standard C code, with the limitation that it will use neither the `MR` register file nor any instruction that operates on the `MR` registers. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mminmax`: Specifying this option tells GCC to enable the use of the optional minimum and maximum value instructions. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mmul16`: Specifying this option tells GCC to enable the use of the 16-bit integer multiplier option. GCC will generate 16-bit multiply instructions for multiplications of 16 bits or smaller in standard C code. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mmul32`: Specifying this option tells GCC to enable the use of the 32-bit integer multiplier option. GCC will generate 32-bit multiply instructions for multiplications of 32 bits or smaller in standard C code. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-booleans`: Specifying this option tells GCC to disable support for the Boolean register file used by Xtensa coprocessors. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-const16`: Specifying this option prevents GCC from generating code that uses the const16 instruction to load values, using the standard l32r instruction instead.

`-mno-density`: Specifying this option tells GCC to disable the use of the optional Xtensa code density instructions. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-longcalls`: Specifying this option tells GCC not to translate direct calls to indirect calls. This is the default. See the description of the `-mlongcalls` option for implementation details.

`-mno-mac16`: Specifying this option tells GCC to disable the use of the Xtensa MAC16 option. GCC will translate 16-bit multiply/accumulate operations to a combination of core instructions and library calls, depending on whether any other multiplier options are enabled. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-fused-madd`: Specifying this option tells GCC to disable the use of fused multiply/add and multiply/subtract instructions in the floating-point option. Disabling fused multiply/add and multiply/subtract instructions forces the compiler to use separate instructions for the multiply and add/subtract operations. This may be desirable in some cases where strict IEEE 754–compliant results are required, since the fused multiply/add and multiply/subtract instructions do not round the intermediate result, thereby producing results with more precision than is specified by the IEEE standard. Disabling fused multiply/add and multiply/subtract instructions also ensures that the program output is not sensitive to the compiler's ability to combine multiply and add/subtract operations.

`-mno-minmax`: Specifying this option tells GCC to disable the use of the optional minimum and maximum value instructions. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-mul16`: Specifying this option tells GCC to disable the use of the 16-bit integer multiplier option. GCC will either use 32-bit multiply or MAC16 instructions if they are available, or generate library calls to perform the multiply operations using shifts and adds. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-mul32`: Specifying this option tells GCC to disable the use of the 32-bit integer multiplier option. GCC will generate library calls to perform the multiply operations using either shifts and adds or 16-bit multiply instructions if they are available. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-nsa`: Specifying this option tells GCC to disable the use of the optional normalization shift amount (NSA) instructions to implement the built-in `ffs` function. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-serialize-volatile`: Specifying this option tells GCC not to use MEMW instructions before volatile memory references in order to guarantee sequential consistency, resulting in decreased code size.

`-mno-sext`: Specifying this option tells GCC to disable the use of the optional sign extend (SEXT) instruction. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mno-target-align`: Specifying this option tells GCC to not have the assembler automatically align instructions, increasing code density. Specifying this option does not affect the treatment of autoaligned instructions such as LOOP, which the assembler will always align, either by widening density instructions or by inserting NOOP instructions.

`-mno-text-section-literals`: Specifying this option tells GCC to place literals in a separate section in the output file. This allows the literal pool to be placed in a data RAM/ROM, and also allows the linker to combine literal pools from separate object files to remove redundant literals and improve code size. This is the default.

`-mnsa`: Specifying this option tells GCC to enable the use of the optional NSA instructions to implement the built-in `ffs` function. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mserialize-volatile`: Specifying this option tells GCC to insert MEMW instructions before volatile memory references in order to guarantee sequential consistency. This option is only supported in GCC 3.x versions of the Xtensa compilers, where this option is the default.

`-msext`: Specifying this option tells GCC to enable the use of the optional SEXT instruction. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-msoft-float`: Specifying this option tells GCC to disable use of the floating-point option. GCC generates library calls to emulate 32-bit floating-point operations using integer instructions. 64-bit `double` operations are always emulated with calls to library functions. This option is only supported in GCC 3.x versions of the Xtensa compilers.

`-mtarget-align`: Specifying this option tells GCC to instruct the assembler to automatically align instructions in order to reduce branch penalties at the expense of some code density. The assembler attempts to widen density instructions to align branch targets and the instruction's following call instructions. If there are not enough preceding safe density instructions to align a target, no widening will be performed. This is the default.

`-mtext-section-literals`: Specifying this option tells GCC to intersperse literals in the text section in order to keep them as close as possible to their references. This may be necessary for large assembly files.

APPENDIX C

Using GCC's Online Help

For better or for worse, the FSF uses a nonstandard online help format, GNU Info, to document its software. GNU Info is one of those utilities people love to hate. Not only is GNU Info a more powerful and flexible system for documenting software than the ubiquitous Unix manual page, it is also dramatically different, which has kept it from becoming popular. Indeed, if you want to start a flame war on your favorite mailing list, don your asbestos underwear and post a message stating that Info is better than man. Seriously, because GNU Info is not very popular, most people do not now how to use it. Unfortunately, most of GCC's excellent documentation is maintained in GNU Info format; so to get the most out of GCC's own documentation, you have to know how to use GNU Info—this chapter teaches you how to do so. I think you will find that it is not nearly as difficult to use as it seems.

What Is GNU Info?

What is GNU Info? The simple answer is that GNU Info is a hypertext online help system intended by the FSF to replace the traditional (and, I dare say, the standard) Unix manual (or man) page. GNU Info enables online help systems to use tables of contents, cross-references, and indexes without requiring the overhead of an HTML browser. Info was originally designed long before the explosion of the Internet and before the World Wide Web became ubiquitous. Because of its hypertext abilities, GNU Info makes it easy to jump from one reference page to another that contains related information and then to return to the original page. This hyperlinking is a clear advantage over the traditional Unix manual page.

Unfortunately, the Info user interface, which bears a close resemblance to the Emacs user interface, is substantially different from and far more complicated than the interface used by traditional Unix manual page viewers, which are usually some variant of the standard text viewing programs, more or less. GNU's stubborn refusal to make traditional manual pages available for most of its software (with some notable exceptions, of course—the Bash manual pages come to mind) and Info's different interface has prevented it from becoming an accepted, popular replacement for the more limited but easier to use Unix manual page. So this short chapter will show you how to use GNU Info while staying out of the Info vs. man dispute.

Note I confess that I started writing this chapter from the perspective of a confirmed manual page advocate and committed Info hater. But my opinion of GNU Info has changed. Although I still wish that the GNU project would make both manual and Info pages available instead of forcing users to use Info, I have found that Info gives users greater capabilities for searching and viewing related information than the current man implementations. I still use the `man` command for quick reference information, but I no longer swear a blue streak if I have to use GNU Info.

GNU Info is more than just a hypertext online help system. It is really a complete document production system. The `info` command is a program that knows how to view and navigate Info files. Info files, in turn, are Texinfo (pronounced "teck-info") format text files that have been processed by the GNU `makeinfo` command in order to create Info files that the `info` command can read. Texinfo source files are plain ASCII text files containing so-called *@-commands*, which are characters and words prefixed with @, and a few other characters with special meanings. The @-commands and other special characters provide instructions to formatting and typesetting programs about how to format and typeset the text.

One of the attractions of Texinfo is that a single Texinfo source file can be used to produce Info files for use with GNU Info, typeset files ready for printing, and HTML output that you can browse with any Web browser. For the curious, Listing C-1 shows a small sample of Texinfo source code taken from the invoke.texi file. This short example of Texinfo source code describes the GCC keywords you can use when invoking the C compiler with the `-ansi` option. Listing C-2 shows the same Texinfo source code after it has been rendered into an Info file and displayed using the Info browser, `info`.

Listing C-1. *Sample Texinfo Source Code*

```
The alternate keywords @code{__asm__}, @code{__extension__},
@code{__inline__} and @code{__typeof__} continue to work despite
@option{-ansi}.  You would not want to use them in an ISO C program, of
course, but it is useful to put them in header files that might be included
in compilations done with @option{-ansi}.  Alternate predefined macros
such as @code{__unix__} and @code{__vax__} are also available, with or
without @option{-ansi}.

The @option{-ansi} option does not cause non-ISO programs to be
rejected gratuitously.  For that, @option{-pedantic} is required in
addition to @option{-ansi}.  @xref{Warning Options}.
```

Listing C-2. *Texinfo Source Code After Rendering*

```
The alternate keywords `__asm__', `__extension__', `__inline__'
and `__typeof__' continue to work despite `-ansi'.  You would not
want to use them in an ISO C program, of course, but it is useful
to put them in header files that might be included in compilations
done with `-ansi'.  Alternate predefined macros such as `__unix__'
and `__vax__' are also available, with or without `-ansi'.

The `-ansi' option does not cause non-ISO programs to be rejected
gratuitously.  For that, `-pedantic' is required in addition to
`-ansi'.  *Note Warning Options::.
```

I strongly recommend that you learn how to use the Info interface because a great deal of valuable information, especially about GCC, is only accessible in Info files. The next section, "Getting Started, or Instructions for the Impatient," should be sufficient to get you started. Subsequent sections go into detail and offer variations and extensions of the techniques discussed in the next section.

Getting Started, or Instructions for the Impatient

To start using GNU Info, type **info *program***, replacing *program* with the name of the program that interests you. This being a book about GCC, you might try

```
$ info gcc
```

To exit Info, type **q** to return to the command prompt.

If you are staring at an Info screen right now and just want to get started, Table C-1 lists the basic Info navigation commands you'll need to find your way around the Info help system.

Table C-1. *Basic Info Navigation Commands*

Key or Key Combination	Description
Ctrl-n	Moves the cursor down to the next line
Down	Moves the cursor down to the next line
Ctrl-p	Moves the cursor up to the previous line
Up	Moves the cursor up to the previous line
Ctrl-b	Moves the cursor one character to the left
Left	Moves the cursor one character to the left
Ctrl-f	Moves the cursor one character to the right
Right	Moves the cursor one character to the right
Ctrl-a	Moves the cursor to the beginning of the current line
Ctrl-e	Moves the cursor to the end of the current line
Spacebar	Scrolls forward (down) one screen or advances to the next screen if you are at the bottom of the current one
Ctrl-v	Scrolls forward (down) one screen
Page Down	Scrolls forward (down) one screen
Meta-v	Scrolls backward (up) one screen
Page Up	Scrolls backward (up) one screen
b	Moves the cursor to the beginning of the current node
e	Moves the cursor to the end of the current node
l	Moves to the most recently visited node
n	Moves to the next node
p	Moves to the previous node
u	Moves up a node
s	Performs a case-insensitive search for a string
S	Performs a case-sensitive search for a string
Ctrl-g	Cancels iterative operations such as an incremental search
Ctrl-x n	Repeats the last search
Ctrl-x N	Repeats the last search in the opposite direction

The notation Ctrl-x means to press and hold the Control key while pressing the x key. Meta-x means to press and hold the Meta key (also known as the Alt key) while pressing the x key. Finally, Ctrl-x n means to press and hold the Control key while pressing the x key, release them, and then press the n key.

■ **Tip** If the Alt key combinations described in this section do not work on your system, you can press the Esc key instead. So, for example, Meta-v (or Alt-v) can also be invoked as Esc-v. Notice that if you use the Esc key, you do not have to press and hold it as you do when using the Alt key.

The Emacs users in the audience will recognize the keystrokes in Table C-1 as Emacs key bindings. If you need more help than the keystroke reference in Table C-1 offers you, read the next few sections. If you get totally lost or confused, just type **q** to exit `info` and return to the command prompt.

Getting Help

Predictably, GNU Info has a help screen, accessible by pressing Ctrl-h. Use the Info commands discussed in this chapter to navigate the help screen. When you want to close the help window, press the l key—the help window is just a specially handled node pointer, so the l key returns you to the node you were originally viewing. If you are so inclined, you can go through the Info tutorial by pressing Ctrl-h within a screen. Remember the magic rescue sequence: if all else fails and you find yourself totally disoriented, press the q key, possibly several times, to exit Info.

The Beginner's Guide to Using GNU Info

This section explains the concepts and commands that cover 80 percent of GNU Info usage. By the time you have completed this section of the chapter, you will know more about GNU Info and how to use it more effectively than almost everyone else who uses GNU software. The other 20 percent of GNU Info commands, covered in the section titled "Stupid Info Tricks," explore advanced features and techniques whose use arguably constitute Info mastery. Okay, perhaps mastery is a bit over the top, but the "Stupid Info Tricks" section at the end of the chapter does describe sophisticated methods you will not use very often.

Anatomy of a GNU Info Screen

This section describes the standard components of an Info screen. You'll need to grasp this material because most of the discussion that follows assumes you know what a typical Info file looks like when displayed using the `info` command. Figure C-1 shows a typical Info screen (as luck would have it, it is the top-level node of the GCC Info file), with the various elements described in the text identified by callouts.

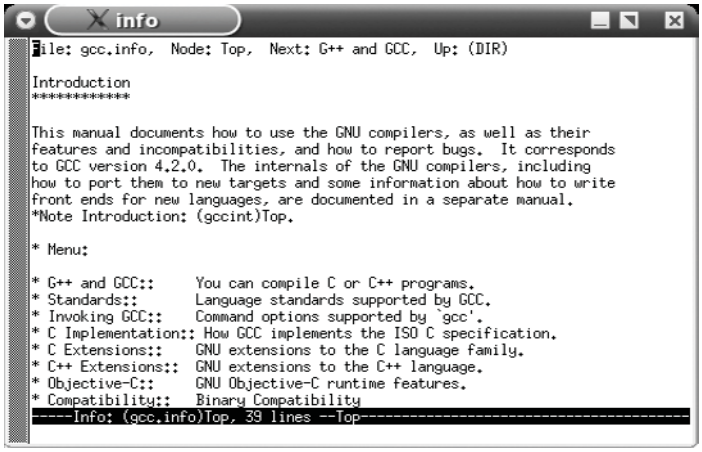

Figure C-1. *A typical GNU Info screen*

In general, GNU Info uses the term *window* to describe the screen area that displays Info text. Each such window has a mode line at the bottom that describes the node being displayed. The information displayed in the mode line includes the following information, reading left to right:

- The name of the file
- The name of the node
- The number of screen lines necessary to display the node
- The amount of text above the current screen, expressed as a percentage

So in Figure C-1, the name of the file is gcc.info.gz; the name of the node is Top; it takes 40 lines to display the entire screen; and the screen currently is displaying the top of the file. The text "zz" at the beginning of the mode line indicates that the file was read from a compressed disk file. The text beginning with "Subfile" means that this node has been split into multiple files and that, in this case, you are viewing the subfile gcc.info-1.gz. This text does not appear if the Info node has been spread across several files.

The documentation for GNU Info (available, oddly enough, as an Info file—type **info info** to view it) refers to the screen real estate occupied by actual Info text as the view area. However, I prefer to use the term *window* because the `info` command is capable of displaying multiple windows inside one screen and I do not want to have to refer to "the view area in the foo window" if "the foo window" is a clear enough reference. When the `info` command displays multiple windows in the same screen, one window is separated from another by its mode line, and the mode line always marks the bottom of a given window.

An echo area appears on the last or bottom line of each Info screen. The echo area, also called an *echo line*, displays status information and error messages, and serves as a buffer for typing the text used for word searches or for other input you might have to provide. When you are at the top or at the beginning of a node, a line of text across the top of the screen, which I call a *node pointer*, identifies the current file and node, and the next, previous, and parent nodes of the current node, when applicable. Node pointers and their usage are described in the next section, "Moving Around in GNU Info."

The topics listed in the center of the screen in Figure C-1 are known as *menu references*. If you move the cursor to any character between * and : : and press Enter, a process called *selecting a node*, you will jump to the Info file corresponding to that topic (commonly called a *node*). For example, Figure C-2 shows the screen that appears if you select the topic "C Extensions" and then, on that screen, select the topic "Labels as Values."

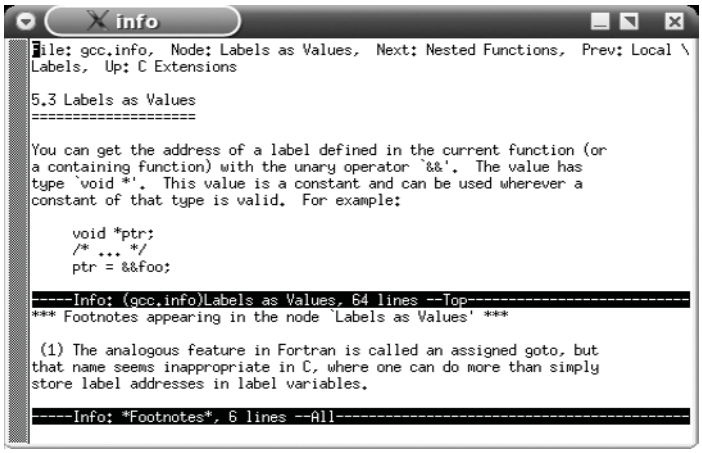

Figure C-2. *Following a GNU Info menu reference*

Notice that the screen shown in Figure C-2 has two windows, each separated from the other by a mode line as described earlier in this appendix. The upper window contains Info text, and the bottom window contains footnotes that appear in this node. If you press the spacebar, the text in the upper window scrolls forward one page or screen, but the footnote window remains static. You will learn later in this chapter how to navigate between windows. You might also notice that the mode line for the upper window indicates that you have jumped to the subfile gcc.info-11.gz.

Moving Around in GNU Info

You can get a lot done in GNU Info files just by pressing the spacebar (or Ctrl-v) to scroll through each screen in order. If you want to go backward, use Meta-v. If your keyboard has them, you can probably use the Next and Previous keys (also known as Page Down and Page Up on standard PC keyboards), instead of Ctrl-v and Meta-v. PC users will find Next and Previous more convenient and familiar. In all cases, though, the canonical sequences (Ctrl-v and Meta-v) should work. Similarly, the cursor keys allow you to scroll up and down through the screen you are currently looking at, one line of text at a time (in this case, the canonical keys are Ctrl-n for up and Ctrl-p for down), or to move left and right on the current line of text using Ctrl-b and Ctrl-f (left and right). Most PC users might find the cursor movement keys the most familiar; but the Control key sequences should always work regardless of the keyboard.

The node movement keys b, e, l, n, p, and u are easiest to understand if you are looking at the top of an Info page. For example, if you type **info gcc invoking** at a command prompt, the first few lines should resemble Listing C-3, the first page of the Invoking GCC node.

Listing C-3. *The First Page of the GCC Info File*

```
File: gcc.info,  Node: Invoking GCC,  Next: Installation,  Prev: G++ and GCC,  \
Up: Top

GCC Command Options
*******************
```

The line of text across the top shows the name of the file containing the node you are currently viewing (gcc.info), the name of the node itself (Node: Invoking GCC), the next node (Next: Installation), the previous node (Prev: G++ and GCC), and the top of the node tree (Up: Top). A node (more precisely, a node pointer) is an organizational unit in Info files that corresponds to chapters, sections, subsections, footnotes, or index entries in a printed book. It also makes it easier to navigate in an Info file. As Table C-1 shows, n jumps to the next node, p jumps to the previous node, and u jumps up a node or to the parent node. So, from the screen illustrated in Listing C-3, typing n jumps to the Installation node, p jumps to the node title G++ and GCC, and u takes you to the Top node, which, in this case, is the main menu or table of contents for the GCC Info file. To put it another way, the Top node is the parent of the current node, which may or may not always be the main menu. However, it usually only takes a few presses of the u key to get to the top of the tree.

Within a node, typing **b** places the cursor at the top of the current node and typing **e** places you at the end of the current node. I think the l (last node) key performs one of the most useful jumps, because it takes you back to the most recently visited node (the last node). The l key is comparable to the Back button in a Web browser. Figure C-3 illustrates how a complete Info topic might be organized.

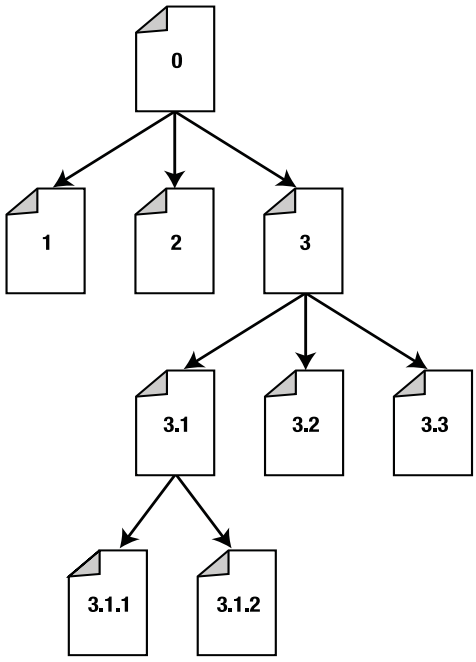

Figure C-3. *The node structure of a GNU Info file*

There are three Top nodes in Figure C-3. The Top node for 3.1.1 and 3.1.2 is 3.1. The Top node for 3.1, 3.2, and 3.3 is 3, and 3's Top node, like 1 and 2, is 0. So, if you were viewing 3.1.2, you would type **u** three times to return to the top or root node. If Figure C-3 were arranged as a table of contents, it would resemble the following:

```
Chapter 1
Chapter 2
Chapter 3
    Section 3.1
        Subsection 3.1.1
        Subsection 3.1.2
    Section 3.2
    Section 3.3
```

Finally, if you walked through the nodes using the n key or the spacebar, the progression would be 0 -> 1 -> 2 -> 3 -> 3.1 -> 3.1.1 -> 3.1.2 -> 3.2 -> 3.3.

I have made such an effort to explain and illustrate the node structure of a GNU Info file and how to navigate through it because most users unfamiliar with Info complain about getting lost in it and not knowing where they are, how to get back to where they were, and where they want to be. It can be boiled down to two simple rules:

1. Learn the mnemonics of the b, e, n, p, and u keys: (b)eginning of node, (e)nd of node, (n)ext node, (p)revious node, and (u)p to the Top node.
2. Keep your eye on the node pointers at the top of a node so you will know where the n, p, and u keys will take you.

Performing Searches in GNU Info

One of the most useful features of GNU Info is its ability to perform fast searches and a large variety of searches. The stand-alone `info` command can search forward and backward searches and searches that are case sensitive or insensitive. You can search incrementally or by complete string, and you can also search an Info file's indices and then jump to the node containing the matching index entry. Although man page viewers have similar functionality, I have found GNU Info's search features to be faster and more capable. As always, your mileage may vary. Before jumping into the details of searches, Table C-2 lists the search functions you need to know.

Table C-2. *GNU Info Search Commands*

Key or Key Combination	Description
?	Searches backward for a string typed in the echo area, ignoring case.
,	Jumps to the next node containing a match for a previously completed index search.
i	Searches the file index for a string typed in the echo area and jumps to the node containing a match.
s	Searches for a string typed in the echo area, ignoring case.
S	Searches for a string typed in the echo area, paying attention to case.
Ctrl-g	Aborts the current search.
Ctrl-r	Searches backward for a string as you type it.

Table C-2. *GNU Info Search Commands*

Key or Key Combination	Description
Ctrl-s	Searches forward for a string as you type it.
Ctrl-x n	Repeats the previous search, retaining the previous search's case sensitivity.
Ctrl-x N	Repeats the previous search in the reverse direction, retaining the previous search's case sensitivity.

While performing a forward search with Ctrl-s, you can press Ctrl-s to search forward for the next occurrence, or Ctrl-r to search backward for the previous occurrence. Conversely, when running a backward search using Ctrl-r, pressing Ctrl-r repeats the backward search for the next (previous) occurrence, and Ctrl-s executes the same search in the forward direction.

Another way to repeat the previous search is to type **s** (or **S**) and press Enter to accept the default string, which is always the last search string embedded in square brackets ([]). Pressing Enter without typing a new search string uses the default value. When repeating previous searches using Ctrl-x n or Ctrl-x N, the previous search's case sensitivity option is applied. The search features Info provides do show one small bit of perversity: the s search is case insensitive unless the string entered contains an uppercase letter, in which case the search automatically becomes case sensitive.

■ **Tip** In almost every mode, pressing Ctrl-g, sometimes several times, will cancel most operations and restore Info's interface to something sane.

Following Cross-References

Another one of GNU Info's most useful features, and the source of much gnashing of teeth for the uninitiated, is its support for cross-references, referred to as *xrefs* in the GNU Info documentation. xrefs are pointers or hotlinks to other Info nodes that contain related information (and that may exist in a separate file). It is these xrefs that give Info its hypertext capability. It is also these xrefs that cause users to get lost and start hating Info. And it is when using these xrefs that the l key becomes most useful—getting you back to where you started. A canonical xref has the following form:

```
* label: target. comment
```

An xref begins with *, followed by *label*, which is a name that you can use to refer to the xref; this is followed by a colon, which separates the label from the target; the target itself; a terminating period to end the target name; and an optional *comment*. *target* is the full name of the node to which the xref refers. The optional comment is often used to describe the contents of the referenced node. Consider the following xref taken from the GCC Info file's index:

```
* allocate_stack instruction pattern:    Standard Names.
```

In this example, *label* is the text "allocate_stack instruction pattern" and *target* is the node named *Standard Names*. The period (.) is not part of the target, but merely indicates the end of the target name. To select an xref, position the cursor anywhere between the * and the end of the target name and then press Enter.

You will often see xrefs that use the following form:

```
* G++ and GCC::     You can compile C or C++ programs.
```

In this case, the target name has been omitted and a second colon used in its place. This means that the label and the target name have the same names. That is, the previous xref is equivalent to

```
* G++ and GCC:G++ and GCC.     You can compile C or C++ programs.
```

This shorthand notation is especially common in node menus, such as the one shown in Figure C-1. In fact, this shorthand notation is so commonly used in node menus that it is called a *menu reference* in order to distinguish from the other standard type of xref, a *note reference*. Unlike menu references, which appear in, you guessed it, node menus, note references appear in the body text of a node, inline with the rest of the text, much like the hyperlinks in the body text of a standard HTML Web page. Note references are very similar to menu references, except that they begin with *Note, rather than just a bare *. Listing C-4, for example, extracted from the GCC Info file, contains a note reference.

Listing C-4. *A Typical Note Reference*

```
element of the union should be used.  For example,

     union foo { int i; double d; };

     union foo f = { d: 4 };

will convert 4 to a 'double' to store it in the union using the second
element.  By contrast, casting 4 to type 'union foo' would store it
into the union as the integer 'i', since it is an integer.  (*Note Cast
to Union::.)
```

The note reference appears at the end of the listing and refers to the node named *Cast to Union*, which explains the semantics of casting a type to a C union type. After selecting either type of xref, you can use the l key to return to the node you were reading before you jumped, or you can follow another reference. No matter how far afield you stray, if you press l enough times, you will eventually traverse the history list the `info` command maintains until you return to the node from which you started.

Printing GNU Info Nodes

GNU Info's documented method for printing an individual node is the command `M-x print-node`, which pipes the contents of the node to the `lpr` command. If the environment variable `INFO_PRINT_COMMAND` is set, the node's contents will be piped through the command defined in this variable. However, if you want to print a complete Info file rather than a single node, you might want to use the following command:

```
$ info --subnodes info_file -o - | lpr
```

This command prints the entire Info file named info_file using the default system printer. The `--subnodes` option tells the `info` command to recurse through each node of info_file's menus and display the contents of each node as it recurses through the node tree. The `-o` option tells the `info` command to save the output in a file and must be followed by a filename. A filename of - specifies standard output. In the command shown, output is piped into `lpr`, which, if your printer is properly configured, will print the entire manual. Thus, to print the entire Info file for the `info` command, the command would be

```
$ info --subnodes info -o - | lpr
```

■ **Caution** You may not want to actually print this, since the entire Info manual is 24 pages long.

Invoking GNU Info

The complete syntax for executing GNU Info from the command line is

`info [`*`options`*`...] [`*`info_file`*`...]`

info_file identifies the Info node(s) you want to view, separated by whitespace. For example, `info gcc` opens the GCC Info file. If you want to visit a particular menu item or subnode available from the menu (table of contents) in the GCC Info file, specify it after the top-level node. So if you want to go straight to the Installation node of the GCC Info file, use the command `info gcc installation`. Additional arguments will take you deeper into the node tree. Thus, `info gcc installation cross-compiler` will take you to the Cross-Compiler subnode of the Installation subnode of the gcc node. The case of the specified nodes and subnodes is ignored. Table C-3 lists the values for the options arguments that I find most useful—the complete list is available in the Invoking Info node of the `info` command's Info file (info info invoking).

Table C-3. *Common GNU Info Command–Line Options*

Option	Description
`--apropos=`*`word`*	Starts `info` looking for *`word`*.
`-d` *`dir`*	Adds the directory *`dir`* to the `info` program's search path.
`--directory=`*`dir`*	Adds the directory *`dir`* to the `info` program's search path.
`--dribble=`*`file`*	Logs keystrokes to *`file`* (see `--restore`).
`-f` *`file`*	Opens the Info file specified by *`file`*, bypassing the special directory file and the default search path.
`--file=`*`file`*	Opens the Info file specified by *`file`*, bypassing the special directory file and the default search path.
`-h`	Displays a short usage message.
`--help`	Displays a short usage message.
`--index-search=`*`string`*	Searches the specified Info file's index node for *`string`* and jumps to the first matching node.
`--location`	Displays the full path name of the Info file that would be used, and exits.
`-n` *`node`*	Goes directly to the specified node in the Info file, if it exists.
`--node=`*`node`*	Goes directly to the specified node in the Info file, if it exists.
`-O`	Jumps directly to an Info node that identifies how to invoke a program, if such a node exists.
`-o` *`file`*	Redirects Info output from the display to the file specified by *file*.
`--output` *`file`*	Redirects Info output from the display to the file specified by *file*.

Table C-3. *Common GNU Info Command–Line Options (Continued)*

Option	Description
`-R`	Leaves ANSI formatting on the page
`--raw-escapes`	Leaves ANSI formatting on the page.
`--restore=`*`file`*	Reads keystrokes from *`file`* (complements `--dribble`).
`--show-options`	Same as `-O` and `--usage`.
`--subnodes`	Recursively displays the child nodes of the top-level Info file (must be used with the `-o` option to specify an output file).
`--usage`	Jumps directly to an Info node that identifies how to invoke a program, if such a node exists.
`--version`	Displays `info` version information and exits.
`--vi-keys`	Starts `info` using key bindings that work somewhat like `vi`.
`-w`	Displays the full pathname of the Info file that would be used, and exits.
`--where`	Displays the full pathname of the Info file that would be used, and exits.

If `-f` *`file`* specifies an absolute pathname, such as `-f ./gcc.info`, Info will only look at the specified file or path, ignoring the built-in search path. If a search specified by `--index-search` fails, Info displays an error message. As explained earlier, a bare `-` specified with `-o` is interpreted as standard output, which allows you to use Info output in pipelines and with the `--subnodes` option.

Stupid Info Tricks

Despite the title, this section offers a few clever tips and hints for using GNU Info. Learning to use the information in this section might turn you into a certified GNU Info power user, allowing you to dazzle your friends and colleagues with your mastery of GNU Info.

One of my favorite tricks uses the `--subnodes` option. As you can probably imagine, I spent *a lot* of time studying GCC's Info pages. When I got tired of looking at onscreen text, I used the following shell pipeline, or a variation of it, to print out all or part of the GCC manual.

```
$ info --subnodes gcc -o - | enscript -G -H -o - | psnup -2 | lpr
```

The first part of the command you have already seen. Instead of piping the output straight to the printer, I ran it through enscript to convert the text to PostScript, piped enscript's output to psnup to create two-up pages (to reduce paper consumption), and then printed the result. Enscript's `-G` option adds a flashy header to the top of each (virtual) page; `-H` adds highlight bars to the printed output, emulating what old computer hacks know as *greenbar* (although, in this case it is probably more accurate to call it *graybar*).

Using Command Multipliers

GNU Info, like GNU Emacs, enables you to prefix most commands with a numeric multiplier, causing the commands to execute multiple times. The command that invokes the multiplier is Ctrl-u [*n*], where *n* is the number of times you want the following command to execute. If you omit *n*, the default

value of the multiplier is 4. So, for example, to scroll down four screens, you can press Ctrl-u 4 Ctrl-v, or Ctrl-u Ctrl-v. To scroll backward two screens, you can try Ctrl-u 2 Meta-v, or Ctrl-u -2 Ctrl-v. Yes, that is -2 in the second command—when you use a negative multiplier value, most operations, such as moving or scrolling, work in the opposite or reverse direction. Thus, for example, using a negative multiplier with Ctrl-b causes the cursor to move forward rather than backward.

Working with Multiple Windows

I mentioned earlier in the chapter that GNU Info can display multiple windows in a single screen, with mode lines serving as the visual separators between windows. Info also provides methods for moving from one window to another, changing the size of windows, and for creating and deleting windows. The unspoken question, of course, is "Why would one want to use multiple windows?" The most common reason is to keep from having to jump back and forth between nodes. Instead, you can open an additional window (strictly speaking, you just split the existing window into two separate windows), display a different node in the new window, and then switch between them. This approach allows you to look at the information in two nodes more or less simultaneously. Obviously, it should be clear that you can open as many windows as you want, although too many windows open in the same screen will quickly become unwieldy and difficult to read.

Table C-4 shows common GNU Info commands related to opening, closing, moving between, and resizing Info windows.

Table C-4. *Common Info Commands*

Action	Command	Explanation
Close current window	Ctrl-x 0	Closes the current window and moves the cursor to the next available window.
Close other windows	Ctrl-x 1	Closes all windows other than the one in which the cursor is currently located.
Increase size of current window	Ctrl-x ^	Incrementally increases the size of the current window by one line. Can be used with the Ctrl-u command multiplier to increase window size by multiple lines, as in Ctrl-u 3 Ctrl-x ^, which would increase the size of the current window by three lines.
Move to other window	Ctrl-x o	Moves the cursor to the next, or other, window. If you have more than two windows open, pressing Ctrl-x o repeatedly cycles the cursor through all open windows.
Move to previous window	Esc x o	Moves the cursor to the previous window.
Scroll other window	Esc Ctrl-v	Scrolls the previous/other window forward one screen of text.
Split the current window	Ctrl-x 2	Splits the current window, defined as the window containing the cursor, into two equal-size windows, leaving the cursor in the original window. Info commands, such as cursor movement, scrolling, and following xrefs, only apply to the current window.

Index

B

C

D

E

F

G

H

J

K

L

M

N

O

P

R

S

T

U

V

X