

NAME

addr2line – convert addresses into file names and line numbers

SYNOPSIS**addr2line**

[**-b** *bfdname* | **--target=bfdname**] [**-C** | **--demangle**] [**-e** *filename* | **--exe=filename**]
 [**-f** | **--functions**] [**-s** | **--basenames**] [**-H** | **--help**] [**-V** | **--version**] [**addraddr...**]

DESCRIPTION

addr2line translates program addresses into file names and line numbers. Given an address and an executable, it uses the debugging information in the executable to figure out which file name and line number are associated with a given address.

The executable to use is specified with the **-e** option. The default is **a.out**.

addr2line has two modes of operation.

In the first, hexadecimal addresses are specified on the command line, and **addr2line** displays the file name and line number for each address.

In the second, **addr2line** reads hexadecimal addresses from standard input, and prints the file name and line number for each address on standard output. In this mode, **addr2line** may be used in a pipe to convert dynamically chosen addresses.

The format of the output is FILENAME:LINENO. The file name and line number for each address is printed on a separate line. If the **-f** option is used, then each FILENAME:LINENO line is preceded by a FUNCTIONNAME line which is the name of the function containing the address.

If the file name or function name can not be determined, **addr2line** will print two question marks in their place. If the line number can not be determined, **addr2line** will print 0.

OPTIONS

-b *bfdname*

--target=bfdname

Specify the object-code format for the object files to be *bfdname*.

-C

--demangle

Decode (*demangle*) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable.

-e *filename*

--exe=filename

Specify the name of the executable for which addresses should be translated. The default file is **a.out**.

-f

--functions

Display function names as well as file and line number information.

-s

--basenames

Display only the base of each file name.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991).

COPYING

Copyright (c) 1993, 94, 95, 96, 97, 98, 1999, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

ar – create, modify, and extract from archives.

SYNOPSIS

ar [-]{*dmpqrtx*}[*abcfilNoPsSuvV*] [*membername*] [*count*] *archive files*...

DESCRIPTION

The GNU **ar** program creates, modifies, and extracts from archives. An *archive* is a single file holding a collection of other files in a structure that makes it possible to retrieve the original individual files (called *members* of the archive).

The original files' contents, mode (permissions), timestamp, owner, and group are preserved in the archive, and may be reconstituted on extraction.

GNU **ar** can maintain archives whose members have names of any length; however, depending on how **ar** is configured on your system, a limit on member-name length may be imposed (for compatibility with archive formats maintained with other tools). If it exists, the limit is often 15 characters (typical of formats related to a.out) or 16 characters (typical of formats related to coff).

ar is considered a binary utility because archives of this sort are most often used as *libraries* holding commonly needed subroutines.

ar will create an index to the symbols defined in relocatable object modules in the archive when you specify the modifier '**s**'. Once created, this index is updated in the archive whenever **ar** makes a change to its contents (save for the '**q**' update operation). An archive with such an index speeds up linking to the library, and allows routines in the library to call each other without regard to their placement in the archive.

You may use '**nm -s**' or '**nm --print-armap**' to list this index table. If an archive lacks the table, another form of **ar** called **ranlib** can be used to add just the table.

ar insists on at least two arguments to execute: one keyletter specifying the *operation* (optionally accompanied by other keyletters specifying *modifiers*), and the archive name to act on.

Most operations can also accept further *files* arguments, specifying particular files to operate on.

OPTIONS

GNU **ar** allows you to mix the operation code *p* and modifier flags *mod* in any order, within the first command-line argument.

If you wish, you may begin the first command-line argument with a dash.

The *p* keyletter specifies what operation to execute; it may be any of the following, but you must specify only one of them:

d *Delete* modules from the archive. Specify the names of modules to be deleted as *files*; the archive is untouched if you specify no files to delete.

If you specify the '**v**' modifier, **ar** will list each module as it is deleted.

m Use this operation to *move* members in an archive.

The ordering of members in an archive can make a difference in how programs are linked using

the library, if a symbol is defined in more than one member.

If no modifiers are used with **m**, any members you name in the *files* arguments are moved to the *end* of the archive; you can use the ‘**a**’, ‘**b**’, or ‘**i**’ modifiers to move them to a specified place instead.

- p** *Print* the specified members of the archive, to the standard output file. If the ‘**v**’ modifier is specified, show the member name before copying its contents to standard output.

If you specify no *files*, all the files in the archive are printed.

- q** *Quick append*; add *files* to the end of *archive*, without checking for replacement.

The modifiers ‘**a**’, ‘**b**’, and ‘**i**’ do *not* affect this operation; new members are always placed at the end of the archive.

The modifier ‘**v**’ makes **ar** list each file as it is appended.

Since the point of this operation is speed, the archive’s symbol table index is not updated, even if it already existed; you can use ‘**ar s**’ or **ranlib** explicitly to update the symbol table index.

However, too many different systems assume quick append rebuilds the index, so GNU **ar** implements ‘**q**’ as a synonym for ‘**r**’.

- r** Insert *files* into *archive* (with *replacement*). This operation differs from ‘**q**’ in that any previously existing members are deleted if their names match those being added.

If one of the files named in *files* doesn’t exist, **ar** displays an error message, and leaves undisturbed any existing members of the archive matching that name.

By default, new members are added at the end of the file; but you may use one of the modifiers ‘**a**’, ‘**b**’, or ‘**i**’ to request placement relative to some existing member.

The modifier ‘**v**’ used with this operation elicits a line of output for each file inserted, along with one of the letters ‘**a**’ or ‘**r**’ to indicate whether the file was appended (no old member deleted) or replaced.

- t** Display a *table* listing the contents of *archive*, or those of the files listed in *files* that are present in the archive. Normally only the member name is shown; if you also want to see the modes (permissions), timestamp, owner, group, and size, you can request that by also specifying the ‘**v**’ modifier.

If you do not specify any *files*, all files in the archive are listed.

If there is more than one file with the same name (say, ‘**fie**’) in an archive (say ‘**b.a**’), ‘**ar t b.a fie**’ will list only the first instance; to see them all, you must ask for a complete listing—in our example, ‘**ar t b.a**’.

- x** *Extract* members (named *files*) from the archive. You can use the ‘**v**’ modifier with this operation, to request that **ar** list each name as it extracts it.

If you do not specify any *files*, all files in the archive are extracted.

A number of modifiers (*mod*) may immediately follow the *p* keyletter, to specify variations on an operation's behavior:

- a** Add new files *after* an existing member of the archive. If you use the modifier **a**, the name of an existing archive member must be present as the *membername* argument, before the *archive* specification.
- b** Add new files *before* an existing member of the archive. If you use the modifier **b**, the name of an existing archive member must be present as the *membername* argument, before the *archive* specification. (same as '**i**').
- c** *Create* the archive. The specified *archive* is always created if it didn't exist, when you request an update. But a warning is issued unless you specify in advance that you expect to create it, by using this modifier.
- f** Truncate names in the archive. **ar** will normally permit file names of any length. This will cause it to create archives which are not compatible with the native **ar** program on some systems. If this is a concern, the **f** modifier may be used to truncate file names when putting them in the archive.
- i** Insert new files *before* an existing member of the archive. If you use the modifier **i**, the name of an existing archive member must be present as the *membername* argument, before the *archive* specification. (same as '**b**').
- I** This modifier is accepted but not used.
- N** Uses the *count* parameter. This is used if there are multiple entries in the archive with the same name. Extract or delete instance *count* of the given name from the archive.
- o** Preserve the *original* dates of members when extracting them. If you do not specify this modifier, files extracted from the archive will be stamped with the time of extraction.
- P** Use the full path name when matching names in the archive. **ar** can not create an archive with a full path name (such archives are not POSIX complaint), but other archive creators can. This option will cause **ar** to match file names using a complete path name, which can be convenient when extracting a single file from an archive created by another tool.
- s** Write an object-file index into the archive, or update an existing one, even if no other change is made to the archive. You may use this modifier flag either with any operation, or alone. Running '**ar s**' on an archive is equivalent to running '**ranlib**' on it.
- S** Do not generate an archive symbol table. This can speed up building a large library in several steps. The resulting archive can not be used with the linker. In order to build a symbol table, you must omit the '**S**' modifier on the last execution of '**ar**', or you must run '**ranlib**' on the archive.
- u** Normally, **ar r...** inserts all files listed into the archive. If you would like to insert *only* those of the files you list that are newer than existing members of the same names, use this modifier. The '**u**' modifier is allowed only for the operation '**r**' (replace). In particular, the combination '**qu**' is not allowed, since checking the timestamps would lose any speed advantage from the operation '**q**'.

- v** This modifier requests the *verbose* version of an operation. Many operations display additional information, such as filenames processed, when the modifier ‘**v**’ is appended.
- V** This modifier shows the version number of **ar**.

SEE ALSO

‘**binutils**’ entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991). **nm(1)**, **ranlib(1)**.

COPYING

Copyright (c) 1991, 1992, 1993, 1995, 1998, 1999, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

GNU **as** – the portable GNU assembler.

SYNOPSIS

```
as [ -a[ dhlns ] [ =file ] ] [ -D ] [ --defsym SYM=VAL ] [ -f ] [ --gstabs ] [ -I path ] [ -K ] [ -L ]
[ -M | --mri ] [ -o objfile ] [ -R ] [ --traditional-format ] [ -v ] [ -w ] [ -- | files ... ]
```

i960-only options:

```
[ -ACA | -ACA_A | -ACB | -ACC | -AKA | -AKB | -AKC | -AMC ] [ -b ] [ -no-relax ]
```

m680x0-only options:

```
[ -l ] [ -mc68000 | -mc68010 | -mc68020 ]
```

DESCRIPTION

GNU **as** is really a family of assemblers. If you use (or have used) the GNU assembler on one architecture, you should find a fairly similar environment when you use it on another architecture. Each version has much in common with the others, including object file formats, most assembler directives (often called *pseudo-ops*) and assembler syntax.

For information on the syntax and pseudo-ops used by GNU **as**, see ‘**as**’ entry in **info** (or the manual *Using as: The GNU Assembler*).

as is primarily intended to assemble the output of the GNU C compiler **gcc** for use by the linker **ld**. Nevertheless, we’ve tried to make **as** assemble correctly everything that the native assembler would. This doesn’t mean **as** always uses the same syntax as another assembler for the same architecture; for example, we know of several incompatible versions of 680x0 assembly language syntax.

Each time you run **as** it assembles exactly one source program. The source program is made up of one or more files. (The standard input is also a file.)

If **as** is given no file names it attempts to read one input file from the **as** standard input, which is normally your terminal. You may have to type **ctl-D** to tell **as** there is no more program to assemble. Use ‘**--**’ if you need to explicitly name the standard input file in your command line.

as may write warnings and error messages to the standard error file (usually your terminal). This should not happen when **as** is run automatically by a compiler. Warnings report an assumption made so that **as** could keep assembling a flawed program; errors report a grave problem that stops the assembly.

OPTIONS

-a Turn on assembly listings. There are various suboptions. **d** omits debugging directives. **h** includes the high level source code; this is only available if the source file can be found, and the code was compiled with **-g**. **l** includes an assembly listing. **n** omits forms processing. **s** includes a symbol listing. **=file** sets the listing file name; this must be the last suboption. The default suboptions are **hls**.

-D This option is accepted only for script compatibility with calls to other assemblers; it has no effect on **as**.

--defsym **SYM=VALUE**

Define the symbol **SYM** to be **VALUE** before assembling the input file. **VALUE** must be an integer constant. As in C, a leading 0x indicates a hexadecimal value, and a leading 0 indicates an octal value.

-f “fast” --skip preprocessing (assume source is compiler output).

- I *path***
Add *path* to the search list for **.include** directives.
- gstabs**
Generate stabs debugging information for each assembler line. This may help debugging assembler code, if the debugger can handle it.
- K** Issue warnings when difference tables altered for long displacements.
- L** Keep (in symbol table) local symbols, starting with '**L**'
- M, --mri**
Assemble in MRI compatibility mode.
- o *objfile***
Name the object-file output from **as**
- R** Fold data section into text section
- traditional-format**
Use same format as native assembler, when possible.
- v** Announce **as** version
- W, --no-warn**
Suppress warning messages.
- fatal-warnings**
Consider warnings to be fatal.
- warn**
Just warn on warnings.
- | *files...***
Source files to assemble, or standard input (**--**)
- Avar** (*When configured for Intel 960.*) Specify which variant of the 960 architecture is the target.
- b** (*When configured for Intel 960.*) Add code to collect statistics about branches taken.
- no-relax**
(*When configured for Intel 960.*) Do not alter compare-and-branch instructions for long displacements; error if necessary.
- l** (*When configured for Motorola 68000.*)
Shorten references to undefined symbols, to one word instead of two.
- mc68000 | -mc68010 | -mc68020**
(*When configured for Motorola 68000.*)
Specify what processor in the 68000 family is the target (default 68020)

Options may be in any order, and may be before, after, or between file names. The order of file names is significant.

'**--**' (two hyphens) by itself names the standard input file explicitly, as one of the files for **as** to assemble.

Except for '**--**' any command line argument that begins with a hyphen ('-') is an option. Each option changes the behavior of **as**. No option changes the way another option works. An option is a '-' followed by one or more letters; the case of the letter is important. All options are optional.

The '**-o**' option expects exactly one file name to follow. The file name may either immediately follow the option's letter (compatible with older assemblers) or it may be the next command argument (GNU standard).

These two command lines are equivalent:

as -o my-object-file.o mumble.s

as -omy-object-file.o mumble.s

SEE ALSO

‘**as**’ entry in **info**; *Using as: The GNU Assembler*; **gcc(1)**, **ld(1)**.

COPYING

Copyright (c) 1991, 1992, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

c++filt • demangle C++ symbols

SYNOPSIS

```
c++filt [ •_ | •strip-underscores ] [ •s {gnu,lucid,arm} | •format={gnu,lucid,arm} ] [ •help ]
[ •version ] [ symbol... ]
```

DESCRIPTION

The C++ language provides function overloading, which means that you can write many functions with the same name (providing each takes parameters of different types). All C++ function names are encoded into a low-level assembly label (this process is known as *mangling*). The **c++filt** program does the inverse mapping: it decodes (*demangles*) low-level names into user-level names so that the linker can keep these overloaded functions from clashing.

Every alphanumeric word (consisting of letters, digits, underscores, dollars, or periods) seen in the input is a potential label. If the label decodes into a C++ name, the C++ name replaces the low-level name in the output.

You can use **c++filt** to decipher individual symbols by specifying these symbols on the command line.

If no **symbol** arguments are given, **c++filt** reads symbol names from the standard input and writes the demangled names to the standard output. All results are printed on the standard output.

OPTIONS

•_

•strip-underscores

On some systems, both the C and C++ compilers put an underscore in front of every name. For example, the C name **foo** gets the low-level name **_foo**. This option removes the leading underscore.

•s {gnu,lucid,arm}**•format={gnu,lucid,arm}**

GNU **nm** can decode three different methods of mangling, used by different C++ compilers. This option selects which method it uses: the one used by the GNU compiler, the one used by the Lucid compiler, or the one specified by the C++ Annotated Reference Manual. The default is the GNU style.

•help Print a summary of the options to **c++filt** and exit.

•version

Print the version number of **c++filt** and exit.

SEE ALSO

binutils 5 entry **info**; *The GNU Binary Utilities*, Roland H. Pesch (June 1993).

COPYING

Copyright (c) 1993 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the

above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

NAME

g77 – GNU project Fortran Compiler (v0.5.24)

SYNOPSIS

g77 [*option* | *filename*] ...

WARNING

The information in this man page is an extract from the full documentation of the GNU Fortran compiler (version 0.5.24), and is limited to the meaning of some of the options.

This man page is not up to date, since no volunteers want to maintain it. If you find a discrepancy between the man page and the software, please check the Info file, which is the authoritative documentation.

If we find that the things in this man page that are out of date cause significant confusion or complaints, we will stop distributing the man page. The alternative, updating the man page when we update the Info file, is impractical because the rest of the work of maintaining GNU Fortran leaves us no time for that. The GNU project regards man pages as obsolete and should not let them take time away from other things.

For complete and current documentation, refer to the Info file '**g77**' or the manual *Using and Porting GNU Fortran* (for version 0.5.24). Both are made from the Texinfo source file **g77.texi**.

If your system has the '**info**' command installed, the command '**info g77**' should work, unless **g77** has not been properly installed. If your system lacks '**info**', or you wish to avoid using it for now, the command '**more /usr/info/g77.info***' should work, unless **g77** has not been properly installed.

If **g77** has not been properly installed, so that you cannot easily access the Info file for it, ask your system administrator, or the installer of **g77** (if you know who that is) to fix the problem.

DESCRIPTION

The C and F77 compilers are integrated; **g77** is a program to call **gcc** with options to recognize programs written in Fortran (ANSI FORTRAN 77, also called F77). **gcc** processes input files through one or more of four stages: preprocessing, compilation, assembly, and linking. This man page contains full descriptions for *only* F77-specific aspects of the compiler, though it also contains summaries of some general-purpose options. For a fuller explanation of the compiler, see **gcc(1)**.

For complete documentation on GNU Fortran, type '**info g77**'.

F77 source files use the suffix '**.f**', '**.for**', or '**.FOR**'; F77 files to be preprocessed by **cpp(1)** use the suffix '**.F**', '**.fpp**', or '**.FPP**'; Ratfor source files use the suffix '**.r**' (though **ratfor** itself is not supplied as part of **g77**).

OPTIONS

There are many command-line options, including options to control details of optimization, warnings, and code generation, which are common to both **gcc** and **g77**. For full information on all options, see **gcc(1)**.

Options must be separate: '**-dr**' is quite different from '**-d -r**'.

Most '**-f**' and '**-W**' options have two contrary forms: **-fname** and **-fno-name** (or **-Wname** and **-Wno-name**). Only the non-default forms are shown here.

-c Compile or assemble the source files, but do not link. The compiler output is an object file corresponding to each source file.

-Dmacro

Define macro *macro* with the string '**1**' as its definition.

-Dmacro=defn

Define macro *macro* as *defn*.

-E Stop after the preprocessing stage; do not run the compiler proper. The output is preprocessed source code, which is sent to the standard output.

- g** Produce debugging information in the operating system's native format (for DBX or SDB or DWARF). GDB also can work with this debugging information. On most systems that use DBX format, '**-g**' enables use of extra debugging information that only GDB can use.

Unlike most other Fortran compilers, GNU Fortran allows you to use '**-g**' with '**-O**'. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

- I***dir* Append directory *dir* to the list of directories searched for include files.
- L***dir* Add directory *dir* to the list of directories to be searched for '**-l**'.
- l***library* Use the library named *library* when linking.
- nostdinc** Do not search the standard system directories for header files. Only the directories you have specified with **-I** options (and the current directory, if appropriate) are searched.
- O** Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. See the GCC documentation for further optimisation options. Loop unrolling, in particular, may be worth investigating for typical numerical Fortran programs.
- o** *file* Place output in file *file*.
- S** Stop after the stage of compilation proper; do not assemble. The output is an assembler code file for each non-assembler input file specified.
- U***macro* Undefine macro *macro*.
- v** Print (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper. The version numbers of g77 itself and the GCC distribution on which it is based are distinct.
- Wall** Issue warnings for conditions which pertain to usage that we recommend avoiding and that we believe is easy to avoid, even in conjunction with macros.

FILES

file.h C header (preprocessor) file
 file.f Fortran source file
 file.for Fortran source file
 file.FOR Fortran source file
 file.F preprocessed Fortran source file
 file.fpp preprocessed Fortran source file
 file.FPP preprocessed Fortran source file
 file.r Ratfor source file (ratfor not included)
 file.s assembly language file
 file.o object file
 a.out link edited output
 TMPDIR/cc* temporary files
 LIBDIR/cpp preprocessor
 LIBDIR/f771 compiler
 LIBDIR/libg2c.a Fortran run-time library

<i>LIBDIR</i> /libgcc.a	GCC subroutine library
/lib/crt[01n].o	start-up routine
/lib/libc.a	standard C library, see <i>intro</i> (3)
/usr/include	standard directory for #include files
<i>LIBDIR</i> /include	standard gcc directory for #include files.

LIBDIR is usually **/usr/local/lib/machine/version**.

TMPDIR comes from the environment variable **TMPDIR** (default **/usr/tmp** if available, else **/tmp**).

SEE ALSO

gcc(1), cpp(1), as(1), ld(1), gdb(1), adb(1), dbx(1), sdb(1).
'g77', **'gcc'**, **'cpp'**, **'as'**, **'ld'**, and **'gdb'** entries in **info**.

Using and Porting GNU Fortran (for version 0.5.24), James Craig Burley; *Using and Porting GNU CC (for version 2.0)*, Richard M. Stallman; *The C Preprocessor*, Richard M. Stallman; *Debugging with GDB: the GNU Source-Level Debugger*, Richard M. Stallman and Roland H. Pesch; *Using as: the GNU Assembler*, Dean Elsner, Jay Fenlason & friends; *gld: the GNU linker*, Steve Chamberlain and Roland Pesch.

BUGS

For instructions on how to report bugs, type **'info g77 -n Bugs'**.

COPYING

Copyright (c) 1991-1998 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

AUTHORS

See the GNU CC Manual for the contributors to GNU CC. See the GNU Fortran Manual for the contributors to GNU Fortran.

NAME

gcc – GNU project C and C++ compiler

SYNOPSIS

```
gcc [-c|-S|-E] [-std=standard]
    [-g] [-pg] [-Olevel]
    [-Wwarn...] [-pedantic]
    [-Idir...] [-Ldir...]
    [-Dmacro[=defn]...] [-Umacro]
    [-foption...] [-mmachine-option...]
    [-o outfile] infile...
```

Only the most useful options are listed here; see below for the remainder. **g++** accepts mostly the same options as **gcc**.

DESCRIPTION

When you invoke GCC, it normally does preprocessing, compilation, assembly and linking. The “overall options” allow you to stop this process at an intermediate stage. For example, the **-c** option says not to run the linker. Then the output consists of object files output by the assembler.

Other options are passed on to one stage of processing. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; most of these are not documented here, since you rarely need to use any of them.

Most of the command line options that you can use with GCC are useful for C programs; when an option is only useful with another language (usually C++), the explanation says so explicitly. If the description for a particular option does not mention a source language, you can use that option with all supported languages.

The **gcc** program accepts options and file names as operands. Many options have multi-letter names; therefore multiple single-letter options may *not* be grouped: **-dr** is very different from **-d -r**.

You can mix options and other arguments. For the most part, the order you use doesn’t matter. Order does matter when you use several options of the same kind; for example, if you specify **-L** more than once, the directories are searched in the order specified.

Many options have long names starting with **-f** or with **-W**—for example, **-fforce-mem**, **-fstrength-reduce**, **-Wformat** and so on. Most of these have both positive and negative forms; the negative form of **-ffoo** would be **-fno-foo**. This manual documents only one of these two forms, whichever one is not the default.

OPTIONS**Option Summary**

Here is a summary of all the options, grouped by type. Explanations are in the following sections.

Overall Options

-c -S -E -o file -pipe -pass-exit-codes -x language -v --target-help --help

C Language Options

-ansi -std=standard -fno-asm -fno-builtin -fhosted -ffreestanding -trigraphs -traditional -traditional-cpp -fallow-single-precision -fcond-mismatch -fsigned-bitfields -fsigned-char -funsigned-bitfields -funsigned-char -fwritable-strings -fshort-wchar

C++ Language Options

-fno-access-control -fcheck-new -fconserve-space -fdollars-in-identifiers -fno-elide-constructors -fno-enforce-eh-specs -fexternal-templates -falt-external-templates -ffor-scope -fno-for-scope -fno-gnu-keywords -fhonor-std -fhuge-objects -fno-implicit-templates -fno-implicit-inline-templates -fno-implement-inlines -fms-extensions -fno-operator-names -fno-optional-diags -fpermissive -frepo -fno-rtti -ftemplate-depth-*n* -fuse-cxa-atexit -fvtable-thunks -nostdinc++ -fno-default-inline -Wctor-dtor-privacy -Wnon-virtual-dtor -Wreorder -Weffc++ -Wno-deprecated -Wno-non-template-friend -Wold-style-cast -Woverloaded-virtual -Wno-pmf-conversions -Wsign-promo -Wsynth

Language Independent Options

-fmessage-length=*n* **-fdiagnostics-show-location=[once|every-line]**

Warning Options

-fsyntax-only **-pedantic** **-pedantic-errors** **-w** **-W** **-Wall** **-Waggregate-return** **-Wcast-align** **-Wcast-qual** **-Wchar-subscripts** **-Wcomment** **-Wconversion** **-Wdisabled-optimization** **-Werror** **-Wfloat-equal** **-Wformat** **-Wformat=2** **-Wformat-nonliteral** **-Wformat-security** **-Wid-clash-len** **-Wimplicit** **-Wimplicit-int** **-Wimplicit-function-declaration** **-Werror-implicit-function-declaration** **-Wimport** **-Winline** **-Wlarger-than-len** **-Wlong-long** **-Wmain** **-Wmissing-declarations** **-Wmissing-format-attribute** **-Wmissing-noreturn** **-Wmultichar** **-Wno-format-extra-args** **-Wno-format-y2k** **-Wno-import** **-Wpacked** **-Wpadded** **-Wparentheses** **-Wpointer-arith** **-Wredundant-decls** **-Wreturn-type** **-Wsequence-point** **-Wshadow** **-Wsign-compare** **-Wswitch** **-Wsystem-headers** **-Wtrigraphs** **-Wundef** **-Wuninitialized** **-Wunknown-pragmas** **-Wunreachable-code** **-Wunused** **-Wunused-function** **-Wunused-label** **-Wunused-parameter** **-Wunused-value** **-Wunused-variable** **-Wwrite-strings**

C-only Warning Options

-Wbad-function-cast **-Wmissing-prototypes** **-Wnested-externs** **-Wstrict-prototypes** **-Wtraditional**

Debugging Options

-a **-ax** **-dletters** **-fdump-unnumbered** **-fdump-translation-unit=file** **-fdump-class-layout=file** **-fpretend-float** **-fprofile-arcs** **-ftest-coverage** **-g** **-glevel** **-gcoff** **-gdwarf** **-gdwarf-1** **-gdwarf-1+** **-gdwarf-2** **-ggdb** **-gstabs** **-gstabs+** **-gxcoff** **-gxcoff+** **-p** **-pg** **-print-file-name=library** **-print-libgcc-file-name** **-print-prog-name=program** **-print-search-dirs** **-Q** **-save-temps** **-time**

Optimization Options

-falign-functions=*n* **-falign-jumps=*n*** **-falign-labels=*n*** **-falign-loops=*n*** **-fbranch-probabilities** **-fcaller-saves** **-fcse-follow-jumps** **-fcse-skip-blocks** **-fdata-sections** **-fdce** **-fdelayed-branch** **-fdelete-null-pointer-checks** **-fexpensive-optimizations** **-ffast-math** **-ffloat-store** **-fforce-addr** **-fforce-mem** **-ffunction-sections** **-fgcse** **-finline-functions** **-finline-limit=*n*** **-fkeep-inline-functions** **-fkeep-static-consts** **-fmove-all-movables** **-fno-default-inline** **-fno-defer-pop** **-fno-function-cse** **-fno-guess-branch-probability** **-fno-inline** **-fno-math-errno** **-fno-peephole** **-fomit-frame-pointer** **-foptimize-register-move** **-foptimize-sibling-calls** **-freduce-all-givs** **-fregmove** **-frename-registers** **-frerun-cse-after-loop** **-frerun-loop-opt** **-fschedule-insns** **-fschedule-insns2** **-fsingle-precision-constant** **-fssa** **-fstrength-reduce** **-fstrict-aliasing** **-fthread-jumps** **-ftrapv** **-funroll-all-loops** **-funroll-loops** **--param name=value** **-O** **-O0** **-O1** **-O2** **-O3** **-Os**

Preprocessor Options

-\$ **-Aquestion=answer** **-A-question[=answer]** **-C** **-dD** **-dI** **-dM** **-dN** **-Dmacro[=defn]** **-E** **-H** **-idirafter dir** **-include file** **-imacros file** **-iprefix file** **-iwithprefix dir** **-iwithprefixbefore dir** **-isystem dir** **-isystem-c++ dir** **-M** **-MM** **-MF** **-MG** **-MP** **-MQ** **-MT** **-nostdinc** **-P** **-remap** **-trigraphs** **-undef** **-Umacro** **-Wp,option**

Assembler Option

-Wa,option

Linker Options

object-file-name **-llibrary** **-nostartfiles** **-nodefaultlibs** **-nostdlib** **-s** **-static** **-static-libgcc** **-shared** **-shared-libgcc** **-symbolic** **-Wl,option** **-Xlinker option** **-u symbol**

Directory Options

-Bprefix **-Idir** **-I-** **-Ldir** **-specs=file**

Target Options

-b machine **-V version**

*Machine Dependent Options**M680x0 Options*

**-m68000 -m68020 -m68020-40 -m68020-60 -m68030 -m68040 -m68060 -mcpu32
-m5200 -m68881 -mbitfield -mc68000 -mc68020 -mfpa -mnobitfield -mrtd -mshort
-msoft-float -mpcrel -malign-int -mstrict-align**

M68hc1x Options

-m6811 -m6812 -m68hc11 -m68hc12 -mauto-incdec -mshort -msoft-reg-count=*count*

VAX Options

-mg -mgnu -munix

SPARC Options

-mcpu=*cpu type* -mtune=*cpu type* -mcmodel=*code model* -m32 -m64 -mapp-regs -mbroken-saverstore -mcyppress -mepilogue -mfaster-structs -mflat -mfpu -mhard-float -mhard-quad-float -mimpure-text -mlive-g0 -mno-app-regs -mno-epilogue -mno-faster-structs -mno-flat -mno-fpu -mno-impure-text -mno-stack-bias -mno-unaligned-doubles -msoft-float -msoft-quad-float -msparclite -mstack-bias -msupersparc -munaligned-doubles -mv8

Convex Options

**-mc1 -mc2 -mc32 -mc34 -mc38 -margcount -mnoargcount -mlong32 -mlong64
-mvolatile-cache -mvolatile-nocache**

AMD29K Options

-m29000 -m29050 -mbw -mnbw -mdw -mndw -mlarge -mnormal -msmall -mkernel-registers -mno-reuse-arg-regs -mno-stack-check -mno-storem-bug -mreuse-arg-regs -msoft-float -mstack-check -mstorem-bug -muser-registers

ARM Options

-mapcs-frame -mno-apcs-frame -mapcs-26 -mapcs-32 -mapcs-stack-check -mno-apcs-stack-check -mapcs-float -mno-apcs-float -mapcs-reentrant -mno-apcs-reentrant -msched-prolog -mno-sched-prolog -mlittle-endian -mbig-endian -mwords-little-endian -malignment-traps -mno-alignment-traps -msoft-float -mhard-float -mfpe -mthumb-interwork -mno-thumb-interwork -mcpu= -march= -mfpe= -mstructure-size-boundary= -mbsd -mxopen -mno-sym-rename -mabort-on-noreturn -mlong-calls -mno-long-calls -mnop-fun-dllimport -mno-nop-fun-dllimport -msingle-pic-base -mno-single-pic-base -mpic-register=

Thumb Options

-mtpcs-frame -mno-tpcs-frame -mtpcs-leaf-frame -mno-tpcs-leaf-frame -mlittle-endian -mbig-endian -mthumb-interwork -mno-thumb-interwork -mstructure-size-boundary= -mnop-fun-dllimport -mno-nop-fun-dllimport -mcallee-super-interworking -mno-callee-super-interworking -mcaller-super-interworking -mno-caller-super-interworking -msingle-pic-base -mno-single-pic-base -mpic-register=

MN10200 Options

-mrelax

MN10300 Options

-mmult-bug -mno-mult-bug -mam33 -mno-am33 -mrelax

M32R/D Options

-mcode-model=*model type* -msdata=*sdata type* -G *num*

M88K Options

-m88000 -m88100 -m88110 -mbig-pic -mcheck-zero-division -mhandle-large-shift -midentify-revision -mno-check-zero-division -mno-ocs-debug-info -mno-ocs-frame-position -mno-optimize-arg-area -mno-serialize-volatile -mno-underscores -mocs-debug-info -mocs-frame-position -moptimize-arg-area -mserialize-volatile -mshort-data-*num* -msvr3 -msvr4 -mtrap-large-shift -muse-div-instruction -mversion-03.00 -mwarn-passed-structs

RS/6000 and PowerPC Options

-mcpu=*cpu type* -mtune=*cpu type* -mpower -mno-power -mpower2 -mno-power2 -mpowerpc -mpowerpc64 -mno-powerpc -mpowerpc-gpopt -mno-powerpc-gpopt -mpowerpc-gfxopt -mno-powerpc-gfxopt -mnew-mnemonics -mold-mnemonics -mfull-toc -mminimal-toc -mno-fop-in-toc -mno-sum-in-toc -m64 -m32 -mxl-call -mno-xl-call -mthreads -mpe -msoft-float -mhard-float -mmultiple -mno-multiple -mstring -mno-string -mupdate -mno-update -mfused-madd -mno-fused-madd -mbit-align -mno-bit-align -mstrict-align -mno-strict-align -mrelocatable -mno-relocatable -mrelocatable-lib -mno-relocatable-lib -mtoc -mno-toc -mlittle -mlittle-endian -mbig -mbig-endian -mcall-aix -mcall-sysv -mprototype -mno-prototype -msim -mmvme -mads -myellowknife -memb -msdata -msdata=*opt* -mvxworks -G *num*

RT Options

-mcall-lib-mul -mfp-arg-in-fpregs -mfp-arg-in-gregs -mfull-fp-blocks -mhc-struct-return -min-line-mul -mminimum-fp-blocks -mnohc-struct-return

MIPS Options

-mabicalls -mcpu=*cpu type* -membedded-data -muninit-const-in-rodata -membedded-pic -mfp32 -mfp64 -mgas -mfp32 -mfp64 -mgpopt -mhalf-pic -mhard-float -mint64 -mips1 -mips2 -mips3 -mips4 -mlong64 -mlong32 -mlong-calls -mmemcpy -mmips-as -mmips-tfile -mno-abicalls -mno-embedded-data -mno-uninit-const-in-rodata -mno-embedded-pic -mno-gpopt -mno-long-calls -mno-memcpy -mno-mips-tfile -mno-rnames -mno-stats -mrnames -msoft-float -m4650 -msingle-float -mmad -mstats -EL -EB -G *num* -nocpp -mabi=32 -mabi=n32 -mabi=64 -mabi=eabi -mfix7000 -mno-crt0

i386 Options

-mcpu=*cpu type* -march=*cpu type* -mintel-syntax -mieee-fp -mno-fancy-math-387 -mno-fp-ret-in-387 -msoft-float -msvr3-shlib -mno-wide-multiply -mrtd -malign-double -malign-jumps=*num* -malign-loops=*num* -malign-functions=*num* -mpreferred-stack-boundary=*num* -mthreads -mno-align-stringops -minline-all-stringops -mpush-args -maccumulate-outgoing-args -m128bit-long-double -m96bit-long-double -mregparm=*num*

HPPA Options

-march=*architecture type* -mbig-switch -mdisable-fpregs -mdisable-indexing -mfast-indirect-calls -mgas -mjump-in-delay -mlong-load-store -mno-big-switch -mno-disable-fpregs -mno-disable-indexing -mno-fast-indirect-calls -mno-gas -mno-jump-in-delay -mno-long-load-store -mno-portable-runtime -mno-soft-float -mno-space-regs -msoft-float -mpa-risc-1-0 -mpa-risc-1-1 -mpa-risc-2-0 -mportable-runtime -mschedule=*cpu type* -mspace-regs

Intel 960 Options

-mcpu *type* -masm-compatible -mclean-linkage -mcode-align -mcomplex-addr -mleaf-procedures -mic-compatible -mic2.0-compatible -mic3.0-compatible -mintel-asm -mno-clean-linkage -mno-code-align -mno-complex-addr -mno-leaf-procedures -mno-old-align -mno-strict-align -mno-tail-call -mnumerics -mold-align -msoft-float -mstrict-align -mtail-call

DEC Alpha Options

-mfp-regs -mno-fp-regs -mno-soft-float -msoft-float -malpha-as -mgas -mieee -mieee-with-inexact -mieee-conformant -mfp-trap-mode=*mode* -mfp-rounding-mode=*mode* -mtrap-

precision=mode **-mbuild-constants** **-mcpu=cpu type** **-mbwx** **-mno-bwx** **-mcix** **-mno-cix** **-mmax**
-mno-max **-mmemory-latency=time**

Clipper Options

-mc300 **-mc400**

H8/300 Options

-mrelax **-mh** **-ms** **-mint32** **-malign-300**

SH Options

-m1 **-m2** **-m3** **-m3e** **-m4-nofpu** **-m4-single-only** **-m4-single** **-m4** **-mb** **-ml** **-mdalign**
-mrelax **-mbigtable** **-mfmovd** **-mhitachi** **-mnomacsave** **-misize** **-mpadstruct** **-mspace**
-mprefergot **-musermode**

System V Options

-Qy **-Qn** **-YP_{paths}** **-Ym_{dir}**

ARC Options

-EB **-EL** **-mmangle-cpu** **-mcpu=cpu** **-mtext=text section** **-mdata=data section** **-mro-**
data=readonly data section

TMS320C3x/C4x Options

-mcpu=cpu **-mbig** **-msmall** **-mregparm** **-mmemparm** **-mfast-fix** **-mmpyi** **-mbk** **-mti** **-mdp-**
isr-reload **-mrpts=count** **-mrptb** **-mdb** **-mloop-unsigned** **-mparallel-insns** **-mparallel-mpy**
-mpreserve-float

V850 Options

-mlong-calls **-mno-long-calls** **-mep** **-mno-ep** **-mprolog-function** **-mno-prolog-function**
-mspace **-mtda=n** **-msda=n** **-mzda=n** **-mv850** **-mbig-switch**

NS32K Options

-m32032 **-m32332** **-m32532** **-m32081** **-m32381** **-mmult-add** **-mnomult-add** **-msoft-float** **-mrtd**
-mnortd **-mregparam** **-mnoregparam** **-msb** **-mnosb** **-mbitfield** **-mnobitfield** **-mhimem** **-mno-**
himem

AVR Options

-mmcu=mcu **-msize** **-minit-stack=n** **-mno-interrupts** **-mcall-prologues** **-mno-tablejump**
-mtiny-stack

MCore Options

-mhardlit **-mno-hardlit** **-mdiv** **-mno-div** **-mrelax-immediates** **-mno-relax-immediates** **-mwide-**
bitfields **-mno-wide-bitfields** **-m4byte-functions** **-mno-4byte-functions** **-mcallgraph-data** **-mno-**
callgraph-data **-mslow-bytes** **-mno-slow-bytes** **-mno-lsim** **-mlittle-endian** **-mbig-endian** **-m210**
-m340 **-mstack-increment**

IA-64 Options

-mbig-endian **-mlittle-endian** **-mgnu-as** **-mgnu-ld** **-mno-pic** **-mvolatile-asm-stop** **-mb-step**
-mregister-names **-mno-sdata** **-mconstant-gp** **-mauto-pic** **-minline-divide-min-latency** **-min-**
line-divide-max-throughput **-mno-dwarf2-asm** **-mfixed-range=register range**

Code Generation Options

-fcall-saved-reg **-fcall-used-reg** **-fexceptions** **-funwind-tables** **-ffixed-reg** **-finhibit-size-direc-**
tive **-finstrument-functions** **-fcheck-memory-usage** **-fprefix-function-name** **-fno-common**
-fno-ident **-fno-gnu-linker** **-fpcc-struct-return** **-fpic** **-fPIC** **-freg-struct-return** **-fshared-data**
-fshort-enums **-fshort-double** **-fvolatile** **-fvolatile-global** **-fvolatile-static** **-fverbose-asm**

-fpack-struct -fstack-check -fstack-limit-register=*reg* -fstack-limit-symbol=*sym* -fargument-alias -fargument-noalias -fargument-noalias-global -fleading-underscore

Options Controlling the Kind of Output

Compilation can involve up to four stages: preprocessing, compilation proper, assembly and linking, always in that order. The first three stages apply to an individual source file, and end by producing an object file; linking combines all the object files (those newly compiled, and those specified as input) into an executable file.

For any given input file, the file name suffix determines what kind of compilation is done:

file.c

C source code which must be preprocessed.

file.i

C source code which should not be preprocessed.

file.ii

C++ source code which should not be preprocessed.

file.m

Objective-C source code. Note that you must link with the library *libobjc.a* to make an Objective-C program work.

file.mi

Objective-C source code which should not be preprocessed.

file.h

C header file (not to be compiled or linked).

file.cc

file.cp

file.cxx

file.cpp

file.c++

file.C

C++ source code which must be preprocessed. Note that in *.cxx*, the last two letters must both be literally *x*. Likewise, *.C* refers to a literal capital C.

file.f

file.for

file.FOR

Fortran source code which should not be preprocessed.

file.F

file.fpp

file.FPP

Fortran source code which must be preprocessed (with the traditional preprocessor).

file.r

Fortran source code which must be preprocessed with a RATFOR preprocessor (not included with GCC).

file.ch

file.chi

CHILL source code (preprocessed with the traditional preprocessor).

file.s

Assembler code.

file.S

Assembler code which must be preprocessed.

other

An object file to be fed straight into linking. Any file name with no recognized suffix is treated this way.

You can specify the input language explicitly with the **-x** option:

-x language

Specify explicitly the *language* for the following input files (rather than letting the compiler choose a default based on the file name suffix). This option applies to all following input files until the next **-x** option. Possible values for *language* are:

```
c      c-header  cpp-output
c++    c++-cpp-output
objective-c  objc-cpp-output
assembler  assembler-with-cpp
f77     f77-cpp-input  ratfor
java    chill
```

-x none

Turn off any specification of a language, so that subsequent files are handled according to their file name suffixes (as they are if **-x** has not been used at all).

-pass-exit-codes

Normally the **gcc** program will exit with the code of 1 if any phase of the compiler returns a non-success return code. If you specify **-pass-exit-codes**, the **gcc** program will instead return with numerically highest error produced by any phase that returned an error indication.

If you only want some of the stages of compilation, you can use **-x** (or filename suffixes) to tell **gcc** where to start, and one of the options **-c**, **-S**, or **-E** to say where **gcc** is to stop. Note that some combinations (for example, **-x cpp-output -E**) instruct **gcc** to do nothing at all.

-c Compile or assemble the source files, but do not link. The linking stage simply is not done. The ultimate output is in the form of an object file for each source file.

By default, the object file name for a source file is made by replacing the suffix **.c**, **.i**, **.s**, etc., with **.o**.

Unrecognized input files, not requiring compilation or assembly, are ignored.

-S Stop after the stage of compilation proper; do not assemble. The output is in the form of an assembler code file for each non-assembler input file specified.

By default, the assembler file name for a source file is made by replacing the suffix **.c**, **.i**, etc., with **.s**.

Input files that don't require compilation are ignored.

-E Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code, which is sent to the standard output.

Input files which don't require preprocessing are ignored.

-o file

Place output in file *file*. This applies regardless to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code.

Since only one output file can be specified, it does not make sense to use **-o** when compiling more than one input file, unless you are producing an executable file as output.

If **-o** is not specified, the default is to put an executable file in *a.out*, the object file for *source.suffix* in *source.o*, its assembler file in *source.s*, and all preprocessed C source on standard output.

-v Print (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper.

-pipe

Use pipes rather than temporary files for communication between the various stages of compilation. This fails to work on some systems where the assembler is unable to read from a pipe; but the GNU assembler has no trouble.

—help

Print (on the standard output) a description of the command line options understood by **gcc**. If the **-v** option is also specified then **—help** will also be passed on to the various processes invoked by **gcc**, so that they can display the command line options they accept. If the **-W** option is also specified then command line options which have no documentation associated with them will also be displayed.

—target-help

Print (on the standard output) a description of target specific command line options for each tool.

Compiling C++ Programs

C++ source files conventionally use one of the suffixes **.C**, **.cc**, **.cpp**, **.c++**, **.cp**, or **.cxx**; preprocessed C++ files use the suffix **.ii**. GCC recognizes files with these names and compiles them as C++ programs even if you call the compiler the same way as for compiling C programs (usually with the name **gcc**).

However, C++ programs often require class libraries as well as a compiler that understands the C++ language—and under some circumstances, you might want to compile programs from standard input, or otherwise without a suffix that flags them as C++ programs. **g++** is a program that calls GCC with the default language set to C++, and automatically specifies linking against the C++ library. On many systems, **g++** is also installed with the name **c++**.

When you compile C++ programs, you may specify many of the same command-line options that you use for compiling programs in any language; or command-line options meaningful for C and related languages; or options that are meaningful only for C++ programs.

Options Controlling C Dialect

The following options control the dialect of C (or languages derived from C, such as C++ and Objective C) that the compiler accepts:

-ansi

In C mode, support all ISO C89 programs. In C++ mode, remove GNU extensions that conflict with ISO C++.

This turns off certain features of GCC that are incompatible with ISO C (when compiling C code), or of standard C++ (when compiling C++ code), such as the **asm** and **typeof** keywords, and predefined macros such as **unix** and **vax** that identify the type of system you are using. It also enables the undesirable and rarely used ISO trigraph feature. For the C compiler, it disables recognition of C++ style **//** comments as well as the **inline** keyword.

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**.

The macro **__STRICT_ANSI__** is predefined when the **-ansi** option is used. Some header files may notice this macro and refrain from declaring certain functions or defining certain macros that the ISO standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things.

Functions which would normally be builtin but do not have semantics defined by ISO C (such as

`alloca` and `ffs`) are not builtin functions with **-ansi** is used.

-std=

Determine the language standard. A value for this option must be provided; possible values are

iso9899:1990

Same as **-ansi**

iso9899:199409

ISO C as modified in amend. 1

iso9899:1999

ISO C99. Note that this standard is not yet fully supported; see <<http://gcc.gnu.org/c99status.html>> for more information.

c89 same as **-std=iso9899:1990**

c99 same as **-std=iso9899:1999**

gnu89

default, iso9899:1990 + gnu extensions

gnu99

iso9899:1999 + gnu extensions

iso9899:199x

same as **-std=iso9899:1999**, deprecated

c9x same as **-std=iso9899:1999**, deprecated

gnu9x

same as **-std=gnu99**, deprecated

Even when this option is not specified, you can still use some of the features of newer standards in so far as they do not conflict with previous C standards. For example, you may use `__restrict__` even when **-std=c99** is not specified.

The **-std** options specifying some version of ISO C have the same effects as **-ansi**, except that features that were not in ISO C89 but are in the specified version (for example, `//` comments and the `inline` keyword in ISO C99) are not disabled.

-fno-asm

Do not recognize `asm`, `inline` or `typeof` as a keyword, so that code can use these words as identifiers. You can use the keywords `__asm__`, `__inline__` and `__typeof__` instead. **-ansi** implies **-fno-asm**.

In C++, this switch only affects the `typeof` keyword, since `asm` and `inline` are standard keywords. You may want to use the **-fno-gnu-keywords** flag instead, which has the same effect. In C99 mode (**-std=c99** or **-std=gnu99**), this switch only affects the `asm` and `typeof` keywords, since `inline` is a standard keyword in ISO C99.

-fno-builtin

Don't recognize builtin functions that do not begin with `__builtin_` as prefix.

GCC normally generates special code to handle certain builtin functions more efficiently; for instance, calls to `alloca` may become single instructions that adjust the stack directly, and calls to `memcpy` may become inline copy loops. The resulting code is often both smaller and faster, but since the function calls no longer appear as such, you cannot set a breakpoint on those calls, nor can you change the behavior of the functions by linking with a different library.

-fhosted

Assert that compilation takes place in a hosted environment. This implies **-fbuiltin**. A hosted environment is one in which the entire standard library is available, and in which `main` has a return type of `int`. Examples are nearly everything except a kernel. This is equivalent to **-fno-freestanding**.

-ffreestanding

Assert that compilation takes place in a freestanding environment. This implies **-fno-builtin**. A freestanding environment is one in which the standard library may not exist, and program startup may not necessarily be at `main`. The most obvious example is an OS kernel. This is equivalent to **-fno-hosted**.

-trigraphs

Support ISO C trigraphs. You don't want to know about this brain-damage. The **-ansi** option (and **-std** options for strict ISO C conformance) implies **-trigraphs**.

-traditional

Attempt to support some aspects of traditional C compilers. Specifically:

- All `extern` declarations take effect globally even if they are written inside of a function definition. This includes implicit declarations of functions.
- The newer keywords `typeof`, `inline`, `signed`, `const` and `volatile` are not recognized. (You can still use the alternative keywords such as `__typeof__`, `__inline__`, and so on.)
- Comparisons between pointers and integers are always allowed.
- Integer types `unsigned short` and `unsigned char` promote to `unsigned int`.
- Out-of-range floating point literals are not an error.
- Certain constructs which ISO regards as a single invalid preprocessing number, such as **0xe-0xd**, are treated as expressions instead.
- String “constants” are not necessarily constant; they are stored in writable space, and identical looking constants are allocated separately. (This is the same as the effect of **-fwritable-strings**.)
- All automatic variables not declared `register` are preserved by `longjmp`. Ordinarily, GNU C follows ISO C: automatic variables not declared `volatile` may be clobbered.
- The character escape sequences `\x` and `\a` evaluate as the literal characters `x` and `a` respectively. Without **-traditional**, `\x` is a prefix for the hexadecimal representation of a character, and `\a` produces a bell.

You may wish to use **-fno-builtin** as well as **-traditional** if your program uses names that are normally GNU C builtin functions for other purposes of its own.

You cannot use **-traditional** if you include any header files that rely on ISO C features. Some vendors are starting to ship systems with ISO C header files and you cannot use **-traditional** on such systems to compile files that include any system headers.

The **-traditional** option also enables **-traditional-cpp**, which is described next.

-traditional-cpp

Attempt to support some aspects of traditional C preprocessors. Specifically:

- Comments convert to nothing at all, rather than to a space. This allows traditional token concatenation.
- In a preprocessing directive, the `#` symbol must appear as the first character of a line.
- Macro arguments are recognized within string constants in a macro definition (and their values are stringified, though without additional quote marks, when they appear in such a context). The preprocessor always considers a string constant to end at a newline.
- The predefined macro `__STDC__` is not defined when you use **-traditional**, but `__GNUC__` is (since the GNU extensions which `__GNUC__` indicates are not affected by **-traditional**). If you need to write header files that work differently depending on whether **-traditional** is in use, by testing both of these predefined macros you can distinguish four situations: GNU C, traditional GNU C, other ISO C compilers, and other old C compilers. The predefined macro `__STDC_VERSION__` is also not defined when you use **-traditional**.

- The preprocessor considers a string constant to end at a newline (unless the newline is escaped with `\`). (Without **-traditional**, string constants can contain the newline character as typed.)

-fcond-mismatch

Allow conditional expressions with mismatched types in the second and third arguments. The value of such an expression is void. This option is not supported for C++.

-funsigned-char

Let the type `char` be unsigned, like `unsigned char`.

Each kind of machine has a default for what `char` should be. It is either like `unsigned char` by default or like `signed char` by default.

Ideally, a portable program should always use `signed char` or `unsigned char` when it depends on the signedness of an object. But many programs have been written to use plain `char` and expect it to be signed, or expect it to be unsigned, depending on the machines they were written for. This option, and its inverse, let you make such a program work with the opposite default.

The type `char` is always a distinct type from each of `signed char` or `unsigned char`, even though its behavior is always just like one of those two.

-fsigned-char

Let the type `char` be signed, like `signed char`.

Note that this is equivalent to **-fno-unsigned-char**, which is the negative form of **-funsigned-char**. Likewise, the option **-fno-signed-char** is equivalent to **-funsigned-char**.

You may wish to use **-fno-builtin** as well as **-traditional** if your program uses names that are normally GNU C builtin functions for other purposes of its own.

You cannot use **-traditional** if you include any header files that rely on ISO C features. Some vendors are starting to ship systems with ISO C header files and you cannot use **-traditional** on such systems to compile files that include any system headers.

-fsigned-bitfields

-funsigned-bitfields

-fno-signed-bitfields

-fno-unsigned-bitfields

These options control whether a bitfield is signed or unsigned, when the declaration does not use either `signed` or `unsigned`. By default, such a bitfield is signed, because this is consistent: the basic integer types such as `int` are signed types.

However, when **-traditional** is used, bitfields are all unsigned no matter what.

-fwritable-strings

Store string constants in the writable data segment and don't uniquize them. This is for compatibility with old programs which assume they can write into string constants. The option **-traditional** also has this effect.

Writing into string constants is a very bad idea; "constants" should be constant.

-fallow-single-precision

Do not promote single precision math operations to double precision, even when compiling with **-traditional**.

Traditional K&R C promotes all floating point operations to double precision, regardless of the sizes of the operands. On the architecture for which you are compiling, single precision may be faster than double precision. If you must use **-traditional**, but want to use single precision operations when the operands are single precision, use this option. This option has no effect when compiling with ISO or GNU C conventions (the default).

-fshort-wchar

Override the underlying type for `wchar_t` to be **short unsigned int** instead of the default for the target. This option is useful for building programs to run under WINE.

Options Controlling C++ Dialect

This section describes the command-line options that are only meaningful for C++ programs; but you can also use most of the GNU compiler options regardless of what language your program is in. For example, you might compile a file `firstClass.C` like this:

```
g++ -g -frepo -O -c firstClass.C
```

In this example, only **-frepo** is an option meant only for C++ programs; you can use the other options with any language supported by GCC.

Here is a list of options that are *only* for compiling C++ programs:

-fno-access-control

Turn off all access checking. This switch is mainly useful for working around bugs in the access control code.

-fcheck-new

Check that the pointer returned by `operator new` is non-null before attempting to modify the storage allocated. The current Working Paper requires that `operator new` never return a null pointer, so this check is normally unnecessary.

An alternative to using this option is to specify that your `operator new` does not throw any exceptions; if you declare it *throw()*, g++ will check the return value. See also **new (nothrow)**.

-fconserve-space

Put uninitialized or runtime-initialized global variables into the common segment, as C does. This saves space in the executable at the cost of not diagnosing duplicate definitions. If you compile with this flag and your program mysteriously crashes after `main()` has completed, you may have an object that is being destroyed twice because two definitions were merged.

This option is no longer useful on most targets, now that support has been added for putting variables into BSS without making them common.

-fdollars-in-identifiers

Accept `$` in identifiers. You can also explicitly prohibit use of `$` with the option **-fno-dollars-in-identifiers**. (GNU C allows `$` by default on most target systems, but there are a few exceptions.) Traditional C allowed the character `$` to form part of identifiers. However, ISO C and C++ forbid `$` in identifiers.

-fno-elide-constructors

The C++ standard allows an implementation to omit creating a temporary which is only used to initialize another object of the same type. Specifying this option disables that optimization, and forces g++ to call the copy constructor in all cases.

-fno-enforce-eh-specs

Don't check for violation of exception specifications at runtime. This option violates the C++ standard, but may be useful for reducing code size in production builds, much like defining **NDEBUG**. The compiler will still optimize based on the exception specifications.

-fexternal-templates

Cause template instantiations to obey **#pragma interface** and **implementation**; template instances are emitted or not according to the location of the template definition.

This option is deprecated.

-falt-external-templates

Similar to `-fexternal-templates`, but template instances are emitted or not according to the place where they are first instantiated.

This option is deprecated.

-ffor-scope**-fno-for-scope**

If `-ffor-scope` is specified, the scope of variables declared in a *for-init-statement* is limited to the **for** loop itself, as specified by the C++ standard. If `-fno-for-scope` is specified, the scope of variables declared in a *for-init-statement* extends to the end of the enclosing scope, as was the case in old versions of gcc, and other (traditional) implementations of C++.

The default if neither flag is given to follow the standard, but to allow and give a warning for old-style code that would otherwise be invalid, or have different behavior.

-fno-gnu-keywords

Do not recognize `typeof` as a keyword, so that code can use this word as an identifier. You can use the keyword `__typeof__` instead. `-ansi` implies **-fno-gnu-keywords**.

-fhonor-std

Treat the namespace `std` as a namespace, instead of ignoring it. For compatibility with earlier versions of g++, the compiler will, by default, ignore namespace-declarations, using-declarations, using-directives, and namespace-names, if they involve `std`.

-fhuge-objects

Support virtual function calls for objects that exceed the size representable by a **short int**. Users should not use this flag by default; if you need to use it, the compiler will tell you so.

This flag is not useful when compiling with `-fvtable-thunks`.

Like all options that change the ABI, all C++ code, *including libgcc* must be built with the same setting of this option.

-fno-implicit-templates

Never emit code for non-inline templates which are instantiated implicitly (i.e. by use); only emit code for explicit instantiations.

-fno-implicit-inline-templates

Don't emit code for implicit instantiations of inline templates, either. The default is to handle inlines differently so that compiles with and without optimization will need the same set of explicit instantiations.

-fno-implement-inlines

To save space, do not emit out-of-line copies of inline functions controlled by **#pragma implementation**. This will cause linker errors if these functions are not inlined everywhere they are called.

-fms-extensions

Disable pedantic warnings about constructs used in MFC, such as implicit int and getting a pointer to member function via non-standard syntax.

-fno-operator-names

Do not treat the operator name keywords `and`, `bitand`, `bitor`, `compl`, `not`, `or` and `xor` as synonyms as keywords.

-fno-optional-diags

Disable diagnostics that the standard says a compiler does not need to issue. Currently, the only such diagnostic issued by g++ is the one for a name having multiple meanings within a class.

-fpermissive

Downgrade messages about nonconformant code from errors to warnings. By default, g++ effectively sets **-pedantic-errors** without **-pedantic**; this option reverses that. This behavior and this option are superseded by **-pedantic**, which works as it does for GNU C.

-frepo

Enable automatic template instantiation. This option also implies **-fno-implicit-templates**.

-fno-rtti

Disable generation of information about every class with virtual functions for use by the C++ runtime type identification features (**dynamic_cast** and **typeid**). If you don't use those parts of the language, you can save some space by using this flag. Note that exception handling uses the same information, but it will generate it as needed.

-ftemplate-depth-*n*

Set the maximum instantiation depth for template classes to *n*. A limit on the template instantiation depth is needed to detect endless recursions during template class instantiation. ANSI/ISO C++ conforming programs must not rely on a maximum depth greater than 17.

-fuse-cxa-atexit

Register destructors for objects with static storage duration with the `__cxa_atexit` function rather than the `atexit` function. This option is required for fully standards-compliant handling of static destructors, but will only work if your C library supports `__cxa_atexit`.

-fvtable-thunks

Use **thunks** to implement the virtual function dispatch table (**vtable**). The traditional (cfront-style) approach to implementing vtables was to store a pointer to the function and two offsets for adjusting the **this** pointer at the call site. Newer implementations store a single pointer to a **thunk** function which does any necessary adjustment and then calls the target function.

This option also enables a heuristic for controlling emission of vtables; if a class has any non-inline virtual functions, the vtable will be emitted in the translation unit containing the first one of those.

Like all options that change the ABI, all C++ code, *including libgcc.a* must be built with the same setting of this option.

-nostdinc++

Do not search for header files in the standard directories specific to C++, but do still search the other standard directories. (This option is used when building the C++ library.)

In addition, these optimization, warning, and code generation options have meanings only for C++ programs:

-fno-default-inline

Do not assume **inline** for functions defined inside a class scope.

Note that these functions will have linkage like inline functions; they just won't be inlined by default.

-Wctor-dtor-privacy (C++ only)

Warn when a class seems unusable, because all the constructors or destructors in a class are private and the class has no friends or public static member functions.

-Wnon-virtual-dtor (C++ only)

Warn when a class declares a non-virtual destructor that should probably be virtual, because it looks like the class will be used polymorphically.

-Wreorder (C++ only)

Warn when the order of member initializers given in the code does not match the order in which they must be executed. For instance:

```
struct A {
    int i;
    int j;
    A(): j (0), i (1) { }
};
```

Here the compiler will warn that the member initializers for **i** and **j** will be rearranged to match the declaration order of the members.

The following **-W...** options are not affected by **-Wall**.

-Weffc++ (C++ only)

Warn about violations of various style guidelines from Scott Meyers' *Effective C++* books. If you use this option, you should be aware that the standard library headers do not obey all of these guidelines; you can use **grep -v** to filter out those warnings.

-Wno-deprecated (C++ only)

Do not warn about usage of deprecated features.

-Wnon-template-friend (C++ only)

Disable warnings when non-templated friend functions are declared within a template. With the advent of explicit template specification support in g++, if the name of the friend is an unqualified-id (ie, **friend foo(int)**), the C++ language specification demands that the friend declare or define an ordinary, nontemplate function. (Section 14.5.3). Before g++ implemented explicit specification, unqualified-ids could be interpreted as a particular specialization of a templated function. Because this non-conforming behavior is no longer the default behavior for g++, **-Wnon-template-friend** allows the compiler to check existing code for potential trouble spots, and is on by default. This new compiler behavior can be turned off with **-Wno-non-template-friend** which keeps the conformant compiler code but disables the helpful warning.

-Wold-style-cast (C++ only)

Warn if an old-style (C-style) cast is used within a C++ program. The new-style casts (**static_cast**, **reinterpret_cast**, and **const_cast**) are less vulnerable to unintended effects.

-Woverloaded-virtual (C++ only)

Warn when a function declaration hides virtual functions from a base class. For example, in:

```
struct A {
    virtual void f();
};

struct B: public A {
    void f(int);
};
```

the A class version of `f` is hidden in B, and code like this:

```
B* b;
b->f();
```

will fail to compile.

-Wno-pmf-conversions (C++ only)

Disable the diagnostic for converting a bound pointer to member function to a plain pointer.

-Wsign-promo (C++ only)

Warn when overload resolution chooses a promotion from unsigned or enumerational type to a signed type over a conversion to an unsigned type of the same size. Previous versions of g++ would try to preserve unsignedness, but the standard mandates the current behavior.

-Wsynth (C++ only)

Warn when g++'s synthesis behavior does not match that of cfront. For instance:

```
struct A {
    operator int ();
    A& operator = (int);
};
```

```
main ()
{
    A a,b;
    a = b;
}
```

In this example, g++ will synthesize a default **A& operator = (const A&);**, while cfront will use the user-defined **operator =**.

Options to Control Diagnostic Messages Formatting

Traditionally, diagnostic messages have been formatted irrespective of the output device's aspect (e.g. its width, ...). The options described below can be used to control the diagnostic messages formatting algorithm, e.g. how many characters per line, how often source location information should be reported. Right now, only the C++ front-end can honor these options. However it is expected, in the near future, that the remaining front-ends would be able to digest them correctly.

-fmessage-length=*n*

Try to format error messages so that they fit on lines of about *n* characters. The default is 72 characters for g++ and 0 for the rest of the front-ends supported by GCC. If *n* is zero, then no line-wrapping will be done; each error message will appear on a single line.

-fdiagnostics-show-location=once

Only meaningful in line-wrapping mode. Instructs the diagnostic messages reporter to emit *once* source location information; that is, in case the message is too long to fit on a single physical line and has to be wrapped, the source location won't be emitted (as prefix) again, over and over, in subsequent continuation lines. This is the default behaviour.

-fdiagnostics-show-location=every-line

Only meaningful in line-wrapping mode. Instructs the diagnostic messages reporter to emit the same source location information (as prefix) for physical lines that result from the process of breaking a message which is too long to fit on a single line.

Options to Request or Suppress Warnings

Warnings are diagnostic messages that report constructions which are not inherently erroneous but which are risky or suggest there may have been an error.

You can request many specific warnings with options beginning **-W**, for example **-Wimplicit** to request warnings on implicit declarations. Each of these specific warning options also has a negative form beginning **-Wno-** to turn off warnings; for example, **-Wno-implicit**. This manual lists only one of the two forms, whichever is not the default.

These options control the amount and kinds of warnings produced by GCC:

-fsyntax-only

Check the code for syntax errors, but don't do anything beyond that.

-pedantic

Issue all the warnings demanded by strict ISO C and ISO C++; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. For ISO C, follows the version of the ISO C standard specified by any **-std** option used.

Valid ISO C and ISO C++ programs should compile properly with or without this option (though a rare few will require **-ansi** or a **-std** option specifying the required version of ISO C). However, without this option, certain GNU extensions and traditional C and C++ features are supported as well. With this option, they are rejected.

-pedantic does not cause warning messages for use of the alternate keywords whose names begin and end with `__`. Pedantic warnings are also disabled in the expression that follows `__extension__`. However, only system header files should use these escape routes; application programs should avoid

them.

Some users try to use **-pedantic** to check programs for strict ISO C conformance. They soon find that it does not do quite what they want: it finds some non-ISO practices, but not all—only those for which ISO C *requires* a diagnostic, and some others for which diagnostics have been added.

A feature to report any failure to conform to ISO C might be useful in some instances, but would require considerable additional work and would be quite different from **-pedantic**. We don't have plans to support such a feature in the near future.

-pedantic-errors

Like **-pedantic**, except that errors are produced rather than warnings.

-w Inhibit all warning messages.

-Wno-import

Inhibit warning messages about the use of **#import**.

-Wchar-subscripts

Warn if an array subscript has type `char`. This is a common cause of error, as programmers often forget that this type is signed on some machines.

-Wcomment

Warn whenever a comment-start sequence `/*` appears in a `/*` comment, or whenever a Backslash-Newline appears in a `//` comment.

-Wformat

Check calls to `printf` and `scanf`, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense. This includes standard functions, and others specified by format attributes, in the `printf`, `scanf`, `strftime` and `strfmon` (an X/Open extension, not in the C standard) families.

The formats are checked against the format features supported by GNU libc version 2.2. These include all ISO C89 and C99 features, as well as features from the Single Unix Specification and some BSD and GNU extensions. Other library implementations may not support all these features; GCC does not support warning about features that go beyond a particular library's limitations. However, if **-pedantic** is used with **-Wformat**, warnings will be given about format features not in the selected standard version (but not for `strfmon` formats, since those are not in any version of the C standard).

-Wformat is included in **-Wall**. For more control over some aspects of format checking, the options **-Wno-format-y2k**, **-Wno-format-extra-args**, **-Wformat-nonliteral**, **-Wformat-security** and **-Wformat=2** are available, but are not included in **-Wall**.

-Wno-format-y2k

If **-Wformat** is specified, do not warn about `strftime` formats which may yield only a two-digit year.

-Wno-format-extra-args

If **-Wformat** is specified, do not warn about excess arguments to a `printf` or `scanf` format function. The C standard specifies that such arguments are ignored.

-Wformat-nonliteral

If **-Wformat** is specified, also warn if the format string is not a string literal and so cannot be checked, unless the format function takes its format arguments as a `va_list`.

-Wformat-security

If **-Wformat** is specified, also warn about uses of format functions that represent possible security problems. At present, this warns about calls to `printf` and `scanf` functions where the format string is not a string literal and there are no format arguments, as in `printf (foo);`. This may be a security hole if the format string came from untrusted input and contains **%n**. (This is currently a subset of what **-Wformat-nonliteral** warns about, but in future warnings may be added to **-Wformat-security** that are not included in **-Wformat-nonliteral**.)

-Wformat=2

Enable **-Wformat** plus format checks not included in **-Wformat**. Currently equivalent to **-Wformat -Wformat-nonliteral -Wformat-security**.

-Wimplicit-int

Warn when a declaration does not specify a type.

-Wimplicit-function-declaration**-Werror-implicit-function-declaration**

Give a warning (or error) whenever a function is used before being declared.

-Wimplicit

Same as **-Wimplicit-int** and **-Wimplicit-function-declaration**.

-Wmain

Warn if the type of **main** is suspicious. **main** should be a function with external linkage, returning int, taking either zero arguments, two, or three arguments of appropriate types.

-Wmultichar

Warn if a multicharacter constant (**'FOOF'**) is used. Usually they indicate a typo in the user's code, as they have implementation-defined values, and should not be used in portable code.

-Wparentheses

Warn if parentheses are omitted in certain contexts, such as when there is an assignment in a context where a truth value is expected, or when operators are nested whose precedence people often get confused about.

Also warn about constructions where there may be confusion to which **if** statement an **else** branch belongs. Here is an example of such a case:

```

{
    if (a)
        if (b)
            foo ();
    else
        bar ();
}

```

In C, every **else** branch belongs to the innermost possible **if** statement, which in this example is **if (b)**. This is often not what the programmer expected, as illustrated in the above example by indentation the programmer chose. When there is the potential for this confusion, GNU C will issue a warning when this flag is specified. To eliminate the warning, add explicit braces around the innermost **if** statement so there is no way the **else** could belong to the enclosing **if**. The resulting code would look like this:

```

{
    if (a)
    {
        if (b)
            foo ();
        else
            bar ();
    }
}

```

-Wsequence-point

Warn about code that may have undefined semantics because of violations of sequence point rules in the C standard.

The C standard defines the order in which expressions in a C program are evaluated in terms of

sequence points, which represent a partial ordering between the execution of parts of the program: those executed before the sequence point, and those executed after it. These occur after the evaluation of a full expression (one which is not part of a larger expression), after the evaluation of the first operand of a `&&`, `||`, `?` : or `,` (comma) operator, before a function is called (but after the evaluation of its arguments and the expression denoting the called function), and in certain other places. Other than as expressed by the sequence point rules, the order of evaluation of subexpressions of an expression is not specified. All these rules describe only a partial order rather than a total order, since, for example, if two functions are called within one expression with no sequence point between them, the order in which the functions are called is not specified. However, the standards committee have ruled that function calls do not overlap.

It is not specified when between sequence points modifications to the values of objects take effect. Programs whose behavior depends on this have undefined behavior; the C standard specifies that “Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.”. If a program breaks these rules, the results on any particular implementation are entirely unpredictable.

Examples of code with undefined behavior are `a = a++; a[n] = b[n++]` and `a[i++] = i;`. Some more complicated cases are not diagnosed by this option, and it may give an occasional false positive result, but in general it has been found fairly effective at detecting this sort of problem in programs.

The present implementation of this option only works for C programs. A future implementation may also work for C++ programs.

There is some controversy over the precise meaning of the sequence point rules in subtle cases. Alternative formal definitions may be found in Clive Feather’s “Annex S” <<http://wwwold.dkuug.dk/JTC1/SC22/WG14/www/docs/n925.htm>> and in Michael Norrish’s thesis <<http://www.cl.cam.ac.uk/users/mn200/PhD/thesis-report.ps.gz>>. Other discussions are by Raymond Mak <<http://wwwold.dkuug.dk/JTC1/SC22/WG14/www/docs/n926.htm>> and D. Hugh Redelmeier <<http://wwwold.dkuug.dk/JTC1/SC22/WG14/www/docs/n927.htm>>.

-Wreturn-type

Warn whenever a function is defined with a return-type that defaults to `int`. Also warn about any return statement with no return-value in a function whose return-type is not `void`.

For C++, a function without return type always produces a diagnostic message, even when **-Wno-return-type** is specified. The only exceptions are **main** and functions defined in system headers.

-Wswitch

Warn whenever a `switch` statement has an index of enumerational type and lacks a `case` for one or more of the named codes of that enumeration. (The presence of a `default` label prevents this warning.) `case` labels outside the enumeration range also provoke warnings when this option is used.

-Wtrigraphs

Warn if any trigraphs are encountered that might change the meaning of the program (trigraphs within comments are not warned about).

-Wunused-function

Warn whenever a static function is declared but not defined or a non-`inline` static function is unused.

-Wunused-label

Warn whenever a label is declared but not used.

To suppress this warning use the **unused** attribute.

-Wunused-parameter

Warn whenever a function parameter is unused aside from its declaration.

To suppress this warning use the **unused** attribute.

-Wunused-variable

Warn whenever a local variable or non-constant static variable is unused aside from its declaration

To suppress this warning use the **unused** attribute.

-Wunused-value

Warn whenever a statement computes a result that is explicitly not used.

To suppress this warning cast the expression to **void**.

-Wunused

All the above **-Wunused** options combined.

In order to get a warning about an unused function parameter, you must either specify **-W -Wunused** or separately specify **-Wunused-parameter**.

-Wuninitialized

Warn if an automatic variable is used without first being initialized or if a variable may be clobbered by a `setjmp` call.

These warnings are possible only in optimizing compilation, because they require data flow information that is computed only when optimizing. If you don't specify **-O**, you simply won't get these warnings.

These warnings occur only for variables that are candidates for register allocation. Therefore, they do not occur for a variable that is declared `volatile`, or whose address is taken, or whose size is other than 1, 2, 4 or 8 bytes. Also, they do not occur for structures, unions or arrays, even when they are in registers.

Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed.

These warnings are made optional because GCC is not smart enough to see all the reasons why the code might be correct despite appearing to have an error. Here is one example of how this can happen:

```
{
  int x;
  switch (y)
  {
    case 1: x = 1;
           break;
    case 2: x = 4;
           break;
    case 3: x = 5;
           }
  foo (x);
}
```

If the value of `y` is always 1, 2 or 3, then `x` is always initialized, but GCC doesn't know this. Here is another common case:

```
{
  int save_y;
  if (change_y) save_y = y, y = new_y;
  ...
  if (change_y) y = save_y;
}
```

This has no bug because `save_y` is used only if it is set.

This option also warns when a non-volatile automatic variable might be changed by a call to `longjmp`. These warnings as well are possible only in optimizing compilation.

The compiler sees only the calls to `setjmp`. It cannot know where `longjmp` will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact no problem because `longjmp` cannot in fact be called at the place which would cause a problem.

Some spurious warnings can be avoided if you declare all the functions you use that never return as `noreturn`.

-Wreorder (C++ only)

Warn when the order of member initializers given in the code does not match the order in which they must be executed. For instance:

-Wunknown-pragmas

Warn when a `#pragma` directive is encountered which is not understood by GCC. If this command line option is used, warnings will even be issued for unknown pragmas in system header files. This is not the case if the warnings were only enabled by the **-Wall** command line option.

-Wall

All of the above **-W** options combined. This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.

-Wsystem-headers

Print warning messages for constructs found in system header files. Warnings from system headers are normally suppressed, on the assumption that they usually do not indicate real problems and would only make the compiler output harder to read. Using this command line option tells GCC to emit warnings from system headers as if they occurred in user code. However, note that using **-Wall** in conjunction with this option will *not* warn about unknown pragmas in system headers—for that, **-Wunknown-pragmas** must also be used.

The following **-W...** options are not implied by **-Wall**. Some of them warn about constructions that users generally do not consider questionable, but which occasionally you might wish to check for; others warn about constructions that are necessary or hard to avoid in some cases, and there is no simple way to modify the code to suppress the warning.

-W

Print extra warning messages for these events:

- A function can return either with or without a value. (Falling off the end of the function body is considered returning without a value.) For example, this function would evoke such a warning:

```
foo (a)
{
    if (a > 0)
        return a;
}
```

- An expression-statement or the left-hand side of a comma expression contains no side effects. To suppress the warning, cast the unused expression to void. For example, an expression such as **x[i,j]** will cause a warning, but **x[(void)i,j]** will not.
- An unsigned value is compared against zero with `<` or `<=`.
- A comparison like **x<=y<=z** appears; this is equivalent to **(x<=y ? 1 : 0) <= z**, which is a different interpretation from that of ordinary mathematical notation.

- Storage-class specifiers like `static` are not the first things in a declaration. According to the C Standard, this usage is obsolescent.
- The return type of a function has a type qualifier such as `const`. Such a type qualifier has no effect, since the value returned by a function is not an lvalue. (But don't warn about the GNU extension of `volatile void` return types. That extension will be warned about if **-pedantic** is specified.)
- If **-Wall** or **-Wunused** is also specified, warn about unused arguments.
- A comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. (But don't warn if **-Wno-sign-compare** is also specified.)
- An aggregate has a partly bracketed initializer. For example, the following code would evoke such a warning, because braces are missing around the initializer for `x.h`:

```
struct s { int f, g; };
struct t { struct s h; int i; };
struct t x = { 1, 2, 3 };
```

- An aggregate has an initializer which does not initialize all members. For example, the following code would cause such a warning, because `x.h` would be implicitly initialized to zero:

```
struct s { int f, g, h; };
struct s x = { 3, 4 };
```

-Wfloat-equal

Warn if floating point values are used in equality comparisons.

The idea behind this is that sometimes it is convenient (for the programmer) to consider floating-point values as approximations to infinitely precise real numbers. If you are doing this, then you need to compute (by analysing the code, or in some other way) the maximum or likely maximum error that the computation introduces, and allow for it when performing comparisons (and when producing output, but that's a different problem). In particular, instead of testing for equality, you would check to see whether the two values have ranges that overlap; and this is done with the relational operators, so equality comparisons are probably mistaken.

-Wtraditional (C only)

Warn about certain constructs that behave differently in traditional and ISO C.

- Macro parameters that appear within string literals in the macro body. In traditional C macro replacement takes place within string literals, but does not in ISO C.
- In traditional C, some preprocessor directives did not exist. Traditional preprocessors would only consider a line to be a directive if the `#` appeared in column 1 on the line. Therefore **-Wtraditional** warns about directives that traditional C understands but would ignore because the `#` does not appear as the first character on the line. It also suggests you hide directives like **#pragma** not understood by traditional C by indenting them. Some traditional implementations would not recognise **#elif**, so it suggests avoiding it altogether.
- A function-like macro that appears without arguments.
- The unary plus operator.
- The 'U' integer constant suffix, or the 'F' or 'L' floating point constant suffixes. (Traditional C does support the 'L' suffix on integer constants.) Note, these suffixes appear in macros defined in the system headers of most modern systems, e.g. the `_MIN/_MAX` macros in `limits.h`. Use of these macros can lead to spurious warnings as they do not necessarily reflect whether the code in question is any less portable to traditional C given that suitable backup definitions are provided.

- A function declared external in one block and then used after the end of the block.
- A `switch` statement has an operand of type `long`.
- A non-`static` function declaration follows a `static` one. This construct is not accepted by some traditional C compilers.
- The ISO type of an integer constant has a different width or signedness from its traditional type. This warning is only issued if the base of the constant is ten. I.e. hexadecimal or octal values, which typically represent bit patterns, are not warned about.
- Usage of ISO string concatenation is detected.
- Initialization of automatic aggregates.
- Identifier conflicts with labels. Traditional C lacks a separate namespace for labels.
- Initialization of unions. If the initializer is zero, the warning is omitted. This is done under the assumption that the zero initializer in user code appears conditioned on e.g. `__STDC__` to avoid missing initializer warnings and relies on default initialization to zero in the traditional C case.

-Wundef

Warn if an undefined identifier is evaluated in an `#if` directive.

-Wshadow

Warn whenever a local variable shadows another local variable.

-Wid-clash-len

Warn whenever two distinct identifiers match in the first *len* characters. This may help you prepare a program that will compile with certain obsolete, brain-damaged compilers.

-Wlarger-than-len

Warn whenever an object of larger than *len* bytes is defined.

-Wpointer-arith

Warn about anything that depends on the “size of” a function type or of `void`. GNU C assigns these types a size of 1, for convenience in calculations with `void *` pointers and pointers to functions.

-Wbad-function-cast (C only)

Warn whenever a function call is cast to a non-matching type. For example, warn if `int malloc()` is cast to `anything *`.

-Wcast-qual

Warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a `const char *` is cast to an ordinary `char *`.

-Wcast-align

Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a `char *` is cast to an `int *` on machines where integers can only be accessed at two- or four-byte boundaries.

-Wwrite-strings

Give string constants the type `const char[length]` so that copying the address of one into a non-`const char *` pointer will get a warning. These warnings will help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using `const` in declarations and prototypes. Otherwise, it will just be a nuisance; this is why we did not make **-Wall** request these warnings.

-Wconversion

Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and conversions changing the width or signedness of a fixed point argument except when the same as the default promotion.

Also, warn if a negative integer constant expression is implicitly converted to an unsigned type. For

example, warn about the assignment `x = -1` if `x` is unsigned. But do not warn about explicit casts like `(unsigned) -1`.

-Wsign-compare

Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. This warning is also enabled by **-W**; to get the other warnings of **-W** without this warning, use **-W -Wno-sign-compare**.

-Waggregate-return

Warn if any functions that return structures or unions are defined or called. (In languages where you can return an array, this also elicits a warning.)

-Wstrict-prototypes (C only)

Warn if a function is declared or defined without specifying the argument types. (An old-style function definition is permitted without a warning if preceded by a declaration which specifies the argument types.)

-Wmissing-prototypes (C only)

Warn if a global function is defined without a previous prototype declaration. This warning is issued even if the definition itself provides a prototype. The aim is to detect global functions that fail to be declared in header files.

-Wmissing-declarations

Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files.

-Wmissing-noreturn

Warn about functions which might be candidates for attribute `noreturn`. Note these are only possible candidates, not absolute ones. Care should be taken to manually verify functions actually do not ever return before adding the `noreturn` attribute, otherwise subtle code generation bugs could be introduced. You will not get a warning for `main` in hosted C environments.

-Wmissing-format-attribute

If **-Wformat** is enabled, also warn about functions which might be candidates for `format` attributes. Note these are only possible candidates, not absolute ones. GCC will guess that `format` attributes might be appropriate for any function that calls a function like `vprintf` or `vscanf`, but this might not always be the case, and some functions for which `format` attributes are appropriate may not be detected. This option has no effect unless **-Wformat** is enabled (possibly by **-Wall**).

-Wpacked

Warn if a structure is given the `packed` attribute, but the `packed` attribute has no effect on the layout or size of the structure. Such structures may be mis-aligned for little benefit. For instance, in this code, the variable `f.x` in `struct bar` will be misaligned even though `struct bar` does not itself have the `packed` attribute:

```
struct foo {
    int x;
    char a, b, c, d;
} __attribute__((packed));
struct bar {
    char z;
    struct foo f;
};
```

-Wpadded

Warn if padding is included in a structure, either to align an element of the structure or to align the whole structure. Sometimes when this happens it is possible to rearrange the fields of the structure to reduce the padding and so make the structure smaller.

-Wredundant-decls

Warn if anything is declared more than once in the same scope, even in cases where multiple declaration is valid and changes nothing.

-Wnested-externs (C only)

Warn if an `extern` declaration is encountered within a function.

-Wunreachable-code

Warn if the compiler detects that code will never be executed.

This option is intended to warn when the compiler detects that at least a whole line of source code will never be executed, because some condition is never satisfied or because it is after a procedure that never returns.

It is possible for this option to produce a warning even though there are circumstances under which part of the affected line can be executed, so care should be taken when removing apparently-unreachable code.

For instance, when a function is inlined, a warning may mean that the line is unreachable in only one inlined copy of the function.

This option is not made part of **-Wall** because in a debugging version of a program there is often substantial code which checks correct functioning of the program and is, hopefully, unreachable because the program does work. Another common use of unreachable code is to provide behaviour which is selectable at compile-time.

-Winline

Warn if a function can not be inlined and it was declared as inline.

-Wlong-long

Warn if **long long** type is used. This is default. To inhibit the warning messages, use **-Wno-long-long**. Flags **-Wlong-long** and **-Wno-long-long** are taken into account only when **-pedantic** flag is used.

-Wdisabled-optimization

Warn if a requested optimization pass is disabled. This warning does not generally indicate that there is anything wrong with your code; it merely indicates that GCC's optimizers were unable to handle the code effectively. Often, the problem is that your code is too big or too complex; GCC will refuse to optimize programs when the optimization itself is likely to take inordinate amounts of time.

-Werror

Make all warnings into errors.

Options for Debugging Your Program or GCC

GCC has various special options that are used for debugging either your program or GCC:

- g** Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.

On most systems that use stabs format, **-g** enables use of extra debugging information that only GDB can use; 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 control for certain whether to generate the extra information, use **-gstabs+**, **-gstabs**, **-gxcoff+**, **-gxcoff**, **-gdwarf-1+**, or **-gdwarf-1** (see below).

Unlike most other C compilers, GCC allows you to use **-g** with **-O**. The shortcuts taken by optimized code may occasionally produce surprising results: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values were already at hand; some statements may execute in different places because they were moved out of loops.

Nevertheless it proves possible to debug optimized output. This makes it reasonable to use the

optimizer for programs that might have bugs.

The following options are useful when GCC is generated with the capability for more than one debugging format.

-ggdb

Produce debugging information for use by GDB. This means to use the most expressive format available (DWARF 2, stabs, or the native format if neither of those are supported), including GDB extensions if at all possible.

-gstabs

Produce debugging information in stabs format (if that is supported), without GDB extensions. This is the format used by DBX on most BSD systems. On MIPS, Alpha and System V Release 4 systems this option produces stabs debugging output which is not understood by DBX or SDB. On System V Release 4 systems this option requires the GNU assembler.

-gstabs+

Produce debugging information in stabs format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program.

-gcoff

Produce debugging information in COFF format (if that is supported). This is the format used by SDB on most System V systems prior to System V Release 4.

-gxcoff

Produce debugging information in XCOFF format (if that is supported). This is the format used by the DBX debugger on IBM RS/6000 systems.

-gxcoff+

Produce debugging information in XCOFF format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program, and may cause assemblers other than the GNU assembler (GAS) to fail with an error.

-gdwarf

Produce debugging information in DWARF version 1 format (if that is supported). This is the format used by SDB on most System V Release 4 systems.

-gdwarf+

Produce debugging information in DWARF version 1 format (if that is supported), using GNU extensions understood only by the GNU debugger (GDB). The use of these extensions is likely to make other debuggers crash or refuse to read the program.

-gdwarf-2

Produce debugging information in DWARF version 2 format (if that is supported). This is the format used by DBX on IRIX 6.

-glevel

-ggdblevel

-gstabslevel

-gcofflevel

-gxcofflevel

-gdwarflevel

-gdwarf-2level

Request debugging information and also use *level* to specify how much information. The default level is 2.

Level 1 produces minimal information, enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, but no information about local variables and no line numbers.

Level 3 includes extra information, such as all the macro definitions present in the program. Some debuggers support macro expansion when you use **-g3**.

- p** Generate extra code to write profile information suitable for the analysis program `prof`. You must use this option when compiling the source files you want data about, and you must also use it when linking.

-pg

Generate extra code to write profile information suitable for the analysis program `gprof`. You must use this option when compiling the source files you want data about, and you must also use it when linking.

- a** Generate extra code to write profile information for basic blocks, which will record the number of times each basic block is executed, the basic block start address, and the function name containing the basic block. If **-g** is used, the line number and filename of the start of the basic block will also be recorded. If not overridden by the machine description, the default action is to append to the text file *bb.out*.

This data could be analyzed by a program like `tcov`. Note, however, that the format of the data is not what `tcov` expects. Eventually GNU `gprof` should be extended to process this data.

- Q** Makes the compiler print out each function name as it is compiled, and print some statistics about each pass when it finishes.

-ax

Generate extra code to profile basic blocks. Your executable will produce output that is a superset of that produced when **-a** is used. Additional output is the source and target address of the basic blocks where a jump takes place, the number of times a jump is executed, and (optionally) the complete sequence of basic blocks being executed. The output is appended to file *bb.out*.

You can examine different profiling aspects without recompilation. Your executable will read a list of function names from file *bb.in*. Profiling starts when a function on the list is entered and stops when that invocation is exited. To exclude a function from profiling, prefix its name with '-'. If a function name is not unique, you can disambiguate it by writing it in the form **/path/filename.d:functionname**. Your executable will write the available paths and filenames in file *bb.out*.

Several function names have a special meaning:

Write source, target and frequency of jumps to file *bb.out*. Exclude function calls from frequency count. Include function returns in frequency count. Write the sequence of basic blocks executed to file *bbtrace.gz*. The file will be compressed using the program **gzip**, which must exist in your **PATH**. On systems without the **popen** function, the file will be named *bbtrace* and will not be compressed.

Profiling for even a few seconds on these systems will produce a very large file. Note: `__bb_hidecall__` and `__bb_showret__` will not affect the sequence written to *bbtrace.gz*.

Here's a short example using different profiling parameters in file *bb.in*. Assume function `foo` consists of basic blocks 1 and 2 and is called twice from block 3 of function `main`. After the calls, block 3 transfers control to block 4 of `main`.

With `__bb_trace__` and `main` contained in file *bb.in*, the following sequence of blocks is written to file *bbtrace.gz*: 0 3 1 2 1 2 4. The return from block 2 to block 3 is not shown, because the return is to a point inside the block and not to the top. The block address 0 always indicates, that control is transferred to the trace from somewhere outside the observed functions. With **-foo** added to *bb.in*, the blocks of function `foo` are removed from the trace, so only 0 3 4 remains.

With `__bb_jumps__` and `main` contained in file *bb.in*, jump frequencies will be written to file *bb.out*. The frequencies are obtained by constructing a trace of blocks and incrementing a counter for every neighbouring pair of blocks in the trace. The trace 0 3 1 2 1 2 4 displays the following frequencies:

```

Jump from block 0x0 to block 0x3 executed 1 time(s)
Jump from block 0x3 to block 0x1 executed 1 time(s)
Jump from block 0x1 to block 0x2 executed 2 time(s)
Jump from block 0x2 to block 0x1 executed 1 time(s)
Jump from block 0x2 to block 0x4 executed 1 time(s)

```

With `__bb_hidecall__`, control transfer due to call instructions is removed from the trace, that is the trace is cut into three parts: 0 3 4, 0 1 2 and 0 1 2. With `__bb_showret__`, control transfer due to return instructions is added to the trace. The trace becomes: 0 3 1 2 3 1 2 3 4. Note, that this trace is not the same, as the sequence written to *bbtrace.gz*. It is solely used for counting jump frequencies.

-fprofile-arcs

Instrument *arcs* during compilation. For each function of your program, GCC creates a program flow graph, then finds a spanning tree for the graph. Only arcs that are not on the spanning tree have to be instrumented: the compiler adds code to count the number of times that these arcs are executed. When an arc is the only exit or only entrance to a block, the instrumentation code can be added to the block; otherwise, a new basic block must be created to hold the instrumentation code.

Since not every arc in the program must be instrumented, programs compiled with this option run faster than programs compiled with `-a`, which adds instrumentation code to every basic block in the program. The tradeoff: since *gcov* does not have execution counts for all branches, it must start with the execution counts for the instrumented branches, and then iterate over the program flow graph until the entire graph has been solved. Hence, *gcov* runs a little more slowly than a program which uses information from `-a`.

-fprofile-arcs also makes it possible to estimate branch probabilities, and to calculate basic block execution counts. In general, basic block execution counts do not give enough information to estimate all branch probabilities. When the compiled program exits, it saves the arc execution counts to a file called *sourcename.da*. Use the compiler option **-fbranch-probabilities** when recompiling, to optimize using estimated branch probabilities.

-ftest-coverage

Create data files for the *gcov* code-coverage utility. The data file names begin with the name of your source file:

sourcename.bb

A mapping from basic blocks to line numbers, which *gcov* uses to associate basic block execution counts with line numbers.

sourcename.bbg

A list of all arcs in the program flow graph. This allows *gcov* to reconstruct the program flow graph, so that it can compute all basic block and arc execution counts from the information in the *sourcename.da* file (this last file is the output from **-fprofile-arcs**).

-dletters

Says to make debugging dumps during compilation at times specified by *letters*. This is used for debugging the compiler. The file names for most of the dumps are made by appending a pass number and a word to the source file name (e.g. *foo.c.00.rtl* or *foo.c.01.sibling*). Here are the possible letters for use in *letters*, and their meanings:

- A** Annotate the assembler output with miscellaneous debugging information.
- b** Dump after computing branch probabilities, to *file.11.bp*.
- B** Dump after block reordering, to *file.26.bbro*.
- c** Dump after instruction combination, to the file *file.14.combine*.
- C** Dump after the first if conversion, to the file *file.15.ce*.

- d** Dump after delayed branch scheduling, to *file.29.dbr*.
- D** Dump all macro definitions, at the end of preprocessing, in addition to normal output.
- e** Dump after SSA optimizations, to *file.05.ssa* and *file.06.ussa*.
- E** Dump after the second if conversion, to *file.24.ce2*.
- f** Dump after life analysis, to *file.13.life*.
- F** Dump after purging ADDRESSOF codes, to *file.04.addressof*.
- g** Dump after global register allocation, to *file.19.greg*.
- o** Dump after post-reload CSE and other optimizations, to *file.20.postreload*.
- G** Dump after GCSE, to *file.08.gcse*.
- i** Dump after sibling call optimizations, to *file.01.sibling*.
- j** Dump after the first jump optimization, to *file.02.jump*.
- J** Dump after the last jump optimization, to *file.27.jump2*.
- k** Dump after conversion from registers to stack, to *file.29.stack*.
- l** Dump after local register allocation, to *file.18.lreg*.
- L** Dump after loop optimization, to *file.09.loop*.
- M** Dump after performing the machine dependent reorganisation pass, to *file.28.mach*.
- n** Dump after register renumbering, to *file.23.rnreg*.
- N** Dump after the register move pass, to *file.16.regmove*.
- r** Dump after RTL generation, to *file.00.rtl*.
- R** Dump after the second instruction scheduling pass, to *file.25.sched2*.
- s** Dump after CSE (including the jump optimization that sometimes follows CSE), to *file.03.cse*.
- S** Dump after the first instruction scheduling pass, to *file.17.sched*.
- t** Dump after the second CSE pass (including the jump optimization that sometimes follows CSE), to *file.10.cse2*.
- w** Dump after the second flow pass, to *file.21.flow2*.
- X** Dump after dead code elimination, to *file.06.dce*.
- z** Dump after the peephole pass, to *file.22.peephole2*.
- a** Produce all the dumps listed above.
- m** Print statistics on memory usage, at the end of the run, to standard error.
- p** Annotate the assembler output with a comment indicating which pattern and alternative was used. The length of each instruction is also printed.
- P** Dump the RTL in the assembler output as a comment before each instruction. Also turns on **-dp** annotation.
- v** For each of the other indicated dump files (except for *file.00.rtl*), dump a representation of the control flow graph suitable for viewing with VCG to *file.pass.vcg*.
- x** Just generate RTL for a function instead of compiling it. Usually used with **r**.
- y** Dump debugging information during parsing, to standard error.

-fdump-unnumbered

When doing debugging dumps (see **-d** option above), suppress instruction numbers and line number note output. This makes it more feasible to use diff on debugging dumps for compiler invocations with different options, in particular with and without **-g**.

-fdump-translation-unit=file (C and C++ only)

Dump a representation of the tree structure for the entire translation unit to *file*.

-fdump-class_layout=file (C++ only)**-fdump-class_layout (C++ only)**

Dump a representation of each class's hierarchy to *file*, or `stderr` if not specified.

-fpretend-float

When running a cross-compiler, pretend that the target machine uses the same floating point format as the host machine. This causes incorrect output of the actual floating constants, but the actual instruction sequence will probably be the same as GCC would make when running on the target machine.

-save-temps

Store the usual “temporary” intermediate files permanently; place them in the current directory and name them based on the source file. Thus, compiling *foo.c* with **-c -save-temps** would produce files *foo.i* and *foo.s*, as well as *foo.o*. This creates a preprocessed *foo.i* output file even though the compiler now normally uses an integrated preprocessor.

-time

Report the CPU time taken by each subprocess in the compilation sequence. For C source files, this is the compiler proper and assembler (plus the linker if linking is done). The output looks like this:

```
# cc1 0.12 0.01
# as 0.00 0.01
```

The first number on each line is the “user time,” that is time spent executing the program itself. The second number is “system time,” time spent executing operating system routines on behalf of the program. Both numbers are in seconds.

-print-file-name=library

Print the full absolute name of the library file *library* that would be used when linking—and don't do anything else. With this option, GCC does not compile or link anything; it just prints the file name.

-print-prog-name=program

Like **-print-file-name**, but searches for a program such as **cpp**.

-print-libgcc-file-name

Same as **-print-file-name=libgcc.a**.

This is useful when you use **-nostdlib** or **-nodefaultlibs** but you do want to link with *libgcc.a*. You can do

```
gcc -nostdlib I<files>... 'gcc -print-libgcc-file-name'
```

-print-search-dirs

Print the name of the configured installation directory and a list of program and library directories gcc will search—and don't do anything else.

This is useful when gcc prints the error message **installation problem, cannot exec cpp0: No such file or directory**. To resolve this you either need to put *cpp0* and the other compiler components where gcc expects to find them, or you can set the environment variable **GCC_EXEC_PREFIX** to the directory where you installed them. Don't forget the trailing `'/'`.

Options That Control Optimization

These options control various sorts of optimizations:

-O**-O1**

Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

Without **-O**, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.

Without **-O**, the compiler only allocates variables declared `register` in registers. The resulting compiled code is a little worse than produced by PCC without **-O**.

With **-O**, the compiler tries to reduce code size and execution time.

When you specify **-O**, the compiler turns on **-fthread-jumps** and **-fdefer-pop** on all machines. The compiler turns on **-fdelayed-branch** on machines that have delay slots, and **-fomit-frame-pointer** on machines that can support debugging even without a frame pointer. On some machines the compiler also turns on other flags.

-O2

Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. The compiler does not perform loop unrolling or function inlining when you specify **-O2**. As compared to **-O**, this option increases both compilation time and the performance of the generated code.

-O2 turns on all optional optimizations except for loop unrolling, function inlining, and register renaming. It also turns on the **-fforce-mem** option on all machines and frame pointer elimination on machines where doing so does not interfere with debugging.

-O3

Optimize yet more. **-O3** turns on all optimizations specified by **-O2** and also turns on the **-finline-functions** and **-frename-registers** options.

-O0

Do not optimize.

-Os

Optimize for size. **-Os** enables all **-O2** optimizations that do not typically increase code size. It also performs further optimizations designed to reduce code size.

If you use multiple **-O** options, with or without level numbers, the last such option is the one that is effective.

Options of the form **-f*flag*** specify machine-independent flags. Most flags have both positive and negative forms; the negative form of **-ffoo** would be **-fno-foo**. In the table below, only one of the forms is listed—the one which is not the default. You can figure out the other form by either removing **no-** or adding it.

-ffloat-store

Do not store floating point variables in registers, and inhibit other options that might change whether a floating point value is taken from a register or memory.

This option prevents undesirable excess precision on machines such as the 68000 where the floating registers (of the 68881) keep more precision than a `double` is supposed to have. Similarly for the x86 architecture. For most programs, the excess precision does only good, but a few programs rely on the precise definition of IEEE floating point. Use **-ffloat-store** for such programs, after modifying them to store all pertinent intermediate computations into variables.

-fno-default-inline

Do not make member functions inline by default merely because they are defined inside the class scope (C++ only). Otherwise, when you specify **-O**, member functions defined inside class scope are compiled inline by default; i.e., you don't need to add **inline** in front of the member function name.

-fno-defer-pop

Always pop the arguments to each function call as soon as that function returns. For machines which must pop arguments after a function call, the compiler normally lets arguments accumulate on the

stack for several function calls and pops them all at once.

-fforce-mem

Force memory operands to be copied into registers before doing arithmetic on them. This produces better code by making all memory references potential common subexpressions. When they are not common subexpressions, instruction combination should eliminate the separate register-load. The **-O2** option turns on this option.

-fforce-addr

Force memory address constants to be copied into registers before doing arithmetic on them. This may produce better code just as **-fforce-mem** may.

-fomit-frame-pointer

Don't keep the frame pointer in a register for functions that don't need one. This avoids the instructions to save, set up and restore frame pointers; it also makes an extra register available in many functions. **It also makes debugging impossible on some machines.**

On some machines, such as the Vax, this flag has no effect, because the standard calling sequence automatically handles the frame pointer and nothing is saved by pretending it doesn't exist. The machine-description macro `FRAME_POINTER_REQUIRED` controls whether a target machine supports this flag.

-foptimize-sibling-calls

Optimize sibling and tail recursive calls.

-ftrapv

This option generates traps for signed overflow on addition, subtraction, multiplication operations.

-fno-inline

Don't pay attention to the `inline` keyword. Normally this option is used to keep the compiler from expanding any functions inline. Note that if you are not optimizing, no functions can be expanded inline.

-finline-functions

Integrate all simple functions into their callers. The compiler heuristically decides which functions are simple enough to be worth integrating in this way.

If all calls to a given function are integrated, and the function is declared `static`, then the function is normally not output as assembler code in its own right.

-finline-limit=*n*

By default, gcc limits the size of functions that can be inlined. This flag allows the control of this limit for functions that are explicitly marked as inline (ie marked with the `inline` keyword or defined within the class definition in c++). *n* is the size of functions that can be inlined in number of pseudo instructions (not counting parameter handling). The default value of *n* is 10000. Increasing this value can result in more inlined code at the cost of compilation time and memory consumption. Decreasing usually makes the compilation faster and less code will be inlined (which presumably means slower programs). This option is particularly useful for programs that use inlining heavily such as those based on recursive templates with c++.

Note: pseudo instruction represents, in this particular context, an abstract measurement of function's size. In no way, it represents a count of assembly instructions and as such its exact meaning might change from one release to another.

-fkeep-inline-functions

Even if all calls to a given function are integrated, and the function is declared `static`, nevertheless output a separate run-time callable version of the function. This switch does not affect extern inline functions.

-fkeep-static-consts

Emit variables declared `static const` when optimization isn't turned on, even if the variables aren't referenced.

GCC enables this option by default. If you want to force the compiler to check if the variable was referenced, regardless of whether or not optimization is turned on, use the **-fno-keep-static-consts** option.

-fno-function-cse

Do not put function addresses in registers; make each instruction that calls a constant function contain the function's address explicitly.

This option results in less efficient code, but some strange hacks that alter the assembler output may be confused by the optimizations performed when this option is not used.

-ffast-math

This option allows GCC to violate some ISO or IEEE rules and/or specifications in the interest of optimizing code for speed. For example, it allows the compiler to assume arguments to the `sqrt` function are non-negative numbers and that no floating-point values are NaNs.

This option should never be turned on by any **-O** option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules/specifications for math functions.

-fno-math-errno

Do not set `ERRNO` after calling math functions that are executed with a single instruction, e.g., `sqrt`. A program that relies on IEEE exceptions for math error handling may want to use this flag for speed while maintaining IEEE arithmetic compatibility.

The default is **-fmath-errno**. The **-ffast-math** option sets **-fno-math-errno**.

The following options control specific optimizations. The **-O2** option turns on all of these optimizations except **-funroll-loops** and **-funroll-all-loops**. On most machines, the **-O** option turns on the **-fthread-jumps** and **-fdelayed-branch** options, but specific machines may handle it differently.

You can use the following flags in the rare cases when "fine-tuning" of optimizations to be performed is desired.

-fstrength-reduce

Perform the optimizations of loop strength reduction and elimination of iteration variables.

-fthread-jumps

Perform optimizations where we check to see if a jump branches to a location where another comparison subsumed by the first is found. If so, the first branch is redirected to either the destination of the second branch or a point immediately following it, depending on whether the condition is known to be true or false.

-fcse-follow-jumps

In common subexpression elimination, 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 is similar to **-fcse-follow-jumps**, but causes CSE to follow jumps which conditionally skip over blocks. When CSE encounters a simple `if` statement with no `else` clause, **-fcse-skip-blocks** causes CSE to follow the jump around the body of the `if`.

-frerun-cse-after-loop

Re-run common subexpression elimination after loop optimizations has been performed.

-frerun-loop-opt

Run the loop optimizer twice.

-fgcse

Perform a global common subexpression elimination pass. This pass also performs global constant and copy propagation.

-fdelete-null-pointer-checks

Use global dataflow analysis to identify and eliminate useless null pointer checks. Programs which rely on NULL pointer dereferences *not* halting the program may not work properly with this option. Use `-fno-delete-null-pointer-checks` to disable this optimizing for programs which depend on that behavior.

-fexpensive-optimizations

Perform a number of minor optimizations that are relatively expensive.

-foptimize-register-move**-fregmove**

Attempt to reassign register numbers in move instructions and as operands of other simple instructions in order to maximize the amount of register tying. This is especially helpful on machines with two-operand instructions. GCC enables this optimization by default with `-O2` or higher.

Note `-fregmove` and `-foptimize-register-move` are the same optimization.

-fdelayed-branch

If supported for the target machine, attempt to reorder instructions to exploit instruction slots available after delayed branch instructions.

-fschedule-insns

If supported for the target machine, attempt to reorder instructions to eliminate execution stalls due to required data being unavailable. This helps machines that have slow floating point or memory load instructions by allowing other instructions to be issued until the result of the load or floating point instruction is required.

-fschedule-insns2

Similar to `-fschedule-insns`, but requests an additional pass of instruction scheduling after register allocation has been done. This is especially useful on machines with a relatively small number of registers and where memory load instructions take more than one cycle.

-ffunction-sections**-fdata-sections**

Place each function or data item into its own section in the output file if the target supports arbitrary sections. The name of the function or the name of the data item determines the section's name in the output file.

Use these options on systems where the linker can perform optimizations to improve locality of reference in the instruction space. HPPA processors running HP-UX and Sparc processors running Solaris 2 have linkers with such optimizations. Other systems using the ELF object format as well as AIX may have these optimizations in the future.

Only use these options when there are significant benefits from doing so. When you specify these options, the assembler and linker will create larger object and executable files and will also be slower. You will not be able to use `gprof` on all systems if you specify this option and you may have problems with debugging if you specify both this option and `-g`.

-fcaller-saves

Enable values to be allocated in registers that will be clobbered by function calls, by emitting extra instructions to save and restore the registers around such calls. Such allocation is done only when it seems to result in better code than would otherwise be produced.

This option is always enabled by default on certain machines, usually those which have no call-preserved registers to use instead.

For all machines, optimization level 2 and higher enables this flag by default.

-funroll-loops

Perform the optimization of loop unrolling. This is only done for loops whose number of iterations can be determined at compile time or run time. **-funroll-loops** implies both **-fstrength-reduce** and **-frerun-cse-after-loop**.

-funroll-all-loops

Perform the optimization of loop unrolling. This is done for all loops and usually makes programs run more slowly. **-funroll-all-loops** implies **-fstrength-reduce** as well as **-frerun-cse-after-loop**.

-fmove-all-movables

Forces all invariant computations in loops to be moved outside the loop.

-freduce-all-givs

Forces all general-induction variables in loops to be strength-reduced.

Note: When compiling programs written in Fortran, **-fmove-all-movables** and **-freduce-all-givs** are enabled by default when you use the optimizer.

These options may generate better or worse code; results are highly dependent on the structure of loops within the source code.

These two options are intended to be removed someday, once they have helped determine the efficacy of various approaches to improving loop optimizations.

Please let us (<gcc@gcc.gnu.org> and <fortran@gnu.org>) know how use of these options affects the performance of your production code. We're very interested in code that runs *slower* when these options are *enabled*.

-fno-peephole

Disable any machine-specific peephole optimizations.

-fbranch-probabilities

After running a program compiled with **-fprofile-arcs**, you can compile it a second time using **-fbranch-probabilities**, to improve optimizations based on guessing the path a branch might take.

-fno-guess-branch-probability

Sometimes gcc will opt to guess branch probabilities when none are available from either profile directed feedback (**-fprofile-arcs**) or **__builtin_expect**. In a hard real-time system, people don't want different runs of the compiler to produce code that has different behavior; minimizing non-determinism is of paramount import. This switch allows users to reduce non-determinism, possibly at the expense of inferior optimization.

-fstrict-aliasing

Allows the compiler to assume the strictest aliasing rules applicable to the language being compiled. For C (and C++), this activates optimizations based on the type of expressions. In particular, an object of one type is assumed never to reside at the same address as an object of a different type, unless the types are almost the same. For example, an `unsigned int` can alias an `int`, but not a `void*` or a `double`. A character type may alias any other type.

Pay special attention to code like this:

```
union a_union {
    int i;
    double d;
};

int f() {
    a_union t;
    t.d = 3.0;
    return t.i;
}
```

The practice of reading from a different union member than the one most recently written to (called “type-punning”) is common. Even with **-fstrict-aliasing**, type-punning is allowed, provided the memory is accessed through the union type. So, the code above will work as expected. However, this code might not:

```
int f() {
    a_union t;
    int* ip;
    t.d = 3.0;
    ip = &t.i;
    return *ip;
}
```

-falign-functions

-falign-functions=*n*

Align the start of functions to the next power-of-two greater than *n*, skipping up to *n* bytes. For instance, **-falign-functions=32** aligns functions to the next 32-byte boundary, but **-falign-functions=24** would align to the next 32-byte boundary only if this can be done by skipping 23 bytes or less.

-fno-align-functions and **-falign-functions=1** are equivalent and mean that functions will not be aligned.

Some assemblers only support this flag when *n* is a power of two; in that case, it is rounded up.

If *n* is not specified, use a machine-dependent default.

-falign-labels

-falign-labels=*n*

Align all branch targets to a power-of-two boundary, skipping up to *n* bytes like **-falign-functions**. This option can easily make code slower, because it must insert dummy operations for when the branch target is reached in the usual flow of the code.

If **-falign-loops** or **-falign-jumps** are applicable and are greater than this value, then their values are used instead.

If *n* is not specified, use a machine-dependent default which is very likely to be **1**, meaning no alignment.

-falign-loops

-falign-loops=*n*

Align loops to a power-of-two boundary, skipping up to *n* bytes like **-falign-functions**. The hope is that the loop will be executed many times, which will make up for any execution of the dummy operations.

If *n* is not specified, use a machine-dependent default.

-falign-jumps

-falign-jumps=*n*

Align branch targets to a power-of-two boundary, for branch targets where the targets can only be reached by jumping, skipping up to *n* bytes like **-falign-functions**. In this case, no dummy operations need be executed.

If *n* is not specified, use a machine-dependent default.

-fssa

Perform optimizations in static single assignment form. Each function’s flow graph is translated into SSA form, optimizations are performed, and the flow graph is translated back from SSA form. User’s should not specify this option, since it is not yet ready for production use.

-fdce

Perform dead-code elimination in SSA form. Requires **-fssa**. Like **-fssa**, this is an experimental feature.

-fsingle-precision-constant

Treat floating point constant as single precision constant instead of implicitly converting it to double precision constant.

-frename-registers

Attempt to avoid false dependencies in scheduled code by making use of registers left over after register allocation. This optimization will most benefit processors with lots of registers. It can, however, make debugging impossible, since variables will no longer stay in a “home register”.

—param name=value

In some places, GCC uses various constants to control the amount of optimization that is done. For example, GCC will not inline functions that contain more than a certain number of instructions. You can control some of these constants on the command-line using the **—param** option.

In each case, the *value* is a integer. The allowable choices for *name* are given in the following table:

max-inline-insns

If an function contains more than this many instructions, it will not be inlined. This option is precisely equivalent to **-finline-limit**.

Options Controlling the Preprocessor

These options control the C preprocessor, which is run on each C source file before actual compilation.

If you use the **-E** option, nothing is done except preprocessing. Some of these options make sense only together with **-E** because they cause the preprocessor output to be unsuitable for actual compilation.

-include file

Process *file* as input before processing the regular input file. In effect, the contents of *file* are compiled first. Any **-D** and **-U** options on the command line are always processed before **-include file**, regardless of the order in which they are written. All the **-include** and **-imacros** options are processed in the order in which they are written.

-imacros file

Process *file* as input, discarding the resulting output, before processing the regular input file. Because the output generated from *file* is discarded, the only effect of **-imacros file** is to make the macros defined in *file* available for use in the main input. All the **-include** and **-imacros** options are processed in the order in which they are written.

-idirafter dir

Add the directory *dir* to the second include path. The directories on the second include path are searched when a header file is not found in any of the directories in the main include path (the one that **-I** adds to).

-iprefix prefix

Specify *prefix* as the prefix for subsequent **-iwithprefix** options.

-iwithprefix dir

Add a directory to the second include path. The directory’s name is made by concatenating *prefix* and *dir*, where *prefix* was specified previously with **-iprefix**. If you have not specified a prefix yet, the directory containing the installed passes of the compiler is used as the default.

-iwithprefixbefore dir

Add a directory to the main include path. The directory’s name is made by concatenating *prefix* and *dir*, as in the case of **-iwithprefix**.

-isystem *dir*

Add a directory to the beginning of the second include path, marking it as a system directory, so that it gets the same special treatment as is applied to the standard system directories.

-nostdinc

Do not search the standard system directories for header files. Only the directories you have specified with **-I** options (and the current directory, if appropriate) are searched.

By using both **-nostdinc** and **-I-**, you can limit the include-file search path to only those directories you specify explicitly.

-remap

When searching for a header file in a directory, remap file names if a file named *header.gcc* exists in that directory. This can be used to work around limitations of file systems with file name restrictions. The *header.gcc* file should contain a series of lines with two tokens on each line: the first token is the name to map, and the second token is the actual name to use.

-undef

Do not predefine any nonstandard macros. (Including architecture flags).

-E Run only the C preprocessor. Preprocess all the C source files specified and output the results to standard output or to the specified output file.

-C Tell the preprocessor not to discard comments. Used with the **-E** option.

-P Tell the preprocessor not to generate **#line** directives. Used with the **-E** option.

-M

Instead of outputting the result of preprocessing, output a rule suitable for *make* describing the dependencies of the main source file. The preprocessor outputs one *make* rule containing the object file name for that source file, a colon, and the names of all the included files. Unless overridden explicitly, the object file name consists of the basename of the source file with any suffix replaced with object file suffix. If there are many included files then the rule is split into several lines using **\-newline**.

-M implies **-E**.

-MM

Like **-M**, but mention only the files included with **#include "file"**. System header files included with **#include <file>** are omitted.

-MD

Like **-M** but the dependency information is written to a file rather than stdout. *gcc* will use the same file name and directory as the object file, but with the suffix **".d"** instead.

This is in addition to compiling the main file as specified — **-MD** does not inhibit ordinary compilation the way **-M** does, unless you also specify **-MG**.

With Mach, you can use the utility *md* to merge multiple dependency files into a single dependency file suitable for using with the **make** command.

-MMD

Like **-MD** except mention only user header files, not system **-header** files.

-MF *file*

When used with **-M** or **-MM**, specifies a file to write the dependencies to. This allows the preprocessor to write the preprocessed file to stdout normally. If no **-MF** switch is given, CPP sends the rules to stdout and suppresses normal preprocessed output.

Another way to specify output of a *make* rule is by setting the environment variable **DEPENDENCIES_OUTPUT**.

-MG

When used with **-M** or **-MM**, **-MG** says to treat missing header files as generated files and assume they live in the same directory as the source file. It suppresses preprocessed output, as a missing

header file is ordinarily an error.

This feature is used in automatic updating of makefiles.

-MP

This option instructs CPP to add a phony target for each dependency other than the main file, causing each to depend on nothing. These dummy rules work around errors `make` gives if you remove header files without updating the `Makefile` to match.

This is typical output:-

```
/tmp/test.o: /tmp/test.c /tmp/test.h
/tmp/test.h:
```

-MQ target

-MT target

By default CPP uses the main file name, including any path, and appends the object suffix, normally “.o”, to it to obtain the name of the target for dependency generation. With **-MT** you can specify a target yourself, overriding the default one.

If you want multiple targets, you can specify them as a single argument to **-MT**, or use multiple **-MT** options.

The targets you specify are output in the order they appear on the command line. **-MQ** is identical to **-MT**, except that the target name is quoted for `Make`, but with **-MT** it isn't. For example, **-MT** `'$(objpfx)foo.o'` gives

```
$(objpfx)foo.o: /tmp/foo.c
```

but **-MQ** `'$(objpfx)foo.o'` gives

```
$$$(objpfx)foo.o: /tmp/foo.c
```

The default target is automatically quoted, as if it were given with **-MQ**.

-H Print the name of each header file used, in addition to other normal activities.

-Aquestion(answer)

Assert the answer *answer* for *question*, in case it is tested with a preprocessing conditional such as **#if** *#question(answer)*. **-A-** disables the standard assertions that normally describe the target machine.

-Dmacro

Define macro *macro* with the string **1** as its definition.

-Dmacro=defn

Define macro *macro* as *defn*. All instances of **-D** on the command line are processed before any **-U** options.

Any **-D** and **-U** options on the command line are processed in order, and always before **-imacros file**, regardless of the order in which they are written.

-Umacro

Undefine macro *macro*. **-U** options are evaluated after all **-D** options, but before any **-include** and **-imacros** options.

Any **-D** and **-U** options on the command line are processed in order, and always before **-imacros file**, regardless of the order in which they are written.

-dM

Tell the preprocessor to output only a list of the macro definitions that are in effect at the end of pre-processing. Used with the **-E** option.

-dD

Tell the preprocessing to pass all macro definitions into the output, in their proper sequence in the rest of the output.

-dN

Like **-dD** except that the macro arguments and contents are omitted. Only **#define** *name* is included in the output.

-dI

Output **#include** directives in addition to the result of preprocessing.

-trigraphs

Process ISO standard trigraph sequences. These are three-character sequences, all starting with **??**, that are defined by ISO C to stand for single characters. For example, **??/** stands for ****, so **'??/n'** is a character constant for a newline. By default, GCC ignores trigraphs, but in standard-conforming modes it converts them. See the **-std** and **-ansi** options.

The nine trigraph sequences are

??(-> [

??) -> **]**

??<

-> **{**

??>

-> **}**

??=

-> **#**

??/ -> ****

??' -> **^**

??! -> **|**

??- -> **~**

Trigraph support is not popular, so many compilers do not implement it properly. Portable code should not rely on trigraphs being either converted or ignored.

-Wp,*option*

Pass *option* as an option to the preprocessor. If *option* contains commas, it is split into multiple options at the commas.

Passing Options to the Assembler

You can pass options to the assembler.

-Wa,*option*

Pass *option* as an option to the assembler. If *option* contains commas, it is split into multiple options at the commas.

Options for Linking

These options come into play when the compiler links object files into an executable output file. They are meaningless if the compiler is not doing a link step.

object-file-name

A file name that does not end in a special recognized suffix is considered to name an object file or library. (Object files are distinguished from libraries by the linker according to the file contents.) If linking is done, these object files are used as input to the linker.

-c

-S

-E If any of these options is used, then the linker is not run, and object file names should not be used as arguments.

-l*library*

Search the library named *library* when linking.

It makes a difference where in the command you write this option; the linker searches processes libraries and object files in the order they are specified. Thus, **foo.o -lz bar.o** searches library **z** after file *foo.o* but before *bar.o*. If *bar.o* refers to functions in **z**, those functions may not be loaded.

The linker searches a standard list of directories for the library, which is actually a file named *liblibrary.a*. The linker then uses this file as if it had been specified precisely by name.

The directories searched include several standard system directories plus any that you specify with **-L**.

Normally the files found this way are library files—archive files whose members are object files. The linker handles an archive file by scanning through it for members which define symbols that have so far been referenced but not defined. But if the file that is found is an ordinary object file, it is linked in the usual fashion. The only difference between using an **-l** option and specifying a file name is that **-l** surrounds *library* with **lib** and **.a** and searches several directories.

-lobjc

You need this special case of the **-l** option in order to link an Objective C program.

-nostartfiles

Do not use the standard system startup files when linking. The standard system libraries are used normally, unless **-nostdlib** or **-nodefaultlibs** is used.

-nodefaultlibs

Do not use the standard system libraries when linking. Only the libraries you specify will be passed to the linker. The standard startup files are used normally, unless **-nostartfiles** is used. The compiler may generate calls to `memcpy`, `memset`, and `memcpy` for System V (and ISO C) environments or to `bcopy` and `bzero` for BSD environments. These entries are usually resolved by entries in `libc`. These entry points should be supplied through some other mechanism when this option is specified.

-nostdlib

Do not use the standard system startup files or libraries when linking. No startup files and only the libraries you specify will be passed to the linker. The compiler may generate calls to `memcpy`, `memset`, and `memcpy` for System V (and ISO C) environments or to `bcopy` and `bzero` for BSD environments. These entries are usually resolved by entries in `libc`. These entry points should be supplied through some other mechanism when this option is specified.

One of the standard libraries bypassed by **-nostdlib** and **-nodefaultlibs** is *libgcc.a*, a library of internal subroutines that GCC uses to overcome shortcomings of particular machines, or special needs for some languages.

In most cases, you need *libgcc.a* even when you want to avoid other standard libraries. In other words, when you specify **-nostdlib** or **-nodefaultlibs** you should usually specify **-lgcc** as well. This ensures that you have no unresolved references to internal GCC library subroutines. (For example, `__main`, used to ensure C++ constructors will be called.)

-s Remove all symbol table and relocation information from the executable.

-static

On systems that support dynamic linking, this prevents linking with the shared libraries. On other systems, this option has no effect.

-shared

Produce a shared object which can then be linked with other objects to form an executable. Not all systems support this option. For predictable results, you must also specify the same set of options that

were used to generate code (**-fpic**, **-fPIC**, or model suboptions) when you specify this option.[1]

-shared-libgcc

-static-libgcc

On systems that provide *libgcc* as a shared library, these options force the use of either the shared or static version respectively. If no shared version of *libgcc* was built when the compiler was configured, these options have no effect.

There are several situations in which an application should use the shared *libgcc* instead of the static version. The most common of these is when the application wishes to throw and catch exceptions across different shared libraries. In that case, each of the libraries as well as the application itself should use the shared *libgcc*.

Therefore, whenever you specify the **-shared** option, the GCC driver automatically adds **-shared-libgcc**, unless you explicitly specify **-static-libgcc**. The G++ driver automatically adds **-shared-libgcc** when you build a main executable as well because for C++ programs that is typically the right thing to do. (Exception-handling will not work reliably otherwise.)

However, when linking a main executable written in C, you must explicitly say **-shared-libgcc** if you want to use the shared *libgcc*.

-symbolic

Bind references to global symbols when building a shared object. Warn about any unresolved references (unless overridden by the link editor option **-Xlinker -z -Xlinker defs**). Only a few systems support this option.

-Xlinker option

Pass *option* as an option to the linker. You can use this to supply system-specific linker options which GCC does not know how to recognize.

If you want to pass an option that takes an argument, you must use **-Xlinker** twice, once for the option and once for the argument. For example, to pass **-assert definitions**, you must write **-Xlinker -assert -Xlinker definitions**. It does not work to write **-Xlinker "-assert definitions"**, because this passes the entire string as a single argument, which is not what the linker expects.

-Wl,option

Pass *option* as an option to the linker. If *option* contains commas, it is split into multiple options at the commas.

-u symbol

Pretend the symbol *symbol* is undefined, to force linking of library modules to define it. You can use **-u** multiple times with different symbols to force loading of additional library modules.

Options for Directory Search

These options specify directories to search for header files, for libraries and for parts of the compiler:

-Idir

Add the directory *dir* to the head of the list of directories to be searched for header files. This can be used to override a system header file, substituting your own version, since these directories are searched before the system header file directories. However, you should not use this option to add directories that contain vendor-supplied system header files (use **-isystem** for that). If you use more than one **-I** option, the directories are scanned in left-to-right order; the standard system directories come after.

-I- Any directories you specify with **-I** options before the **-I-** option are searched only for the case of **#include "file"**; they are not searched for **#include <file>**.

If additional directories are specified with **-I** options after the **-I-**, these directories are searched for all **#include** directives. (Ordinarily *all* **-I** directories are used this way.)

In addition, the **-I-** option inhibits the use of the current directory (where the current input file came

from) as the first search directory for **#include "file"**. There is no way to override this effect of **-I-**. With **-I**, you can specify searching the directory which was current when the compiler was invoked. That is not exactly the same as what the preprocessor does by default, but it is often satisfactory.

-I- does not inhibit the use of the standard system directories for header files. Thus, **-I-** and **-nostd-inc** are independent.

-Ldir

Add directory *dir* to the list of directories to be searched for **-I**.

-Bprefix

This option specifies where to find the executables, libraries, include files, and data files of the compiler itself.

The compiler driver program runs one or more of the subprograms *cpp*, *cc1*, *as* and *ld*. It tries *prefix* as a prefix for each program it tries to run, both with and without *machineversion/*.

For each subprogram to be run, the compiler driver first tries the **-B** prefix, if any. If that name is not found, or if **-B** was not specified, the driver tries two standard prefixes, which are */usr/lib/gcc/* and */usr/local/lib/gcc-lib/*. If neither of those results in a file name that is found, the unmodified program name is searched for using the directories specified in your **PATH** environment variable.

-B prefixes that effectively specify directory names also apply to libraries in the linker, because the compiler translates these options into **-L** options for the linker. They also apply to includes files in the preprocessor, because the compiler translates these options into **-isystem** options for the preprocessor. In this case, the compiler appends **include** to the prefix.

The run-time support file *libgcc.a* can also be searched for using the **-B** prefix, if needed. If it is not found there, the two standard prefixes above are tried, and that is all. The file is left out of the link if it is not found by those means.

Another way to specify a prefix much like the **-B** prefix is to use the environment variable **GCC_EXEC_PREFIX**.

-specs=file

Process *file* after the compiler reads in the standard *specs* file, in order to override the defaults that the *gcc* driver program uses when determining what switches to pass to *cc1*, *cc1plus*, *as*, *ld*, etc. More than one **-specs=file** can be specified on the command line, and they are processed in order, from left to right.

Specifying Target Machine and Compiler Version

By default, GCC compiles code for the same type of machine that you are using. However, it can also be installed as a cross-compiler, to compile for some other type of machine. In fact, several different configurations of GCC, for different target machines, can be installed side by side. Then you specify which one to use with the **-b** option.

In addition, older and newer versions of GCC can be installed side by side. One of them (probably the newest) will be the default, but you may sometimes wish to use another.

-b machine

The argument *machine* specifies the target machine for compilation. This is useful when you have installed GCC as a cross-compiler.

The value to use for *machine* is the same as was specified as the machine type when configuring GCC as a cross-compiler. For example, if a cross-compiler was configured with **configure i386v**, meaning to compile for an 80386 running System V, then you would specify **-b i386v** to run that cross compiler.

When you do not specify **-b**, it normally means to compile for the same type of machine that you are using.

-V version

The argument *version* specifies which version of GCC to run. This is useful when multiple versions are installed. For example, *version* might be **2.0**, meaning to run GCC version 2.0.

The default version, when you do not specify **-V**, is the last version of GCC that you installed.

The **-b** and **-V** options actually work by controlling part of the file name used for the executable files and libraries used for compilation. A given version of GCC, for a given target machine, is normally kept in the directory `/usr/local/lib/gcc-lib/machine/version`.

Thus, sites can customize the effect of **-b** or **-V** either by changing the names of these directories or adding alternate names (or symbolic links). If in directory `/usr/local/lib/gcc-lib/` the file `80386` is a link to the file `i386v`, then **-b 80386** becomes an alias for **-b i386v**.

In one respect, the **-b** or **-V** do not completely change to a different compiler: the top-level driver program **gcc** that you originally invoked continues to run and invoke the other executables (preprocessor, compiler per se, assembler and linker) that do the real work. However, since no real work is done in the driver program, it usually does not matter that the driver program in use is not the one for the specified target. It is common for the interface to the other executables to change incompatibly between compiler versions, so unless the version specified is very close to that of the driver (for example, **-V 3.0** with a driver program from GCC version 3.0.1), use of **-V** may not work; for example, using **-V 2.95.2** will not work with a driver program from GCC 3.0.

The only way that the driver program depends on the target machine is in the parsing and handling of special machine-specific options. However, this is controlled by a file which is found, along with the other executables, in the directory for the specified version and target machine. As a result, a single installed driver program adapts to any specified target machine, and sufficiently similar compiler versions.

The driver program executable does control one significant thing, however: the default version and target machine. Therefore, you can install different instances of the driver program, compiled for different targets or versions, under different names.

For example, if the driver for version 2.0 is installed as **ogcc** and that for version 2.1 is installed as **gcc**, then the command **gcc** will use version 2.1 by default, while **ogcc** will use 2.0 by default. However, you can choose either version with either command with the **-V** option.

Hardware Models and Configurations

Earlier we discussed the standard option **-b** which chooses among different installed compilers for completely different target machines, such as Vax vs. 68000 vs. 80386.

In addition, each of these target machine types can have its own special options, starting with **-m**, to choose among various hardware models or configurations—for example, 68010 vs 68020, floating coprocessor or none. A single installed version of the compiler can compile for any model or configuration, according to the options specified.

Some configurations of the compiler also support additional special options, usually for compatibility with other compilers on the same platform.

M680x0 Options

These are the **-m** options defined for the 68000 series. The default values for these options depends on which style of 68000 was selected when the compiler was configured; the defaults for the most common choices are given below.

-m68000**-mc68000**

Generate output for a 68000. This is the default when the compiler is configured for 68000-based systems.

Use this option for microcontrollers with a 68000 or EC000 core, including the 68008, 68302, 68306, 68307, 68322, 68328 and 68356.

-m68020**-mc68020**

Generate output for a 68020. This is the default when the compiler is configured for 68020-based systems.

-m68881

Generate output containing 68881 instructions for floating point. This is the default for most 68020 systems unless **-nfp** was specified when the compiler was configured.

-m68030

Generate output for a 68030. This is the default when the compiler is configured for 68030-based systems.

-m68040

Generate output for a 68040. This is the default when the compiler is configured for 68040-based systems.

This option inhibits the use of 68881/68882 instructions that have to be emulated by software on the 68040. Use this option if your 68040 does not have code to emulate those instructions.

-m68060

Generate output for a 68060. 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. Use this option if your 68060 does not have code to emulate those instructions.

-mcpu32

Generate output for a CPU32. This is the default when the compiler is configured for CPU32-based systems.

Use this option for microcontrollers with a CPU32 or CPU32+ core, including the 68330, 68331, 68332, 68333, 68334, 68336, 68340, 68341, 68349 and 68360.

-m5200

Generate output for a 520X “coldfire” family cpu. This is the default when the compiler is configured for 520X-based systems.

Use this option for microcontroller with a 5200 core, including the MCF5202, MCF5203, MCF5204 and MCF5202.

-m68020-40

Generate output for a 68040, without using any of the new instructions. This results in code which can run relatively efficiently on either a 68020/68881 or a 68030 or a 68040. The generated code does use the 68881 instructions that are emulated on the 68040.

-m68020-60

Generate output for a 68060, without using any of the new instructions. This results in code which can run relatively efficiently on either a 68020/68881 or a 68030 or a 68040. The generated code does use the 68881 instructions that are emulated on the 68060.

-mfpa

Generate output containing Sun FPA instructions for floating point.

-msoft-float

Generate output containing library calls for floating point. **Warning:** the requisite libraries are not available for all m68k targets. Normally the facilities of the machine’s usual C compiler are used, but this can’t be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation. The embedded targets **m68k-*-aout** and **m68k-*-coff** do provide software floating point support.

-mshort

Consider type `int` to be 16 bits wide, like `short int`.

-mnobitfield

Do not use the bit-field instructions. The **-m68000**, **-mcpu32** and **-m5200** options imply **-mnobitfield**.

-mbitfield

Do use the bit-field instructions. The **-m68020** option implies **-mbitfield**. This is the default if you use a configuration designed for a 68020.

-mrtd

Use a different function-calling convention, in which functions that take a fixed number of arguments return with the `rtd` instruction, which pops their arguments while returning. This 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, so you cannot use it if you need to call libraries compiled with the Unix compiler.

Also, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`); otherwise incorrect code will be generated for calls to those functions.

In addition, seriously incorrect code will result if you call a function with too many arguments. (Normally, extra arguments are harmlessly ignored.)

The `rtd` instruction is supported by the 68010, 68020, 68030, 68040, 68060 and CPU32 processors, but not by the 68000 or 5200.

-malign-int**-mno-align-int**

Control whether GCC aligns `int`, `long`, `long long`, `float`, `double`, and `long double` variables on a 32-bit boundary (**-malign-int**) or a 16-bit boundary (**-mno-align-int**). Aligning variables on 32-bit boundaries produces code that runs somewhat faster on processors with 32-bit busses at the expense of more memory.

Warning: if you use the **-malign-int** switch, GCC will align structures containing the above types differently than most published application binary interface specifications for the m68k.

-mpcrel

Use the pc-relative addressing mode of the 68000 directly, instead of using a global offset table. At present, this option implies **-fpic**, allowing at most a 16-bit offset for pc-relative addressing. **-fPIC** is not presently supported with **-mpcrel**, though this could be supported for 68020 and higher processors.

-mno-strict-align**-mstrict-align**

Do not (do) assume that unaligned memory references will be handled by the system.

M68hc1x Options

These are the **-m** options defined for the 68hc11 and 68hc12 microcontrollers. The default values for these options depends on which style of microcontroller was selected when the compiler was configured; the defaults for the most common choices are given below.

-m6811**-m68hc11**

Generate output for a 68HC11. This is the default when the compiler is configured for 68HC11-based systems.

-m6812

-m68hc12

Generate output for a 68HC12. This is the default when the compiler is configured for 68HC12-based systems.

-mauto-incdec

Enable the use of 68HC12 pre and post auto-increment and auto-decrement addressing modes.

-mshort

Consider type `int` to be 16 bits wide, like `short int`.

-msoft-reg-count=count

Specify the number of pseudo-soft registers which are used for the code generation. The maximum number is 32. Using more pseudo-soft register may or may not result in better code depending on the program. The default is 4 for 68HC11 and 2 for 68HC12.

VAX Options

These **-m** options are defined for the Vax:

-munix

Do not output certain jump instructions (`aobleq` and so on) that the Unix assembler for the Vax cannot handle across long ranges.

-mgnu

Do output those jump instructions, on the assumption that you will assemble with the GNU assembler.

-mg

Output code for g-format floating point numbers instead of d-format.

SPARC Options

These **-m** switches are supported on the SPARC:

-mno-app-regs**-mapp-regs**

Specify **-mapp-regs** to generate output using the global registers 2 through 4, which the SPARC SVR4 ABI reserves for applications. This is the default.

To be fully SVR4 ABI compliant at the cost of some performance loss, specify **-mno-app-regs**. You should compile libraries and system software with this option.

-mfpu**-mhard-float**

Generate output containing floating point instructions. This is the default.

-mno-fpu**-msoft-float**

Generate output containing library calls for floating point. **Warning:** the requisite libraries are not available for all SPARC targets. Normally the facilities of the machine's usual C compiler are used, but this cannot be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation. The embedded targets **sparc*-aout** and **sparc-lite-*-*** do provide software floating point support.

-msoft-float changes the calling convention in the output file; therefore, it is only useful if you compile *all* of a program with this option. In particular, you need to compile *libgcc.a*, the library that comes with GCC, with **-msoft-float** in order for this to work.

-mhard-quad-float

Generate output containing quad-word (long double) floating point instructions.

-msoft-quad-float

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.

As of this writing, there are no sparc implementations that have hardware support for the quad-word

floating point instructions. They all invoke a trap handler for one of these instructions, and then the trap handler emulates the effect of the instruction. Because of the trap handler overhead, this is much slower than calling the ABI library routines. Thus the **-msoft-quad-float** option is the default.

-mno-epilogue

-mepilogue

With **-mepilogue** (the default), the compiler always emits code for function exit at the end of each function. Any function exit in the middle of the function (such as a return statement in C) will generate a jump to the exit code at the end of the function.

With **-mno-epilogue**, the compiler tries to emit exit code inline at every function exit.

-mno-flat

-mflat

With **-mflat**, the compiler does not generate save/restore instructions and will use a “flat” or single register window calling convention. This model uses `%i7` as the frame pointer and is compatible with the normal register window model. Code from either may be intermixed. 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.

With **-mno-flat** (the default), the compiler emits save/restore instructions (except for leaf functions) and is the normal mode of operation.

-mno-unaligned-doubles

-munaligned-doubles

Assume that doubles have 8 byte alignment. This is the default.

With **-munaligned-doubles**, 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. It is not the default because it results in a performance loss, especially for floating point code.

-mno-faster-structs

-mfaster-structs

With **-mfaster-structs**, the compiler assumes that structures should have 8 byte alignment. This enables the use of pairs of `ldd` and `std` instructions for copies in structure assignment, in place of twice as many `ld` and `st` pairs. However, the use of this changed alignment directly violates the SPARC ABI. Thus, it's intended only for use on targets where the developer acknowledges that their resulting code will not be directly in line with the rules of the ABI.

-mv8

-msparclite

These two options select variations on the SPARC architecture.

By default (unless specifically configured for the Fujitsu SPARClite), GCC generates code for the v7 variant of the SPARC architecture.

-mv8 will give you SPARC v8 code. The only difference from v7 code is that the compiler emits the integer multiply and integer divide instructions which exist in SPARC v8 but not in SPARC v7.

-msparclite will give you SPARClite code. This adds the integer multiply, integer divide step and scan (`ffs`) instructions which exist in SPARClite but not in SPARC v7.

These options are deprecated and will be deleted in a future GCC release. They have been replaced with **-mcpu=xxx**.

-mcypress

-msupersparc

These two options select the processor for which the code is optimised.

With **-mcypress** (the default), the compiler optimizes 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 etc.

With **-msupersparc** the compiler optimizes code for the SuperSparc cpu, as used in the SparcStation 10, 1000 and 2000 series. This flag also enables use of the full SPARC v8 instruction set.

These options are deprecated and will be deleted in a future GCC release. They have been replaced with **-mcpu=xxx**.

-mcpu=cpu_type

Set the instruction set, register set, and instruction scheduling parameters for machine type *cpu_type*. Supported values for *cpu_type* are **v7**, **cypress**, **v8**, **supersparc**, **sparclite**, **hypersparc**, **sparclite86x**, **f930**, **f934**, **sparclet**, **tsc701**, **v9**, and **ultrasparc**.

Default instruction scheduling parameters are used for values that select an architecture and not an implementation. These are **v7**, **v8**, **sparclite**, **sparclet**, **v9**.

Here is a list of each supported architecture and their supported implementations.

v7:	cypress
v8:	supersparc, hypersparc
sparclite:	f930, f934, sparclite86x
sparclet:	tsc701
v9:	ultrasparc

-mtune=cpu_type

Set the instruction scheduling parameters for machine type *cpu_type*, but do not set the instruction set or register set that the option **-mcpu=cpu_type** would.

The same values for **-mcpu=cpu_type** are used for **-mtune=cpu_type**, though the only useful values are those that select a particular cpu implementation: **cypress**, **supersparc**, **hypersparc**, **f930**, **f934**, **sparclite86x**, **tsc701**, **ultrasparc**.

These **-m** switches are supported in addition to the above on the SPARCLET processor.

-mlittle-endian

Generate code for a processor running in little-endian mode.

-mlive-g0

Treat register %g0 as a normal register. GCC will continue to clobber it as necessary but will not assume it always reads as 0.

-mbroken-saverestore

Generate code that does not use non-trivial forms of the `save` and `restore` instructions. Early versions of the SPARCLET processor do not correctly handle `save` and `restore` instructions used with arguments. They correctly handle them used without arguments. A `save` instruction used without arguments increments the current window pointer but does not allocate a new stack frame. It is assumed that the window overflow trap handler will properly handle this case as will interrupt handlers.

These **-m** switches are supported in addition to the above on SPARC V9 processors in 64 bit environments.

-mlittle-endian

Generate code for a processor running in little-endian mode.

-m32

-m64

Generate code for a 32 bit or 64 bit environment. The 32 bit environment sets `int`, `long` and `pointer` to 32 bits. The 64 bit environment sets `int` to 32 bits and `long` and `pointer` to 64 bits.

-mcmmodel=medlow

Generate code for the Medium/Low code model: the program must be linked in the low 32 bits of the address space. Pointers are 64 bits. Programs can be statically or dynamically linked.

-mcmmodel=medmid

Generate code for the Medium/Middle code model: the program must be linked in the low 44 bits of the address space, the text segment must be less than 2G bytes, and data segment must be within 2G of the text segment. Pointers are 64 bits.

-mcmmodel=medany

Generate code for the Medium/Anywhere code model: the program may be linked anywhere in the address space, the text segment must be less than 2G bytes, and data segment must be within 2G of the text segment. Pointers are 64 bits.

-mcmmodel=embmedany

Generate code for the Medium/Anywhere code model for embedded systems: assume 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 still 64 bits. Programs are statically linked, PIC is not supported.

-mstack-bias**-mno-stack-bias**

With **-mstack-bias**, GCC assumes that the stack pointer, and frame pointer if present, are offset by -2047 which must be added back when making stack frame references. Otherwise, assume no such offset is present.

Convex Options

These **-m** options are defined for Convex:

-mc1

Generate output for C1. The code will run on any Convex machine. The preprocessor symbol `__convex_c1__` is defined.

-mc2

Generate output for C2. Uses instructions not available on C1. Scheduling and other optimizations are chosen for max performance on C2. The preprocessor symbol `__convex_c2__` is defined.

-mc32

Generate output for C32xx. Uses instructions not available on C1. Scheduling and other optimizations are chosen for max performance on C32. The preprocessor symbol `__convex_c32__` is defined.

-mc34

Generate output for C34xx. Uses instructions not available on C1. Scheduling and other optimizations are chosen for max performance on C34. The preprocessor symbol `__convex_c34__` is defined.

-mc38

Generate output for C38xx. Uses instructions not available on C1. Scheduling and other optimizations are chosen for max performance on C38. The preprocessor symbol `__convex_c38__` is defined.

-margcount

Generate code which puts an argument count in the word preceding each argument list. This is compatible with regular CC, and a few programs may need the argument count word. GDB and other source-level debuggers do not need it; this info is in the symbol table.

-mnoargcount

Omit the argument count word. This is the default.

-mvolatile-cache

Allow volatile references to be cached. This is the default.

-mvolatile-nocache

Volatile references bypass the data cache, going all the way to memory. This is only needed for multi-processor code that does not use standard synchronization instructions. Making non-volatile references to volatile locations will not necessarily work.

-mlong32

Type long is 32 bits, the same as type int. This is the default.

-mlong64

Type long is 64 bits, the same as type long long. This option is useless, because no library support exists for it.

AMD29K Options

These **-m** options are defined for the AMD Am29000:

-mdw

Generate code that assumes the DW bit is set, i.e., that byte and halfword operations are directly supported by the hardware. This is the default.

-mndw

Generate code that assumes the DW bit is not set.

-mbw

Generate code that assumes the system supports byte and halfword write operations. This is the default.

-mnbw

Generate code that assumes the system does not support byte and halfword write operations. **-mnbw** implies **-mndw**.

-msmall

Use a small memory model that assumes that all function addresses are either within a single 256 KB segment or at an absolute address of less than 256k. This allows the `call` instruction to be used instead of a `const`, `consth`, `calli` sequence.

-mnormal

Use the normal memory model: Generate `call` instructions only when calling functions in the same file and `calli` instructions otherwise. This works if each file occupies less than 256 KB but allows the entire executable to be larger than 256 KB. This is the default.

-mlarge

Always use `calli` instructions. Specify this option if you expect a single file to compile into more than 256 KB of code.

-m29050

Generate code for the Am29050.

-m29000

Generate code for the Am29000. This is the default.

-mkernel-registers

Generate references to registers `gr64-gr95` instead of to registers `gr96-gr127`. This option can be used when compiling kernel code that wants a set of global registers disjoint from that used by user-mode code.

Note that when this option is used, register names in **-f** flags must use the normal, user-mode, names.

-muser-registers

Use the normal set of global registers, `gr96-gr127`. This is the default.

–mstack-check**–mno-stack-check**

Insert (or do not insert) a call to `__msp_check` after each stack adjustment. This is often used for kernel code.

–mstorem-bug**–mno-storem-bug**

–mstorem-bug handles 29k processors which cannot handle the separation of a `mtsrin` insn and a `storem` instruction (most 29000 chips to date, but not the 29050).

–mno-reuse-arg-regs**–mreuse-arg-regs**

–mno-reuse-arg-regs tells the compiler to only use incoming argument registers for copying out arguments. This helps detect calling a function with fewer arguments than it was declared with.

–mno-impure-text**–mimpure-text**

–mimpure-text, used in addition to **–shared**, tells the compiler to not pass **–assert pure-text** to the linker when linking a shared object.

–msoft-float

Generate output containing library calls for floating point. **Warning:** the requisite libraries are not part of GCC. Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

–mno-multm

Do not generate `multm` or `multmu` instructions. This is useful for some embedded systems which do not have trap handlers for these instructions.

ARM Options

These **–m** options are defined for Advanced RISC Machines (ARM) architectures:

–mapcs-frame

Generate a stack frame that is compliant with the ARM Procedure Call Standard for all functions, even if this is not strictly necessary for correct execution of the code. Specifying **–fomit-frame-pointer** with this option will cause the stack frames not to be generated for leaf functions. The default is **–mno-apcs-frame**.

–mapcs

This is a synonym for **–mapcs-frame**.

–mapcs-26

Generate code for a processor running with a 26-bit program counter, and conforming to the function calling standards for the APCS 26-bit option. This option replaces the **–m2** and **–m3** options of previous releases of the compiler.

–mapcs-32

Generate code for a processor running with a 32-bit program counter, and conforming to the function calling standards for the APCS 32-bit option. This option replaces the **–m6** option of previous releases of the compiler.

–mapcs-stack-check

Generate code to check the amount of stack space available upon entry to every function (that actually uses some stack space). If there is insufficient space available then either the function `__rt_stkovf_split_small` or `__rt_stkovf_split_big` will be called, depending upon the amount of stack space required. The run time system is required to provide these functions. The default is **–mno-apcs-stack-check**, since this produces smaller code.

-mapcs-float

Pass floating point arguments using the float point registers. This is one of the variants of the APCS. This option is recommended if the target hardware has a floating point unit or if a lot of floating point arithmetic is going to be performed by the code. The default is **-mno-apcs-float**, since integer only code is slightly increased in size if **-mapcs-float** is used.

-mapcs-reentrant

Generate reentrant, position independent code. This is the equivalent to specifying the **-fpic** option. The default is **-mno-apcs-reentrant**.

-mthumb-interwork

Generate code which supports calling between the ARM and THUMB instruction sets. Without this option the two instruction sets cannot be reliably used inside one program. The default is **-mno-thumb-interwork**, since slightly larger code is generated when **-mthumb-interwork** is specified.

-mno-sched-prolog

Prevent the reordering of instructions in the function prolog, or the merging of those instruction with the instructions in the function's body. This means that all functions will start with a recognizable set of instructions (or in fact one of a choice from a small set of different function prologues), and this information can be used to locate the start of functions inside an executable piece of code. The default is **-msched-prolog**.

-mhard-float

Generate output containing floating point instructions. This is the default.

-msoft-float

Generate output containing library calls for floating point. **Warning:** the requisite libraries are not available for all ARM targets. Normally the facilities of the machine's usual C compiler are used, but this cannot be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

-msoft-float changes the calling convention in the output file; therefore, it is only useful if you compile *all* of a program with this option. In particular, you need to compile *libgcc.a*, the library that comes with GCC, with **-msoft-float** in order for this to work.

-mlittle-endian

Generate code for a processor running in little-endian mode. This is the default for all standard configurations.

-mbig-endian

Generate code for a processor running in big-endian mode; the default is to compile code for a little-endian processor.

-mwords-little-endian

This option only applies when generating code for big-endian processors. Generate code for a little-endian word order but a big-endian byte order. That is, a byte order of the form **32107654**. Note: this option should only be used if you require compatibility with code for big-endian ARM processors generated by versions of the compiler prior to 2.8.

-malignment-traps

Generate code that will not trap if the MMU has alignment traps enabled. On ARM architectures prior to ARMv4, there were no instructions to access half-word objects stored in memory. However, when reading from memory a feature of the ARM architecture allows a word load to be used, even if the address is unaligned, and the processor core will rotate the data as it is being loaded. This option tells the compiler that such misaligned accesses will cause a MMU trap and that it should instead synthesise the access as 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.

This option is ignored when compiling for ARM architecture 4 or later, since these processors have instructions to directly access half-word objects in memory.

-mno-alignment-traps

Generate code that assumes that the MMU will not trap unaligned accesses. This produces better code when the target instruction set does not have half-word memory operations (implementations prior to ARMv4).

Note that you cannot use this option to access unaligned word objects, since the processor will only fetch one 32-bit aligned object from memory.

The default setting for most targets is `-mno-alignment-traps`, since this produces better code when there are no half-word memory instructions available.

-mshort-load-bytes

This is a deprecated alias for `-malignment-traps`.

-mno-short-load-bytes

This is a deprecated alias for `-mno-alignment-traps`.

-mshort-load-words

This is a deprecated alias for `-mno-alignment-traps`.

-mno-short-load-words

This is a deprecated alias for `-malignment-traps`.

-mbsd

This option only applies to RISC iX. Emulate the native BSD-mode compiler. This is the default if `-ansi` is not specified.

-mxopen

This option only applies to RISC iX. Emulate the native X/Open-mode compiler.

-mno-symrename

This option only applies to RISC iX. Do not run the assembler post-processor, `symrename`, after code has been assembled. Normally it is necessary to modify some of the standard symbols in preparation for linking with the RISC iX C library; this option suppresses this pass. The post-processor is never run when the compiler is built for cross-compilation.

-mcpu=<name>

This specifies the name of the target ARM processor. GCC uses this name to determine what kind of instructions it can use when generating assembly code. Permissible names 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, arm920, arm920t, arm9tdmi.

-mtune=<name>

This option is very similar to the `-mcpu=` option, except that instead of specifying the actual target processor type, and hence restricting which instructions can be used, it specifies that GCC should tune the performance of the code as if the target were of the type specified in this option, but still choosing the instructions that it will generate based on the cpu specified by a `-mcpu=` option. For some arm implementations better performance can be obtained by using this option.

-march=<name>

This specifies the name of the target ARM architecture. GCC uses this name to determine what kind of instructions it can use when generating assembly code. This option can be used in conjunction with or instead of the `-mcpu=` option. Permissible names are: armv2, armv2a, armv3, armv3m, armv4, armv4t, armv5.

-mfpe=<number>**-mfp=<number>**

This specifies the version of the floating point emulation available on the target. Permissible values are 2 and 3. `-mfp=` is a synonym for `-mfpe=` to support older versions of GCC.

-mstructure-size-boundary=<n>

The size of all structures and unions will be rounded up to a multiple of the number of bits set by this option. Permissible values are 8 and 32. The default value varies for different toolchains. For the COFF targeted toolchain the default value is 8. Specifying the larger number can produce faster, more efficient code, but can also increase the size of the program. The two values are potentially incompatible. Code compiled with one value cannot necessarily expect to work with code or libraries compiled with the other value, if they exchange information using structures or unions. Programmers are encouraged to use the 32 value as future versions of the toolchain may default to this value.

-mabort-on-noreturn

Generate a call to the function abort at the end of a noreturn function. It will be executed if the function tries to return.

-mlong-calls**-mno-long-calls**

Tells the compiler 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 switch is needed if the target function will lie outside of the 64 megabyte addressing range of the offset based version of subroutine call instruction.

Even if this switch is enabled, not all function calls will be turned into long calls. The heuristic is that static functions, functions which 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, will not be turned into long calls. The exception to this rule is that 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, will always be turned into long calls.

This feature is not enabled by default. Specifying **—no-long-calls** will restore the default behaviour, as will placing the function calls within the scope of a **#pragma long_calls_off** directive. Note these switches have no effect on how the compiler generates code to handle function calls via function pointers.

-mnop-fun-dllimport

Disable the support for the *dllimport* attribute.

-msingle-pic-base

Treat the register used for PIC addressing as read-only, rather than loading it in the prologue for each function. The run-time system is responsible for initialising this register with an appropriate value before execution begins.

-mpic-register=<reg>

Specify the register to be used for PIC addressing. The default is R10 unless stack-checking is enabled, when R9 is used.

*Thumb Options***-mthumb-interwork**

Generate code which supports calling between the THUMB and ARM instruction sets. Without this option the two instruction sets cannot be reliably used inside one program. The default is **-mno-thumb-interwork**, since slightly smaller code is generated with this option.

-mtpcs-frame

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all non-leaf functions. (A leaf function is one that does not call any other functions). The default is **-mno-apcs-frame**.

-mtpcs-leaf-frame

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all leaf functions. (A leaf function is one that does not call any other functions). The default is **-mno-apcs-leaf-frame**.

-mlittle-endian

Generate code for a processor running in little-endian mode. This is the default for all standard configurations.

-mbig-endian

Generate code for a processor running in big-endian mode.

-mstructure-size-boundary=<n>

The size of all structures and unions will be rounded up to a multiple of the number of bits set by this option. Permissible values are 8 and 32. The default value varies for different toolchains. For the COFF targeted toolchain the default value is 8. Specifying the larger number can produce faster, more efficient code, but can also increase the size of the program. The two values are potentially incompatible. Code compiled with one value cannot necessarily expect to work with code or libraries compiled with the other value, if they exchange information using structures or unions. Programmers are encouraged to use the 32 value as future versions of the toolchain may default to this value.

-mnop-fun-dllimport

Disable the support for the *dllimport* attribute.

-mcallee-super-interworking

Gives all externally visible functions in the file being compiled an ARM instruction set header which switches to Thumb mode before executing the rest of the function. This allows these functions to be called from non-interworking code.

-mcaller-super-interworking

Allows calls via function pointers (including virtual functions) to execute correctly regardless of whether the target code has been compiled for interworking or not. There is a small overhead in the cost of executing a function pointer if this option is enabled.

-msingle-pic-base

Treat the register used for PIC addressing as read-only, rather than loading it in the prologue for each function. The run-time system is responsible for initialising this register with an appropriate value before execution begins.

-mpic-register=<reg>

Specify the register to be used for PIC addressing. The default is R10.

MN10200 Options

These **-m** options are defined for Matsushita MN10200 architectures:

-mrelax

Indicate to the linker that it should perform a relaxation optimization pass to shorten branches, calls and absolute memory addresses. This option only has an effect when used on the command line for the final link step.

This option makes symbolic debugging impossible.

MN10300 Options

These **-m** options are defined for Matsushita MN10300 architectures:

-mmult-bug

Generate code to avoid bugs in the multiply instructions for the MN10300 processors. This is the default.

-mno-mult-bug

Do not generate code to avoid bugs in the multiply instructions for the MN10300 processors.

-mam33

Generate code which uses features specific to the AM33 processor.

-mno-am33

Do not generate code which uses features specific to the AM33 processor. This is the default.

-mrelax

Indicate to the linker that it should perform a relaxation optimization pass to shorten branches, calls and absolute memory addresses. This option only has an effect when used on the command line for the final link step.

This option makes symbolic debugging impossible.

M32R/D Options

These **-m** options are defined for Mitsubishi M32R/D architectures:

-mcode-model=small

Assume all objects live in the lower 16MB of memory (so that their addresses can be loaded with the `ld24` instruction), and assume all subroutines are reachable with the `bl` instruction. This is the default.

The addressability of a particular object can be set with the `model` attribute.

-mcode-model=medium

Assume objects may be anywhere in the 32 bit address space (the compiler will generate `seth/add3` instructions to load their addresses), and assume all subroutines are reachable with the `bl` instruction.

-mcode-model=large

Assume objects may be anywhere in the 32 bit address space (the compiler will generate `seth/add3` instructions to load their addresses), and assume subroutines may not be reachable with the `bl` instruction (the compiler will generate the much slower `seth/add3/jl` instruction sequence).

-msdata=none

Disable use of the small data area. Variables will be put into one of **.data**, **bss**, or **.rodata** (unless the `section` attribute has been specified). This is the default.

The small data area consists of sections **.sdata** and **.sbss**. Objects may be explicitly put in the small data area with the `section` attribute using one of these sections.

-msdata=sdata

Put small global and static data in the small data area, but do not generate special code to reference them.

-msdata=use

Put small global and static data in the small data area, and generate special instructions to reference them.

-G num

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 one of **sdata** or **use** for this option to have any effect.

All modules should be compiled with the same **-G num** value. Compiling with different values of *num* may or may not work; if it doesn't the linker will give an error message – incorrect code will not be generated.

M88K Options

These **-m** options are defined for Motorola 88k architectures:

-m88000

Generate code that works well on both the m88100 and the m88110.

-m88100

Generate code that works best for the m88100, but that also runs on the m88110.

-m88110

Generate code that works best for the m88110, and may not run on the m88100.

-mbig-pic

Obsolete option to be removed from the next revision. Use **-fPIC**.

-midentify-revision

Include an `ident` directive in the assembler output recording the source file name, compiler name and version, timestamp, and compilation flags used.

-mno-underscores

In assembler output, emit symbol names without adding an underscore character at the beginning of each name. The default is to use an underscore as prefix on each name.

-mocs-debug-info**-mno-ocs-debug-info**

Include (or omit) additional debugging information (about registers used in each stack frame) as specified in the 88open Object Compatibility Standard, “OCS”. This extra information allows debugging of code that has had the frame pointer eliminated. The default for DG/UX, SVr4, and Delta 88 SVr3.2 is to include this information; other 88k configurations omit this information by default.

-mocs-frame-position

When emitting COFF debugging information for automatic variables and parameters stored on the stack, use the offset from the canonical frame address, which is the stack pointer (register 31) on entry to the function. The DG/UX, SVr4, Delta88 SVr3.2, and BCS configurations use **-mocs-frame-position**; other 88k configurations have the default **-mno-ocs-frame-position**.

-mno-ocs-frame-position

When emitting COFF debugging information for automatic variables and parameters stored on the stack, use the offset from the frame pointer register (register 30). When this option is in effect, the frame pointer is not eliminated when debugging information is selected by the `-g` switch.

-moptimize-arg-area**-mno-optimize-arg-area**

Control how function arguments are stored in stack frames. **-moptimize-arg-area** saves space by optimizing them, but this conflicts with the 88open specifications. The opposite alternative, **-mno-optimize-arg-area**, agrees with 88open standards. By default GCC does not optimize the argument area.

-mshort-data-num

Generate smaller data references by making them relative to `⌞0`, which allows loading a value using a single instruction (rather than the usual two). You control which data references are affected by specifying *num* with this option. For example, if you specify **-mshort-data-512**, then the data references affected are those involving displacements of less than 512 bytes. **-mshort-data-num** is not effective for *num* greater than 64k.

-mserialize-volatile**-mno-serialize-volatile**

Do, or don't, generate code to guarantee sequential consistency of volatile memory references. By default, consistency is guaranteed.

The order of memory references made by the MC88110 processor does not always match the order of the instructions requesting those references. In particular, a load instruction may execute before a preceding store instruction. Such reordering violates sequential consistency of volatile memory references, when there are multiple processors. When consistency must be guaranteed, GNU C generates special instructions, as needed, to force execution in the proper order.

The MC88100 processor does not reorder memory references and so always provides sequential consistency. However, by default, GNU C generates the special instructions to guarantee consistency even when you use **-m88100**, so that the code may be run on an MC88110 processor. If you intend to run your code only on the MC88100 processor, you may use **-mno-serialize-volatile**.

The extra code generated to guarantee consistency may affect the performance of your application. If you know that you can safely forgo this guarantee, you may use **-mno-serialize-volatile**.

-msvr4

-msvr3

Turn on (**-msvr4**) or off (**-msvr3**) compiler extensions related to System V release 4 (SVr4). This controls the following:

1. Which variant of the assembler syntax to emit.
2. **-msvr4** makes the C preprocessor recognize **#pragma weak** that is used on System V release 4.
3. **-msvr4** makes GCC issue additional declaration directives used in SVr4.

-msvr4 is the default for the m88k-motorola-sysv4 and m88k-dg-dgux m88k configurations. **-msvr3** is the default for all other m88k configurations.

-mversion-03.00

This option is obsolete, and is ignored.

-mno-check-zero-division

-mcheck-zero-division

Do, or don't, generate code to guarantee that integer division by zero will be detected. By default, detection is guaranteed.

Some models of the MC88100 processor fail to trap upon integer division by zero under certain conditions. By default, when compiling code that might be run on such a processor, GNU C generates code that explicitly checks for zero-valued divisors and traps with exception number 503 when one is detected. Use of **mno-check-zero-division** suppresses such checking for code generated to run on an MC88100 processor.

GNU C assumes that the MC88110 processor correctly detects all instances of integer division by zero. When **-m88110** is specified, both **-mcheck-zero-division** and **-mno-check-zero-division** are ignored, and no explicit checks for zero-valued divisors are generated.

-muse-div-instruction

Use the div instruction for signed integer division on the MC88100 processor. By default, the div instruction is not used.

On the MC88100 processor the signed integer division instruction div) traps to the operating system on a negative operand. The operating system transparently completes the operation, but at a large cost in execution time. By default, when compiling code that might be run on an MC88100 processor, GNU C emulates signed integer division using the unsigned integer division instruction divu), thereby avoiding the large penalty of a trap to the operating system. Such emulation has its own, smaller, execution cost in both time and space. To the extent that your code's important signed integer division operations are performed on two nonnegative operands, it may be desirable to use the div instruction directly.

On the MC88110 processor the div instruction (also known as the divs instruction) processes negative operands without trapping to the operating system. When **-m88110** is specified, **-muse-div-instruction** is ignored, and the div instruction is used for signed integer division.

Note that the result of dividing INT_MIN by -1 is undefined. In particular, the behavior of such a division with and without **-muse-div-instruction** may differ.

-mtrap-large-shift

-mhandle-large-shift

Include code to detect bit-shifts of more than 31 bits; respectively, trap such shifts or emit code to handle them properly. By default GCC makes no special provision for large bit shifts.

-mwarn-passed-structs

Warn when a function passes a struct as an argument or result. Structure-passing conventions have changed during the evolution of the C language, and are often the source of portability problems. By default, GCC issues no such warning.

IBM RS/6000 and PowerPC Options

These **-m** options are defined for the IBM RS/6000 and PowerPC:

-mpower
-mno-power
-mpower2
-mno-power2
-mpowerpc
-mno-powerpc
-mpowerpc-gpopt
-mno-powerpc-gpopt
-mpowerpc-gfxopt
-mno-powerpc-gfxopt
-mpowerpc64
-mno-powerpc64

GCC supports two related instruction set architectures for the RS/6000 and PowerPC. The *POWER* instruction set are those instructions supported by the **rios** chip set used in the original RS/6000 systems and 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. However there is a large common subset of instructions supported by both. An MQ register is included in processors supporting the POWER architecture.

You use these options to specify which instructions are available on the processor you are using. The default value of these options is determined when configuring GCC. Specifying the **-mcpu=cpu_type** overrides the specification of these options. We recommend you use the **-mcpu=cpu_type** option rather than the options listed above.

The **-mpower** option allows GCC to generate instructions that are found only in the POWER architecture and to use the MQ register. Specifying **-mpower2** implies **-mpower** and also allows GCC to generate instructions that are present in the POWER2 architecture but not the original POWER architecture.

The **-mpowerpc** option allows GCC to generate instructions that are found only in the 32-bit subset of the PowerPC architecture. Specifying **-mpowerpc-gpopt** implies **-mpowerpc** and also allows GCC to use the optional PowerPC architecture instructions in the General Purpose group, including floating-point square root. Specifying **-mpowerpc-gfxopt** implies **-mpowerpc** and also allows GCC to use the optional PowerPC architecture instructions in the Graphics group, including floating-point select.

The **-mpowerpc64** option allows GCC to generate the additional 64-bit instructions that are found in the full PowerPC64 architecture and to treat GPRs as 64-bit, doubleword quantities. GCC defaults to **-mno-powerpc64**.

If you specify both **-mno-power** and **-mno-powerpc**, 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. Specifying both **-mpower** and **-mpowerpc** permits GCC to use any instruction from either architecture and to allow use of the MQ register; specify this for the Motorola MPC601.

-mnew-mnemonics**-mold-mnemonics**

Select which mnemonics to use in the generated assembler code. **-mnew-mnemonics** requests output that uses the assembler mnemonics defined for the PowerPC architecture, while **-mold-mnemonics** requests 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 in use. Specifying **-mcpu=cpu_type** sometimes overrides the value of these option. Unless you are building a cross-compiler, you should normally not specify either **-mnew-mnemonics** or **-mold-mnemonics**, but should instead accept the

default.

-mcpu=cpu_type

Set architecture type, register usage, choice of mnemonics, and instruction scheduling parameters for machine type *cpu_type*. Supported values for *cpu_type* are **rios**, **rios1**, **rsc**, **rios2**, **rs64a**, **601**, **602**, **603**, **603e**, **604**, **604e**, **620**, **630**, **740**, **750**, **power**, **power2**, **powerpc**, **403**, **505**, **801**, **821**, **823**, and **860** and **common**. **-mcpu=power**, **-mcpu=power2**, **-mcpu=powerpc**, and **-mcpu=powerpc64** specify generic POWER, POWER2, pure 32-bit PowerPC (i.e., not MPC601), and 64-bit PowerPC architecture machine types, with an appropriate, generic processor model assumed for scheduling purposes.

Specifying any of the following options: **-mcpu=rios1**, **-mcpu=rios2**, **-mcpu=rsc**, **-mcpu=power**, or **-mcpu=power2** enables the **-mpower** option and disables the **-mpowerpc** option; **-mcpu=601** enables both the **-mpower** and **-mpowerpc** options. All of **-mcpu=rs64a**, **-mcpu=602**, **-mcpu=603**, **-mcpu=603e**, **-mcpu=604**, **-mcpu=620**, **-mcpu=630**, **-mcpu=740**, and **-mcpu=750** enable the **-mpowerpc** option and disable the **-mpower** option. Exactly similarly, all of **-mcpu=403**, **-mcpu=505**, **-mcpu=821**, **-mcpu=860** and **-mcpu=powerpc** enable the **-mpowerpc** option and disable the **-mpower** option. **-mcpu=common** disables both the **-mpower** and **-mpowerpc** options.

AIX versions 4 or greater selects **-mcpu=common** by default, so that code will operate on all members of the RS/6000 POWER and PowerPC families. In that case, 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. GCC assumes a generic processor model for scheduling purposes.

Specifying any of the options **-mcpu=rios1**, **-mcpu=rios2**, **-mcpu=rsc**, **-mcpu=power**, or **-mcpu=power2** also disables the **new-mnemonics** option. Specifying **-mcpu=601**, **-mcpu=602**, **-mcpu=603**, **-mcpu=603e**, **-mcpu=604**, **-mcpu=620**, **-mcpu=630**, **-mcpu=403**, **-mcpu=505**, **-mcpu=821**, **-mcpu=860** or **-mcpu=powerpc** also enables the **new-mnemonics** option.

Specifying **-mcpu=403**, **-mcpu=821**, or **-mcpu=860** also enables the **-msoft-float** option.

-mtune=cpu_type

Set the instruction scheduling parameters for machine type *cpu_type*, but do not set the architecture type, register usage, choice of mnemonics like **-mcpu=cpu_type** would. The same values for *cpu_type* are used for **-mtune=cpu_type** as for **-mcpu=cpu_type**. The **-mtune=cpu_type** option overrides the **-mcpu=cpu_type** option in terms of instruction scheduling parameters.

-mfull-toc

-mno-fp-in-toc

-mno-sum-in-toc

-mminimal-toc

Modify generation of the TOC (Table Of Contents), which is created for every executable file. The **-mfull-toc** option is selected by default. In that case, GCC will allocate at least one TOC entry for each unique non-automatic variable reference in your program. GCC will also place floating-point constants in the TOC. However, only 16,384 entries are available in the TOC.

If you receive a linker error message that saying you have overflowed the available TOC space, you can reduce the amount of TOC space used with the **-mno-fp-in-toc** and **-mno-sum-in-toc** options. **-mno-fp-in-toc** prevents GCC from putting floating-point constants in the TOC and **-mno-sum-in-toc** forces GCC to generate code to calculate the sum of an address and a constant at run-time instead of putting that sum into the TOC. You may specify one or both of these options. Each causes GCC to produce very slightly slower and larger code at the expense of conserving TOC space.

If you still run out of space in the TOC even when you specify both of these options, specify **-mminimal-toc** instead. This option causes GCC to make only one TOC entry for every file. When you specify this option, GCC will produce code that is slower and larger but which uses extremely little TOC space. You may wish to use this option only on files that contain less frequently executed code.

-maix64**-maix32**

Enable 64-bit AIX ABI and calling convention: 64-bit pointers, 64-bit long type, and the infrastructure needed to support them. Specifying **-maix64** implies **-mpowerpc64** and **-mpowerpc**, while **-maix32** disables the 64-bit ABI and implies **-mno-powerpc64**. GCC defaults to **-maix32**.

-mxl-call**-mno-xl-call**

On AIX, pass floating-point arguments to prototyped functions beyond the register save area (RSA) on the stack in addition to argument FPRs. The AIX calling convention was extended but not initially documented to handle an obscure K&R C case of calling a function that takes the address of its arguments with fewer arguments than declared. AIX XL compilers access floating point arguments which do not fit in the RSA from the stack when a subroutine is compiled without optimization. Because always storing floating-point arguments on the stack is inefficient and rarely needed, this option is not enabled by default and only is necessary when calling subroutines compiled by AIX XL compilers without optimization.

-mthreads

Support *AIX Threads*. Link an application written to use *pthread*s with special libraries and startup code to enable the application to run.

-mpe

Support *IBM RS/6000 SP Parallel Environment* (PE). Link an application written to use message passing 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 the *specs* file must be overridden with the **-specs=** option to specify the appropriate directory location. The Parallel Environment does not support threads, so the **-mpe** option and the **-mthreads** option are incompatible.

-msoft-float**-mhard-float**

Generate code that does not use (uses) the floating-point register set. Software floating point emulation is provided if you use the **-msoft-float** option, and pass the option to GCC when linking.

-mmultiple**-mno-multiple**

Generate code that uses (does not use) 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, since those instructions do not work when the processor is in little endian mode. The exceptions are PPC740 and PPC750 which permit the instructions usage in little endian mode.

-mstring**-mno-string**

Generate code that uses (does not use) 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, since those instructions do not work when the processor is in little endian mode. The exceptions are PPC740 and PPC750 which permit the instructions usage in little endian mode.

-mupdate**-mno-update**

Generate code that uses (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 **-mno-update**, there is a small window between the time that the stack pointer is updated and the address of the previous frame is stored, which means code that walks the stack frame across interrupts or signals may get corrupted data.

-mfused-madd**-mno-fused-madd**

Generate code that uses (does not use) the floating point multiply and accumulate instructions. These instructions are generated by default if hardware floating is used.

-mno-bit-align**-mbit-align**

On System V.4 and embedded PowerPC systems do not (do) 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 8 unsigned bitfields of length 1 would be aligned to a 4 byte boundary and have a size of 4 bytes. By using **-mno-bit-align**, the structure would be aligned to a 1 byte boundary and be one byte in size.

-mno-strict-align**-mstrict-align**

On System V.4 and embedded PowerPC systems do not (do) assume that unaligned memory references will be handled by the system.

-mrelocatable**-mno-relocatable**

On embedded PowerPC systems generate code that allows (does not allow) the program to be relocated to a different address at runtime. If you use **-mrelocatable** on any module, all objects linked together must be compiled with **-mrelocatable** or **-mrelocatable-lib**.

-mrelocatable-lib**-mno-relocatable-lib**

On embedded PowerPC systems generate code that allows (does not allow) the program to be relocated to a different address at runtime. Modules compiled with **-mrelocatable-lib** can be linked with either modules compiled without **-mrelocatable** and **-mrelocatable-lib** or with modules compiled with the **-mrelocatable** options.

-mno-toc**-mtoc**

On System V.4 and embedded PowerPC systems do not (do) assume that register 2 contains a pointer to a global area pointing to the addresses used in the program.

-mlittle**-mlittle-endian**

On System V.4 and embedded PowerPC systems compile code for the processor in little endian mode. The **-mlittle-endian** option is the same as **-mlittle**.

-mbig**-mbig-endian**

On System V.4 and embedded PowerPC systems compile code for the processor in big endian mode. The **-mbig-endian** option is the same as **-mbig**.

-mcall-sysv

On System V.4 and embedded PowerPC systems compile code using calling conventions that adheres to the March 1995 draft of the System V Application Binary Interface, PowerPC processor supplement. This is the default unless you configured GCC using **powerpc-*-eabiaix**.

-mcall-sysv-eabi

Specify both **-mcall-sysv** and **-meabi** options.

-mcall-sysv-noeabi

Specify both **-mcall-sysv** and **-mno-eabi** options.

-mcall-aix

On System V.4 and embedded PowerPC systems compile code using calling conventions that are similar to those used on AIX. This is the default if you configured GCC using **powerpc-*-eabiaix**.

-mcall-solaris

On System V.4 and embedded PowerPC systems compile code for the Solaris operating system.

-mcall-linux

On System V.4 and embedded PowerPC systems compile code for the Linux-based GNU system.

-mprototype**-mno-prototype**

On System V.4 and embedded PowerPC systems assume that all calls to variable argument functions are properly prototyped. Otherwise, the compiler must insert an instruction before every non prototyped call to set or clear bit 6 of the condition code register (*CR*) to indicate whether floating point values were passed in the floating point registers in case the function takes a variable arguments. With **-mprototype**, only calls to prototyped variable argument functions will set or clear the bit.

-msim

On embedded PowerPC systems, 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 **powerpc*-eabisim** configurations.

-mmvme

On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libmvme.a* and *libc.a*.

-mads

On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libads.a* and *libc.a*.

-myellowknife

On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libyk.a* and *libc.a*.

-mvxworks

On System V.4 and embedded PowerPC systems, specify that you are compiling for a VxWorks system.

-memb

On embedded PowerPC systems, set the *PPC_EMB* bit in the ELF flags header to indicate that **eabi** extended relocations are used.

-meabi**-mno-eabi**

On System V.4 and embedded PowerPC systems do (do not) adhere to the Embedded Applications Binary Interface (eabi) which is a set of modifications to the System V.4 specifications. Selecting **-meabi** 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. Selecting **-mno-eabi** means that the stack is aligned to a 16 byte boundary, do not call an initialization function from `main`, and the **-msdata** option will only use `r13` to point to a single small data area. The **-meabi** option is on by default if you configured GCC using one of the **powerpc*-eabi*** options.

-msdata=eabi

On System V.4 and embedded PowerPC systems, put small initialized `const` global and static data in the **.sdata2** section, which is pointed to by register `r2`. Put small initialized non-`const` global and static data in the **.sdata** section, which is pointed to by register `r13`. Put small uninitialized global and static data in the **.sbss** section, which is adjacent to the **.sdata** section. The **-msdata=eabi** option is incompatible with the **-mrelocatable** option. The **-msdata=eabi** option also sets the **-memb** option.

-msdata=sysv

On System V.4 and embedded PowerPC systems, put small global and static data in the **.sdata** section, which is pointed to by register `r13`. Put small uninitialized global and static data in the **.sbss** section, which is adjacent to the **.sdata** section. The **-msdata=sysv** option is incompatible with the

-mrelocatable option.

-msdata=default

-msdata

On System V.4 and embedded PowerPC systems, if **-meabi** is used, compile code the same as **-msdata=eabi**, otherwise compile code the same as **-msdata=sysv**.

-msdata-data

On System V.4 and embedded PowerPC systems, put small global and static data in the **.sdata** section. Put small uninitialized global and static data in the **.sbss** section. Do not use register r13 to address small data however. This is the default behavior unless other **-msdata** options are used.

-msdata=none

-mno-sdata

On embedded PowerPC systems, put all initialized global and static data in the **.data** section, and all uninitialized data in the **.bss** section.

-G num

On embedded PowerPC systems, put global and static items less than or equal to *num* bytes into the small data or bss sections instead of the normal data or bss section. By default, *num* is 8. The **-G num** switch is also passed to the linker. All modules should be compiled with the same **-G num** value.

-mregnames

-mno-regnames

On System V.4 and embedded PowerPC systems do (do not) emit register names in the assembly language output using symbolic forms.

IBM RT Options

These **-m** options are defined for the IBM RT PC:

-min-line-mul

Use an in-line code sequence for integer multiplies. This is the default.

-mcall-lib-mul

Call `lmul$$` for integer multiples.

-mfull-fp-blocks

Generate full-size floating point data blocks, including the minimum amount of scratch space recommended by IBM. This is the default.

-mminimum-fp-blocks

Do not include extra scratch space in floating point data blocks. This results in smaller code, but slower execution, since scratch space must be allocated dynamically.

-mfp-arg-in-fpregs

Use a calling sequence incompatible with the IBM calling convention in which floating point arguments are passed in floating point registers. Note that `varargs.h` and `stdarg.h` will not work with floating point operands if this option is specified.

-mfp-arg-in-gregs

Use the normal calling convention for floating point arguments. This is the default.

-mhc-struct-return

Return structures of more than one word in memory, rather than in a register. This provides compatibility with the MetaWare HighC (hc) compiler. Use the option **-fpcc-struct-return** for compatibility with the Portable C Compiler (pcc).

-mnohc-struct-return

Return some structures of more than one word in registers, when convenient. This is the default. For compatibility with the IBM-supplied compilers, use the option **-fpcc-struct-return** or the option **-mhc-struct-return**.

MIPS Options

These **-m** options are defined for the MIPS family of computers:

-mcpu=cpu type

Assume the defaults for the machine type *cpu type* when scheduling instructions. The choices for *cpu type* are **r2000**, **r3000**, **r3900**, **r4000**, **r4100**, **r4300**, **r4400**, **r4600**, **r4650**, **r5000**, **r6000**, **r8000**, and **orion**. Additionally, the **r2000**, **r3000**, **r4000**, **r5000**, and **r6000** can be abbreviated as **r2k** (or **r2K**), **r3k**, etc. While picking a specific *cpu type* will schedule things appropriately for that particular chip, the compiler will not generate any code that does not meet level 1 of the MIPS ISA (instruction set architecture) without a **-mipsX** or **-mabi** switch being used.

-mips1

Issue instructions from level 1 of the MIPS ISA. This is the default. **r3000** is the default *cpu type* at this ISA level.

-mips2

Issue instructions from level 2 of the MIPS ISA (branch likely, square root instructions). **r6000** is the default *cpu type* at this ISA level.

-mips3

Issue instructions from level 3 of the MIPS ISA (64 bit instructions). **r4000** is the default *cpu type* at this ISA level.

-mips4

Issue instructions from level 4 of the MIPS ISA (conditional move, prefetch, enhanced FPU instructions). **r8000** is the default *cpu type* at this ISA level.

-mfp32

Assume that 32 32-bit floating point registers are available. This is the default.

-mfp64

Assume that 32 64-bit floating point registers are available. This is the default when the **-mips3** option is used.

-mgp32

Assume that 32 32-bit general purpose registers are available. This is the default.

-mgp64

Assume that 32 64-bit general purpose registers are available. This is the default when the **-mips3** option is used.

-mint64

Force int and long types to be 64 bits wide. See **-mlong32** for an explanation of the default, and the width of pointers.

-mlong64

Force long types to be 64 bits wide. See **-mlong32** for an explanation of the default, and the width of pointers.

-mlong32

Force long, int, and pointer types to be 32 bits wide.

If none of **-mlong32**, **-mlong64**, or **-mint64** are set, the size of ints, longs, and pointers depends on the ABI and ISA chosen. For **-mabi=32**, and **-mabi=n32**, ints and longs are 32 bits wide. For **-mabi=64**, ints are 32 bits, and longs are 64 bits wide. For **-mabi=eabi** and either **-mips1** or **-mips2**, ints and longs are 32 bits wide. For **-mabi=eabi** and higher ISAs, ints are 32 bits, and longs are 64 bits wide. The width of pointer types is the smaller of the width of longs or the width of general purpose registers (which in turn depends on the ISA).

-mabi=32

-mabi=o64

-mabi=n32**-mabi=64****-mabi=eabi**

Generate code for the indicated ABI. The default instruction level is **-mips1** for **32**, **-mips3** for **n32**, and **-mips4** otherwise. Conversely, with **-mips1** or **-mips2**, the default ABI is **32**; otherwise, the default ABI is **64**.

-mmips-as

Generate code for the MIPS assembler, and invoke *mips-tfile* to add normal debug information. This is the default for all platforms except for the OSF/1 reference platform, using the OSF/rose object format. If either of the **-gstabs** or **-gstabs+** switches are used, the *mips-tfile* program will encapsulate the stabs within MIPS ECOFF.

-mgas

Generate code for the GNU assembler. This is the default on the OSF/1 reference platform, using the OSF/rose object format. Also, this is the default if the configure option **—with-gnu-as** is used.

-msplit-addresses**-mno-split-addresses**

Generate code to load 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.

-mrnames**-mno-rnames**

The **-mrnames** switch says to output code using the MIPS software names for the registers, instead of the hardware names (ie, *a0* instead of *\$4*). The only known assembler that supports this option is the Algorithmics assembler.

-mgpopt**-mno-gpopt**

The **-mgpopt** switch says 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 on by default if optimization is selected.

-mstats**-mno-stats**

For each non-inline function processed, the **-mstats** switch causes the compiler to emit one line to the standard error file to print statistics about the program (number of registers saved, stack size, etc.).

-mmemcpy**-mno-memcpy**

The **-mmemcpy** switch makes all block moves call the appropriate string function (**memcpy** or **bcopy**) instead of possibly generating inline code.

-mmips-tfile**-mno-mips-tfile**

The **-mno-mips-tfile** switch causes the compiler not postprocess the object file with the *mips-tfile* program, after the MIPS assembler has generated it to add debug support. If *mips-tfile* is not run, then no local variables will be available to the debugger. In addition, *stage2* and *stage3* objects will have the temporary file names 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.

-msoft-float

Generate output containing library calls for floating point. **Warning:** the requisite libraries are not part of GCC. Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

-mhard-float

Generate output containing floating point instructions. This is the default if you use the unmodified sources.

-mabicalls**-mno-abicalls**

Emit (or do not emit) the pseudo operations **.abicalls**, **.cpload**, and **.cpstore** that some System V.4 ports use for position independent code.

-mlong-calls**-mno-long-calls**

Do all calls with the **JALR** instruction, which requires loading up a function's address into a register before the call. You need to use this switch, if you call outside of the current 512 megabyte segment to functions that are not through pointers.

-mhalf-pic**-mno-half-pic**

Put pointers to extern references into the data section and load them up, rather than put the references in the text section.

-membedded-pic**-mno-embedded-pic**

Generate PIC code suitable for some embedded systems. All calls are made using PC relative address, and all data is addressed using the **\$gp** register. No more than 65536 bytes of global data may be used. This requires GNU as and GNU ld which do most of the work. This currently only works on targets which use ECOFF; it does not work with ELF.

-membedded-data**-mno-embedded-data**

Allocate variables to the read-only data section first if possible, then next in the small data section if possible, otherwise in data. This gives slightly slower code than the default, but reduces the amount of RAM required when executing, and thus may be preferred for some embedded systems.

-muninit-const-in-rodata**-mno-uninit-const-in-rodata**

When used together with **-membedded-data**, it will always store uninitialized const variables in the read-only data section.

-msingle-float**-mdouble-float**

The **-msingle-float** switch tells gcc to assume that the floating point coprocessor only supports single precision operations, as on the **r4650** chip. The **-mdouble-float** switch permits gcc to use double precision operations. This is the default.

-mmad**-mno-mad**

Permit use of the **mad**, **madu** and **mul** instructions, as on the **r4650** chip.

-m4650

Turns on **-msingle-float**, **-mmad**, and, at least for now, **-mcpu=r4650**.

-mips16**-mno-mips16**

Enable 16-bit instructions.

-mentry

Use the entry and exit pseudo ops. This option can only be used with **-mips16**.

-EL

Compile code for the processor in little endian mode. The requisite libraries are assumed to exist.

-EB

Compile code for the processor in big endian mode. The requisite libraries are assumed to exist.

-G *num*

Put global and static items less than or equal to *num* bytes into the small data or bss sections instead of the normal data or bss section. This allows the assembler to emit one word memory reference instructions based on the global pointer (*gp* or *\$28*), instead of the normal two words used. 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. All modules should be compiled with the same **-G *num*** value.

-nocpp

Tell the MIPS assembler to not run its preprocessor over user assembler files (with a *.s* suffix) when assembling them.

-mfix7000

Pass an option to gas which will cause nops to be inserted if the read of the destination register of an *mfhi* or *mflo* instruction occurs in the following two instructions.

-no-crt0

Do not include the default *crt0*.

Intel 386 Options

These **-m** options are defined for the i386 family of computers:

-mcpu=*cpu type*

Assume the defaults for the machine type *cpu type* when scheduling instructions. The choices for *cpu type* are **i386**, **i486**, **i586**, **i686**, **pentium**, **pentiumpro**, **k6**, and **athlon**.

While picking a specific *cpu type* will schedule things appropriately for that particular chip, the compiler will not generate any code that does not run on the i386 without the **-march=*cpu type*** option being used. **i586** is equivalent to **pentium** and **i686** is equivalent to **pentiumpro**. **k6** is the AMD chip as opposed to the Intel ones.

-march=*cpu type*

Generate instructions for the machine type *cpu type*. The choices for *cpu type* are the same as for **-mcpu**. Moreover, specifying **-march=*cpu type*** implies **-mcpu=*cpu type***.

-m386**-m486****-mpentium****-mpentiumpro**

Synonyms for **-mcpu=i386**, **-mcpu=i486**, **-mcpu=pentium**, and **-mcpu=pentiumpro** respectively. These synonyms are deprecated.

-mintel-syntax

Emit assembly using Intel syntax opcodes instead of AT&T syntax.

-mieee-fp**-mno-ieee-fp**

Control whether or not the compiler uses IEEE floating point comparisons. These handle correctly the case where the result of a comparison is unordered.

-msoft-float

Generate output containing library calls for floating point. **Warning:** the requisite libraries are not part of GCC. Normally the facilities of the machine's usual C compiler are used, but this can't be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation.

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.

-mno-fp-ret-in-387

Do not use the FPU registers for return values of functions.

The usual calling convention has functions return values of types `float` and `double` in an FPU register, even if there is no FPU. The idea is that the operating system should emulate an FPU.

The option **-mno-fp-ret-in-387** causes such values to be returned in ordinary CPU registers instead.

-mno-fancy-math-387

Some 387 emulators do not support the `sin`, `cos` and `sqrt` instructions for the 387. Specify this option to avoid generating those instructions. This option is the default on FreeBSD. As of revision 2.6.1, these instructions are not generated unless you also use the **-ffast-math** switch.

-malign-double**-mno-align-double**

Control whether GCC aligns `double`, `long double`, and `long long` variables on a two word boundary or a one word boundary. Aligning `double` variables on a two word boundary will produce code that runs somewhat faster on a **Pentium** at the expense of more memory.

-m128bit-long-double**-m128bit-long-double**

Control the size of `long double` type. i386 application binary interface specify the size to be 12 bytes, while modern architectures (Pentium and newer) prefer `long double` aligned to 8 or 16 byte boundary. This is impossible to reach with 12 byte long doubles in the array accesses.

Warning: if you use the **-m128bit-long-double** switch, the structures and arrays containing `long double` will change their size as well as function calling convention for function taking `long double` will be modified.

-m96bit-long-double**-m96bit-long-double**

Set the size of `long double` to 96 bits as required by the i386 application binary interface. This is the default.

-msvr3-shlib**-mno-svr3-shlib**

Control whether GCC places uninitialized locals into `bss` or `data`. **-msvr3-shlib** places these locals into `bss`. These options are meaningful only on System V Release 3.

-mno-wide-multiply**-mwide-multiply**

Control whether GCC uses the `mul` and `imul` that produce 64 bit results in `eax:edx` from 32 bit operands to do `long long` multiplies and 32-bit division by constants.

-mrtd

Use a different function-calling convention, in which functions that take a fixed number of arguments return with the `ret num` instruction, which pops their arguments while returning. This saves one instruction in the caller since there is no need to pop the arguments there.

You can specify that an individual function is called with this calling sequence with the function attribute **stdcall**. You can also override the **-mrtd** option by using the function attribute **cdecl**.

Warning: this calling convention is incompatible with the one normally used on Unix, so you cannot use it if you need to call libraries compiled with the Unix compiler.

Also, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`); otherwise incorrect code will be generated for calls to those functions.

In addition, seriously incorrect code will result if you call a function with too many arguments. (Normally, extra arguments are harmlessly ignored.)

-mregparm=*num*

Control how many registers are used to pass integer arguments. By default, no registers are used to pass arguments, and at most 3 registers can be used. You can control this behavior for a specific function by using the function attribute **regparm**.

Warning: if you use this switch, and *num* is nonzero, then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

-malign-loops=*num*

Align loops to a 2 raised to a *num* byte boundary. If **-malign-loops** is not specified, the default is 2 unless gas 2.8 (or later) is being used in which case the default is to align the loop on a 16 byte boundary if it is less than 8 bytes away.

-malign-jumps=*num*

Align instructions that are only jumped to to a 2 raised to a *num* byte boundary. If **-malign-jumps** is not specified, the default is 2 if optimizing for a 386, and 4 if optimizing for a 486 unless gas 2.8 (or later) is being used in which case the default is to align the instruction on a 16 byte boundary if it is less than 8 bytes away.

-malign-functions=*num*

Align the start of functions to a 2 raised to *num* byte boundary. If **-malign-functions** is not specified, the default is 2 if optimizing for a 386, and 4 if optimizing for a 486.

-mpreferred-stack-boundary=*num*

Attempt to keep the stack boundary aligned to a 2 raised to *num* byte boundary. If **-mpreferred-stack-boundary** is not specified, the default is 4 (16 bytes or 128 bits).

The stack is required to be aligned on a 4 byte boundary. On Pentium and PentiumPro, `double` and `long double` values should be aligned to an 8 byte boundary (see **-malign-double**) or suffer significant run time performance penalties. On Pentium III, the Streaming SIMD Extension (SSE) data type `__m128` suffers similar penalties if it is not 16 byte aligned.

To ensure proper alignment of this values on the stack, the stack boundary must be as aligned as that required by any value stored on the stack. Further, every function must be generated such that it keeps the stack aligned. Thus calling a function compiled with a higher preferred stack boundary from a function compiled with a lower preferred stack boundary will most likely misalign the stack. It is recommended that libraries that use callbacks always use the default setting.

This extra alignment does consume extra stack space. 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**.

-mpush-args

Use PUSH operations to store outgoing parameters. This method is shorter and usually equally fast as method using SUB/MOV operations and is enabled by default. In some cases disabling it may improve performance because of improved scheduling and reduced dependencies.

-maccumulate-outgoing-args

If enabled, the maximum amount of space required for outgoing arguments will be computed in the function prologue. This is faster on most modern CPUs because of reduced dependencies, improved scheduling and reduced stack usage when preferred stack boundary is not equal to 2. The drawback is a notable increase in code size. This switch implies **-mno-push-args**.

-mthreads

Support thread-safe exception handling on **Mingw32**. Code that relies on thread-safe exception handling must compile and link all code with the **-mthreads** option. When compiling, **-mthreads** defines **-D_MT**; when linking, it links in a special thread helper library **-lmingwthrd** which cleans up per thread exception handling data.

–mno-align-stringops

Do not align destination of inlined string operations. This switch reduces code size and improves performance in case the destination is already aligned, but gcc don't know about it.

–minline-all-stringops

By default GCC inlines string operations only when destination is known to be aligned at least to 4 byte boundary. This enables more inlining, increase code size, but may improve performance of code that depends on fast memcpy, strlen and memset for short lengths.

HPPA Options

These **–m** options are defined for the HPPA family of computers:

–march=architecture type

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. Refer to */usr/lib/sched.models* on an HP-UX system to determine the proper architecture option for your machine. Code compiled for lower numbered architectures will run on higher numbered architectures, but not the other way around.

PA 2.0 support currently requires gas snapshot 19990413 or later. The next release of binutils (current is 2.9.1) will probably contain PA 2.0 support.

–mpa-risc-1-0**–mpa-risc-1-1****–mpa-risc-2-0**

Synonyms for **–march=1.0**, **–march=1.1**, and **–march=2.0** respectively.

–mbig-switch

Generate code suitable for big switch tables. Use this option only if the assembler/linker complain about out of range branches within a switch table.

–mjump-in-delay

Fill delay slots of function calls with unconditional jump instructions by modifying the return pointer for the function call to be the target of the conditional jump.

–mdisable-fpregs

Prevent floating point registers from being used in any manner. This is necessary for compiling kernels which perform lazy context switching of floating point registers. If you use this option and attempt to perform floating point operations, the compiler will abort.

–mdisable-indexing

Prevent the compiler from using indexing address modes. This avoids some rather obscure problems when compiling MIG generated code under MACH.

–mno-space-regs

Generate code that assumes the target has no space registers. This allows GCC to generate faster indirect calls and use unscaled index address modes.

Such code is suitable for level 0 PA systems and kernels.

–mfast-indirect-calls

Generate code that assumes calls never cross space boundaries. This allows GCC to emit code which performs faster indirect calls.

This option will not work in the presence of shared libraries or nested functions.

–mlong-load-store

Generate 3-instruction load and store sequences as sometimes required by the HP-UX 10 linker. This is equivalent to the **+k** option to the HP compilers.

–mportable-runtime

Use the portable calling conventions proposed by HP for ELF systems.

-mgas

Enable the use of assembler directives only GAS understands.

-mschedule=*cpu type*

Schedule code according to the constraints for the machine type *cpu type*. The choices for *cpu type* are **700 7100, 7100LC, 7200**, and **8000**. Refer to */usr/lib/sched.models* on an HP-UX system to determine the proper scheduling option for your machine.

-mlinker-opt

Enable the optimization pass in the HP-UX linker. Note this makes symbolic debugging impossible. It also triggers a bug in the HP-UX 8 and HP-UX 9 linkers in which they give bogus error messages when linking some programs.

-msoft-float

Generate output containing library calls for floating point. **Warning:** the requisite libraries are not available for all HPPA targets. Normally the facilities of the machine's usual C compiler are used, but this cannot be done directly in cross-compilation. You must make your own arrangements to provide suitable library functions for cross-compilation. The embedded target **hppa1.1-*-pro** does provide software floating point support.

-msoft-float changes the calling convention in the output file; therefore, it is only useful if you compile *all* of a program with this option. In particular, you need to compile *libgcc.a*, the library that comes with GCC, with **-msoft-float** in order for this to work.

Intel 960 Options

These **-m** options are defined for the Intel 960 implementations:

-mcpu type

Assume the defaults for the machine type *cpu type* for some of the other options, including instruction scheduling, floating point support, and addressing modes. The choices for *cpu type* are **ka, kb, mc, ca, cf, sa**, and **sb**. The default is **kb**.

-mnumerics**-msoft-float**

The **-mnumerics** option indicates that the processor does support floating-point instructions. The **-msoft-float** option indicates that floating-point support should not be assumed.

-mleaf-procedures**-mno-leaf-procedures**

Do (or do not) 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 using a linker that doesn't support this optimization.

-mtail-call**-mno-tail-call**

Do (or do not) 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. The default is **-mno-tail-call**.

-mcomplex-addr**-mno-complex-addr**

Assume (or do not 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, but they definitely are on the C-series. The default is currently **-mcomplex-addr** for all processors except the CB and CC.

-mcode-align**-mno-code-align**

Align code to 8-byte boundaries for faster fetching (or don't bother). Currently turned on by default for C-series implementations only.

–mic-compat**–mic2.0-compat****–mic3.0-compat**

Enable compatibility with iC960 v2.0 or v3.0.

–masm-compat**–mintel-asm**

Enable compatibility with the iC960 assembler.

–mstrict-align**–mno-strict-align**

Do not permit (do permit) unaligned accesses.

–mold-align

Enable structure-alignment compatibility with Intel's gcc release version 1.3 (based on gcc 1.37). This option implies **–mstrict-align**.

–mlong-double-64

Implement type **long double** as 64-bit floating point numbers. Without the option **long double** is implemented by 80-bit floating point numbers. The only reason we have it because there is no 128-bit **long double** support in **fp-bit.c** yet. So it is only useful for people using soft-float targets. Otherwise, we should recommend against use of it.

DEC Alpha Options

These **–m** options are defined for the DEC Alpha implementations:

–mno-soft-float**–msoft-float**

Use (do not use) the hardware floating-point instructions for floating-point operations. When **–msoft-float** is specified, functions in *libgcc1.c* will be used to perform floating-point operations. Unless they are replaced by routines that emulate the floating-point operations, or compiled in such a way as to call such emulations routines, these routines will issue floating-point operations. If you are compiling for an Alpha without floating-point operations, you must ensure that the library is built so as not to call them.

Note that Alpha implementations without floating-point operations are required to have floating-point registers.

–mfp-reg**–mno-fp-regs**

Generate code that uses (does not use) the floating-point register set. **–mno-fp-regs** implies **–msoft-float**. If 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 non-standard 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.

A typical use of this option is building a kernel that does not use, and hence need not save and restore, any floating-point registers.

–mieee

The Alpha architecture implements floating-point hardware optimized for maximum performance. It is mostly compliant with the IEEE floating point standard. However, for full compliance, software assistance is required. This option generates code fully IEEE compliant code *except* that the *inexact flag* is not maintained (see below). If this option is turned on, the CPP macro `__IEEE_FP` is defined during compilation. The option is a shorthand for: **–D_IEEE_FP –mfp-trap-mode=su –mtrap-precision=i –mieee-conformant**. 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. Other Alpha compilers call this option **–ieee_with_no_inexact**.

-mieee-with-inexact

This is like **-mieee** except the generated code also maintains the IEEE *inexact flag*. Turning on this option causes the generated code to implement fully-compliant IEEE math. The option is a shorthand for **-D_IEEE_FP -D_IEEE_FP_INEXACT** plus the three following: **-mieee-conformant**, **-mfp-trap-mode=sui**, and **-mtrap-precision=i**. On some Alpha implementations the resulting code may execute significantly slower than the code generated by default. Since there is very little code that depends on the *inexact flag*, you should normally not specify this option. Other Alpha compilers call this option **-ieeee_with_inexact**.

-mfp-trap-mode=trap mode

This option controls what floating-point related traps are enabled. Other Alpha compilers call this option **-fptm trap mode**. The trap mode can be set to one of four values:

- n** This is the default (normal) setting. The only traps that are enabled are the ones that cannot be disabled in software (e.g., division by zero trap).
- u** In addition to the traps enabled by **n**, underflow traps are enabled as well.
- su** Like **su**, but the instructions are marked to be safe for software completion (see Alpha architecture manual for details).
- sui** Like **su**, but inexact traps are enabled as well.

-mfp-rounding-mode=rounding mode

Selects the IEEE rounding mode. Other Alpha compilers call this option **-fprm rounding mode**. The *rounding mode* can be one of:

- n** Normal IEEE rounding mode. Floating point numbers are rounded towards the nearest machine number or towards the even machine number in case of a tie.
- m** Round towards minus infinity.
- c** Chopped rounding mode. Floating point numbers are rounded towards zero.
- d** Dynamic rounding mode. A field in the floating point control register (*fpcr*, see Alpha architecture reference manual) controls the rounding mode in effect. The C library initializes this register for rounding towards plus infinity. Thus, unless your program modifies the *fpcr*, **d** corresponds to round towards plus infinity.

-mtrap-precision=trap precision

In the Alpha architecture, floating point traps are imprecise. This means without software assistance it is impossible to recover from a floating trap and program execution normally needs to be terminated. GCC can generate code that can assist operating system trap handlers in determining the exact location that caused a floating point trap. Depending on the requirements of an application, different levels of precisions can be selected:

- p** Program precision. This option is the default and means a trap handler can only identify which program caused a floating point exception.
- 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.

Other Alpha compilers provide the equivalent options called **-scope_safe** and **-resumption_safe**.

-mieee-conformant

This option marks the generated code as 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**. Its only effect is to emit the line **.eflag 48** in the function prologue of the generated assembly file. Under DEC Unix, this has the effect that IEEE-conformant math library routines will be linked in.

-mbuild-constants

Normally GCC examines a 32- or 64-bit integer constant to see if it can construct it from smaller constants in two or three instructions. If it cannot, it will output the constant as a literal and generate code to load it from the data segment at runtime.

Use this option to require GCC to construct *all* integer constants using code, even if it takes more instructions (the maximum is six).

You would typically use this option to build a shared library dynamic loader. If itself a shared library, it must relocate itself in memory before it can find the variables and constants in its own data segment.

-malpha-as**-mgas**

Select whether to generate code to be assembled by the vendor-supplied assembler (**-malpha-as**) or by the GNU assembler **-mgas**.

-mbwx**-mno-bwx****-mcix****-mno-cix****-mmax****-mno-max**

Indicate whether GCC should generate code to use the optional BWX, CIX, and MAX instruction sets. The default is to use the instruction sets supported by the CPU type specified via **-mcpu=** option or that of the CPU on which GCC was built if none was specified.

-mcpu=cpu_type

Set the instruction set, register set, and instruction scheduling parameters for machine type *cpu_type*. You can specify either the **EV** style name or the corresponding chip number. GCC supports scheduling parameters for the EV4 and EV5 family of processors and will choose the default values for the instruction set from the processor you specify. If you do not specify a processor type, GCC will default to the processor on which the compiler was built.

Supported values for *cpu_type* are

ev4**21064**

Schedules as an EV4 and has no instruction set extensions.

ev5**21164**

Schedules as an EV5 and has no instruction set extensions.

ev56**21164a**

Schedules as an EV5 and supports the BWX extension.

pca56**21164pc****21164PC**

Schedules as an EV5 and supports the BWX and MAX extensions.

ev6**21264**

Schedules as an EV5 (until Digital releases the scheduling parameters for the EV6) and supports the BWX, CIX, and MAX extensions.

-mmemory-latency=time

Sets the latency the scheduler should assume for typical memory references as seen by the application. This number is highly dependent on the memory access patterns used by the application and the size of the external cache on the machine.

Valid options for *time* are

number

A decimal number representing clock cycles.

L1

L2

L3

main

The compiler contains estimates of the number of clock cycles for “typical” EV4 & EV5 hardware for the Level 1, 2 & 3 caches (also called Dcache, Scache, and Bcache), as well as to main memory. Note that L3 is only valid for EV5.

Clipper Options

These **-m** options are defined for the Clipper implementations:

-mc300

Produce code for a C300 Clipper processor. This is the default.

-mc400

Produce code for a C400 Clipper processor i.e. use floating point registers f8..f15.

H8/300 Options

These **-m** options are defined for the H8/300 implementations:

-mrelax

Shorten some address references at link time, when possible; uses the linker option **-relax**.

-mh

Generate code for the H8/300H.

-ms

Generate code for the H8/S.

-ms2600

Generate code for the H8/S2600. This switch must be used with **-ms**.

-mint32

Make `int` data 32 bits by default.

-malign-300

On the H8/300H and H8/S, 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. **-malign-300** causes them to be aligned on 2 byte boundaries. This option has no effect on the H8/300.

SH Options

These **-m** options are defined for the SH implementations:

-m1

Generate code for the SH1.

-m2

Generate code for the SH2.

-m3

Generate code for the SH3.

-m3e

Generate code for the SH3e.

-m4-nofpu

Generate code for the SH4 without a floating-point unit.

-m4-single-only

Generate code for the SH4 with a floating-point unit that only supports single-precision arithmetic.

-m4-single

Generate code for the SH4 assuming the floating-point unit is in single-precision mode by default.

-m4

Generate code for the SH4.

-mb

Compile code for the processor in big endian mode.

-ml

Compile code for the processor in little endian mode.

-mdalign

Align doubles at 64 bit boundaries. Note that this changes the calling conventions, and thus some functions from the standard C library will not work unless you recompile it first with **-mdalign**.

-mrelax

Shorten some address references at link time, when possible; uses the linker option **-relax**.

-mbigtable

Use 32-bit offsets in switch tables. The default is to use 16-bit offsets.

-mfmovd

Enable the use of the instruction `fmovd`.

-mhitachi

Comply with the calling conventions defined by Hitachi.

-mnomacsave

Mark the MAC register as call-clobbered, even if **-mhitachi** is given.

-misize

Dump instruction size and location in the assembly code.

-mpadstruct

This option is deprecated. It pads structures to multiple of 4 bytes, which is incompatible with the SH ABI.

-mspace

Optimize for space instead of speed. Implied by **-Os**.

-mprefergot

When generating position-independent code, emit function calls using the Global Offset Table instead of the Procedure Linkage Table.

-musermode

Generate a library function call to invalidate instruction cache entries, after fixing up a trampoline. This library function call doesn't assume it can write to the whole memory address space. This is the default when the target is `sh-*-linux*`.

Options for System V

These additional options are available on System V Release 4 for compatibility with other compilers on those systems:

-G Create a shared object. It is recommended that **-symbolic** or **-shared** be used instead.

-Qy

Identify the versions of each tool used by the compiler, in a `.ident` assembler directive in the output.

-Qn

Refrain from adding `.ident` directives to the output file (this is the default).

-YP,*dirs*

Search the directories *dirs*, and no others, for libraries specified with **-l**.

-Ym,*dir*

Look in the directory *dir* to find the M4 preprocessor. The assembler uses this option.

TMS320C3x/C4x Options

These **-m** options are defined for TMS320C3x/C4x implementations:

-mcpu=*cpu_type*

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** to generate code for the TMS320C40.

-mbig-memory**-mbig****-msmall-memory****-msmall**

Generates code for the big or small memory model. The small memory model assumed that all data fits into one 64K word page. At run-time the data page (DP) register must be set to point to the 64K page containing the .bss and .data program sections. The big memory model is the default and requires reloading of the DP register for every direct memory access.

-mbk**-mno-bk**

Allow (disallow) allocation of general integer operands into the block count register BK.

-mdb**-mno-db**

Enable (disable) generation of code using decrement and branch, *DBcond*(D), instructions. This is enabled by default for the C4x. To be on the safe side, this is disabled for the C3x, since the maximum iteration count on the C3x is $2^{23} + 1$ (but who iterates loops more than 2^{23} times on the C3x?). Note that GCC will try to reverse a loop so that it can utilise 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 utilised.

-mdp-isr-reload**-mparanoid**

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 required unless someone has violated the small memory model by modifying the DP register, say within an object library.

-mmpyi**-mno-mpyi**

For the C3x 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, then the multiplication will be performed using shifts and adds. If the **-mmpyi** option is not specified for the C3x, then squaring operations are performed inline instead of a library call.

-mfast-fix**-mno-fast-fix**

The C3x/C4x FIX instruction to convert 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 is necessary to detect and correct this case. This option can be used to disable generation of the additional code required to correct the result.

-mrptb**-mno-rptb**

Enable (disable) 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 having nested RPTB loops due to the overhead required to save and restore the RC, RS, and RE registers. This is enabled by default with `-O2`.

-mrpts=*count***-mno-rpts**

Enable (disable) 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 a RPTS instruction instead of a RPTB. If no value is specified, then a RPTS will be emitted even if the loop count cannot be determined at compile time. Note that the repeated instruction following RPTS does not have to be reloaded from memory each iteration, thus freeing up the CPU buses for operands. However, since interrupts are blocked by this instruction, it is disabled by default.

-mloop-unsigned**-mno-loop-unsigned**

The maximum iteration count when using RPTS and RPTB (and DB on the C40) is $2^{31} + 1$ since these instructions test if the iteration count is negative to terminate the loop. If the iteration count is unsigned there is a possibility than the $2^{31} + 1$ maximum iteration count may be exceeded. This switch allows an unsigned iteration count.

-mti

Try to emit an assembler syntax that the TI assembler (asm30) is happy with. This also enforces compatibility with the API employed by the TI C3x C compiler. For example, long doubles are passed as structures rather than in floating point registers.

-mregparm**-mmemparm**

Generate code that uses registers (stack) for passing arguments to functions. By default, arguments are passed in registers where possible rather than by pushing arguments on to the stack.

-mparallel-insns**-mno-parallel-insns**

Allow the generation of parallel instructions. This is enabled by default with `-O2`.

-mparallel-mpy**-mno-parallel-mpy**

Allow the generation of `MPY||ADD` and `MPY||SUB` parallel instructions, provided `-mparallel-insns` is also specified. These instructions have tight register constraints which can pessimize the code generation of large functions.

V850 Options

These `-m` options are defined for V850 implementations:

-mlong-calls**-mno-long-calls**

Treat all calls as being far away (near). If calls are assumed to be far away, the compiler will always load the functions address up into a register, and call indirect through the pointer.

-mno-ep**-mep**

Do not optimize (do optimize) basic blocks that use the same index pointer 4 or more times to copy pointer into the `ep` register, and use the shorter `sld` and `sst` instructions. The `-mep` option is on by default if you optimize.

-mno-prolog-function

-mprolog-function

Do not use (do use) external functions to save and restore registers at the prolog and epilog of a function. The external functions are slower, but use less code space if more than one function saves the same number of registers. The **-mprolog-function** option is on by default if you optimize.

-mspace

Try to make the code as small as possible. At present, this just turns on the **-mep** and **-mprolog-function** options.

-mtda=*n*

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).

-msda=*n*

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 64 kilobytes.

-mzda=*n*

Put static or global variables whose size is *n* bytes or less into the first 32 kilobytes of memory.

-mv850

Specify that the target processor is the V850.

-mbig-switch

Generate code suitable for big switch tables. Use this option only if the assembler/linker complain about out of range branches within a switch table.

ARC Options

These options are defined for ARC implementations:

-EL

Compile code for little endian mode. This is the default.

-EB

Compile code for big endian mode.

-mmangle-cpu

Prepend the name of the *cpu* to all public symbol names. In multiple-processor systems, there are many ARC variants with different instruction and register set characteristics. This flag prevents code compiled for one *cpu* to be linked with code compiled for another. No facility exists for handling variants that are “almost identical”. This is an all or nothing option.

-mcpu=*cpu*

Compile code for ARC variant *cpu*. Which variants are supported depend on the configuration. All variants support **-mcpu=base**, this is the default.

-mtext=*text section***-mdata=*data section*****-mrodata=*readonly data section***

Put functions, data, and readonly data in *text section*, *data section*, and *readonly data section* respectively by default. This can be overridden with the `section` attribute.

NS32K Options

These are the **-m** options defined for the 32000 series. The default values for these options depends on which style of 32000 was selected when the compiler was configured; the defaults for the most common choices are given below.

-m32032**-m32032**

Generate output for a 32032. This is the default when the compiler is configured for 32032 and 32016 based systems.

-m32332**-m32332**

Generate output for a 32332. This is the default when the compiler is configured for 32332-based systems.

-m32532**-m32532**

Generate output for a 32532. This is the default when the compiler is configured for 32532-based systems.

-m32081

Generate output containing 32081 instructions for floating point. This is the default for all systems.

-m32381

Generate output containing 32381 instructions for floating point. This also implies **-m32081**. The 32381 is only compatible with the 32332 and 32532 cpus. This is the default for the pc532-netbsd configuration.

-mmulti-add

Try and generate multiply-add floating point instructions `polyF` and `dotF`. This option is only available if the **-m32381** option is in effect. Using these instructions requires changes to register allocation which generally has a negative impact on performance. This option should only be enabled when compiling code particularly likely to make heavy use of multiply-add instructions.

-mnomulti-add

Do not try and generate multiply-add floating point instructions `polyF` and `dotF`. This is the default on all platforms.

-msoft-float

Generate output containing library calls for floating point. **Warning:** the requisite libraries may not be available.

-mnobitfield

Do not use the bit-field instructions. On some machines it is faster to use shifting and masking operations. This is the default for the pc532.

-mbitfield

Do use the bit-field instructions. This is the default for all platforms except the pc532.

-mrtd

Use a different function-calling convention, in which functions that take a fixed number of arguments return pop their arguments on return with the `ret` instruction.

This calling convention is incompatible with the one normally used on Unix, so you cannot use it if you need to call libraries compiled with the Unix compiler.

Also, you must provide function prototypes for all functions that take variable numbers of arguments (including `printf`); otherwise incorrect code will be generated for calls to those functions.

In addition, seriously incorrect code will result if you call a function with too many arguments. (Normally, extra arguments are harmlessly ignored.)

This option takes its name from the 680x0 `rtd` instruction.

-mregparam

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, so you cannot use it if you need to call libraries compiled with the Unix compiler.

-mnoregparam

Do not pass any arguments in registers. This is the default for all targets.

-msb

It is OK to use the sb as an index register which is always loaded with zero. This is the default for the pc532-netbsd target.

-mnosb

The sb register is not available for use or has not been initialized to zero by the run time system. This is the default for all targets except the pc532-netbsd. It is also implied whenever **-mhimem** or **-fpic** is set.

-mhimem

Many ns32000 series addressing modes use displacements of up to 512MB. If an address is above 512MB then displacements from zero can not be used. This option causes code to be generated which can be loaded above 512MB. This may be useful for operating systems or ROM code.

-mnohimem

Assume code will be loaded in the first 512MB of virtual address space. This is the default for all platforms.

AVR Options

These options are defined for AVR implementations:

-mmcu=*mcu*

Specify ATMEL AVR instruction set or MCU type.

Instruction set avr1 is for the minimal AVR core, not supported by the C compiler, only for assembler programs (MCU types: at90s1200, attiny10, attiny11, attiny12, attiny15, attiny28).

Instruction set avr2 (default) is for the classic AVR core with up to 8K program memory space (MCU types: at90s2313, at90s2323, attiny22, at90s2333, at90s2343, at90s4414, at90s4433, at90s4434, at90s8515, at90c8534, at90s8535).

Instruction set avr3 is for the classic AVR core with up to 128K program memory space (MCU types: atmega103, atmega603).

Instruction set avr4 is for the enhanced AVR core with up to 8K program memory space (MCU types: atmega83, atmega85).

Instruction set avr5 is for the enhanced AVR core with up to 128K program memory space (MCU types: atmega161, atmega163, atmega32, at94k).

-msize

Output instruction sizes to the asm file.

-minit-stack=*N*

Specify the initial stack address, which may be a symbol or numeric value, `__stack` is the default.

-mno-interrupts

Generated code is not compatible with hardware interrupts. Code size will be smaller.

-mcall-prologues

Functions prologues/epilogues expanded as call to appropriate subroutines. Code size will be smaller.

-mno-tablejump

Do not generate tablejump insns which sometimes increase code size.

-mtiny-stack

Change only the low 8 bits of the stack pointer.

MCore Options

These are the **-m** options defined for the Motorola M*Core processors.

-mhardlit

-mhardlit

-mno-hardlit

Inline constants into the code stream if it can be done in two instructions or less.

-mdiv

-mdiv

-mno-div

Use the divide instruction. (Enabled by default).

-mrelax-immediate

-mrelax-immediate

-mno-relax-immediate

Allow arbitrary sized immediates in bit operations.

-mwide-bitfields

-mwide-bitfields

-mno-wide-bitfields

Always treat bitfields as int-sized.

-m4byte-functions

-m4byte-functions

-mno-4byte-functions

Force all functions to be aligned to a four byte boundary.

-mcallgraph-data

-mcallgraph-data

-mno-callgraph-data

Emit callgraph information.

-mslow-bytes

-mslow-bytes

-mno-slow-bytes

Prefer word access when reading byte quantities.

-mlittle-endian

-mlittle-endian

-mbig-endian

Generate code for a little endian target.

-m210

-m210

-m340

Generate code for the 210 processor.

IA-64 Options

These are the **-m** options defined for the Intel IA-64 architecture.

-mbig-endian

Generate code for a big endian target. This is the default for HPUX.

-mlittle-endian

Generate code for a little endian target. This is the default for AIX5 and Linux.

-mgnu-as

-mno-gnu-as

Generate (or don't) code for the GNU assembler. This is the default.

-mgnu-ld

-mno-gnu-ld

Generate (or don't) code for the GNU linker. This is the default.

-mno-pic

Generate code that does not use a global pointer register. The result is not position independent code, and violates the IA-64 ABI.

-mvolatile-asm-stop**-mno-volatile-asm-stop**

Generate (or don't) a stop bit immediately before and after volatile asm statements.

-mb-step

Generate code that works around Itanium B step errata.

-mregister-names**-mno-register-names**

Generate (or don't) **in**, **loc**, and **out** register names for the stacked registers. This may make assembler output more readable.

-mno-sdata**-msdata**

Disable (or enable) optimizations that use the small data section. This may be useful for working around optimizer bugs.

-mconstant-gp

Generate code that uses a single constant global pointer value. This is useful when compiling kernel code.

-mauto-pic

Generate code that is self-relocatable. This implies **-mconstant-gp**. This is useful when compiling firmware code.

-minline-divide-min-latency

Generate code for inline divides using the minimum latency algorithm.

-minline-divide-max-throughput

Generate code for inline divides using the maximum throughput algorithm.

-mno-dwarf2-asm**-mdwarf2-asm**

Don't (or do) generate assembler code for the DWARF2 line number debugging info. This may be useful when not using the GNU assembler.

-mfixed-range=*register range*

Generate code treating the given register range as fixed registers. A fixed register is one that the register allocator can not use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

D30V Options

These **-m** options are defined for D30V implementations:

-mextmem

Link the **.text**, **.data**, **.bss**, **.strings**, **.rodata**, **.rodata1**, **.data1** sections into external memory, which starts at location 0x80000000.

-mextmemory

Same as the **-mextmem** switch.

-monchip

Link the **.text** section into onchip text memory, which starts at location 0x0. Also link **.data**, **.bss**, **.strings**, **.rodata**, **.rodata1**, **.data1** sections into onchip data memory, which starts at location 0x20000000.

–mno-asm-optimize**–masm-optimize**

Disable (enable) passing **–O** to the assembler when optimizing. The assembler uses the **–O** option to automatically parallelize adjacent short instructions where possible.

–mbranch-cost=*n*

Increase the internal costs of branches to *n*. Higher costs means that the compiler will issue more instructions to avoid doing a branch. The default is 2.

–mcond-exec=*n*

Specify the maximum number of conditionally executed instructions that replace a branch. The default is 4.

Options for Code Generation Conventions

These machine-independent options control the interface conventions used in code generation.

Most of them have both positive and negative forms; the negative form of **–foo** would be **–fno-foo**. In the table below, only one of the forms is listed—the one which is not the default. You can figure out the other form by either removing **no-** or adding it.

–fexceptions

Enable exception handling. Generates extra code needed to propagate exceptions. For some targets, this implies GNU CC will generate frame unwind information for all functions, which can produce significant data size overhead, although it does not affect execution. If you do not specify this option, GNU CC will enable it by default for languages like C++ which normally require exception handling, and disable it for languages like C that do not normally require it. However, you may need to enable this option when compiling C code that needs to interoperate properly with exception handlers written in C++. You may also wish to disable this option if you are compiling older C++ programs that don't use exception handling.

–funwind-tables

Similar to **–fexceptions**, except that it will just generate any needed static data, but will not affect the generated code in any other way. You will normally not enable this option; instead, a language processor that needs this handling would enable it on your behalf.

–fpcc-struct-return

Return “short” struct and union values in memory like longer ones, rather than in registers. This convention is less efficient, but it has the advantage of allowing intercallability between GCC-compiled files and files compiled with other compilers.

The precise convention for returning structures in memory depends on the target configuration macros.

Short structures and unions are those whose size and alignment match that of some integer type.

–freg-struct-return

Use the convention that struct and union values are returned in registers when possible. This is more efficient for small structures than **–fpcc-struct-return**.

If you specify neither **–fpcc-struct-return** nor its contrary **–freg-struct-return**, GCC defaults to whichever convention is standard for the target. If there is no standard convention, GCC defaults to **–fpcc-struct-return**, except on targets where GCC is the principal compiler. In those cases, we can choose the standard, and we chose the more efficient register return alternative.

–fshort-enums

Allocate to an enum type only as many bytes as it needs for the declared range of possible values. Specifically, the enum type will be equivalent to the smallest integer type which has enough room.

–fshort-double

Use the same size for double as for float.

-fshared-data

Requests that the data and non-const variables of this compilation be shared data rather than private data. The distinction makes sense only on certain operating systems, where shared data is shared between processes running the same program, while private data exists in one copy per process.

-fno-common

Allocate even uninitialized global variables in the data section of the object file, rather than generating them as common blocks. This has the effect that if the same variable is declared (without `extern`) in two different compilations, you will get an error when you link them. The only reason this might be useful is if you wish to verify that the program will work on other systems which always work this way.

-fno-ident

Ignore the `#ident` directive.

-fno-gnu-linker

Do not output global initializations (such as C++ constructors and destructors) in the form used by the GNU linker (on systems where the GNU linker is the standard method of handling them). Use this option when you want to use a non-GNU linker, which also requires using the **collect2** program to make sure the system linker includes constructors and destructors. (**collect2** is included in the GCC distribution.) For systems which *must* use **collect2**, the compiler driver **gcc** is configured to do this automatically.

-finhibit-size-directive

Don't output a `.size` assembler directive, or anything else that would cause trouble if the function is split in the middle, and the two halves are placed at locations far apart in memory. This option is used when compiling *crtstuff.c*; you should not need to use it for anything else.

-fverbose-asm

Put extra commentary information in the generated assembly code to make it more readable. This option is generally only of use to those who actually need to read the generated assembly code (perhaps while debugging the compiler itself).

-fno-verbose-asm, the default, causes the extra information to be omitted and is useful when comparing two assembler files.

-fvolatile

Consider all memory references through pointers to be volatile.

-fvolatile-global

Consider all memory references to extern and global data items to be volatile. GCC does not consider static data items to be volatile because of this switch.

-fvolatile-static

Consider all memory references to static data to be volatile.

-fpic

Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of GCC; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-specific maximum size, you get an error message from the linker indicating that **-fpic** does not work; in that case, recompile with **-fPIC** instead. (These maximums are 16k on the m88k, 8k on the Sparc, and 32k on the m68k and RS/6000. The 386 has no such limit.)

Position-independent code requires special support, and therefore works only on certain machines. For the 386, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM RS/6000 is always position-independent.

-fpic

If supported for the target machine, emit position-independent code, suitable for dynamic linking and avoiding any limit on the size of the global offset table. This option makes a difference on the m68k, m88k, and the Sparc.

Position-independent code requires special support, and therefore works only on certain machines.

-ffixed-*reg*

Treat the register named *reg* as a fixed register; generated code should never refer to it (except perhaps as a stack pointer, frame pointer or in some other fixed role).

reg must be the name of a register. The register names accepted are machine-specific and are defined in the REGISTER_NAMES macro in the machine description macro file.

This flag does not have a negative form, because it specifies a three-way choice.

-fcall-used-*reg*

Treat the register named *reg* as an allocable register that is clobbered by function calls. It may be allocated for temporaries or variables that do not live across a call. Functions compiled this way will not save and restore the register *reg*.

It is an error to use this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model will produce disastrous results.

This flag does not have a negative form, because it specifies a three-way choice.

-fcall-saved-*reg*

Treat the register named *reg* as an allocable register saved by functions. It may be allocated even for temporaries or variables that live across a call. Functions compiled this way will save and restore the register *reg* if they use it.

It is an error to use this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model will produce disastrous results.

A different sort of disaster will result from the use of this flag for a register in which function values may be returned.

This flag does not have a negative form, because it specifies a three-way choice.

-fpack-struct

Pack all structure members together without holes. Usually you would not want to use this option, since it makes the code suboptimal, and the offsets of structure members won't agree with system libraries.

-fcheck-memory-usage

Generate extra code to check each memory access. GCC will generate code that is suitable for a detector of bad memory accesses such as *Checker*.

Normally, you should compile all, or none, of your code with this option.

If you do mix code compiled with and without this option, you must ensure that all code that has side effects and that is called by code compiled with this option is, itself, compiled with this option. If you do not, you might get erroneous messages from the detector.

If you use functions from a library that have side-effects (such as `read`), you might not be able to recompile the library and specify this option. In that case, you can enable the **-fprefix-function-name** option, which requests GCC to encapsulate your code and make other functions look as if they were compiled with **-fcheck-memory-usage**. This is done by calling "stubs", which are provided by the detector. If you cannot find or build stubs for every function you call, you might have to specify **-fcheck-memory-usage** without **-fprefix-function-name**.

If you specify this option, you can not use the `asm` or `__asm__` keywords in functions with memory checking enabled. GNU CC cannot understand what the `asm` statement may do, and therefore cannot

generate the appropriate code, so it will reject it. However, if you specify the function attribute `no_check_memory_usage`, GNU CC will disable memory checking within a function; you may use `asm` statements inside such functions. You may have an inline expansion of a non-checked function within a checked function; in that case GNU CC will not generate checks for the inlined function's memory accesses.

If you move your `asm` statements to non-checked inline functions and they do access memory, you can add calls to the support code in your inline function, to indicate any reads, writes, or copies being done. These calls would be similar to those done in the stubs described above.

-fprefix-function-name

Request GCC to add a prefix to the symbols generated for function names. GCC adds a prefix to the names of functions defined as well as functions called. Code compiled with this option and code compiled without the option can't be linked together, unless stubs are used.

If you compile the following code with **-fprefix-function-name**

```
extern void bar (int);
void
foo (int a)
{
    return bar (a + 5);
}
```

GCC will compile the code as if it was written:

```
extern void prefix_bar (int);
void
prefix_foo (int a)
{
    return prefix_bar (a + 5);
}
```

This option is designed to be used with **-fcheck-memory-usage**.

-finstrument-functions

Generate instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions will be called with the address of the current function and its call site. (On some platforms, `__builtin_return_address` does not work beyond the current function, so the call site information may not be available to the profiling functions otherwise.)

```
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 may be looked up exactly in the symbol table.

This instrumentation is also done for functions expanded inline in other functions. The profiling calls will indicate where, conceptually, the inline function is entered and exited. This means that addressable versions of such functions must be available. If all your uses of a function are expanded inline, this may mean an additional expansion of code size. If you use **extern inline** in your C code, an addressable version of such functions must be provided. (This is normally the case anyways, but if you get lucky and the optimizer always expands the functions inline, you might have gotten away without providing static copies.)

A function may be given the attribute `no_instrument_function`, in which case this instrumentation will not be done. This can be used, for example, for the profiling functions listed above, high-priority interrupt routines, and any functions from which the profiling functions cannot safely be called (perhaps signal handlers, if the profiling routines generate output or allocate memory).

-fstack-check

Generate code to verify that you do not go beyond the boundary of the stack. You should specify this flag if you are running in an environment with multiple threads, but only rarely need to specify it in a single-threaded environment since stack overflow is automatically detected on nearly all systems if there is only one stack.

Note that this switch does not actually cause checking to be done; the operating system must do that. The switch causes generation of code to ensure that the operating system sees the stack being extended.

-fstack-limit-register=reg**-fstack-limit-symbol=sym****-fno-stack-limit**

Generate code to ensure that the stack does not grow beyond a certain value, either the value of a register or the address of a symbol. If the stack would grow beyond the value, a signal is raised. For most targets, the signal is raised before the stack overruns the boundary, so it is possible to catch the signal without taking special precautions.

For instance, if the stack starts at address **0x80000000** and grows downwards you can use the flags **-fstack-limit-symbol=__stack_limit -Wl,--defsym,__stack_limit=0x7ffe0000** which will enforce a stack limit of 128K.

-fargument-alias**-fargument-noalias****-fargument-noalias-global**

Specify the possible relationships among parameters and between parameters and global data.

-fargument-alias specifies that arguments (parameters) may alias each other and may alias global storage. **-fargument-noalias** specifies that arguments do not alias each other, but may alias global storage. **-fargument-noalias-global** specifies that arguments do not alias each other and do not alias global storage.

Each language will automatically use whatever option is required by the language standard. You should not need to use these options yourself.

-fleading-underscore

This option and its counterpart, **-fno-leading-underscore**, forcibly change the way C symbols are represented in the object file. One use is to help link with legacy assembly code.

Be warned that you should know what you are doing when invoking this option, and that not all targets provide complete support for it.

ENVIRONMENT

This section describes several environment variables that affect how GCC operates. Some of them work by specifying directories or prefixes to use when searching for various kinds of files. Some are used to specify other aspects of the compilation environment.

Note that you can also specify places to search using options such as **-B**, **-I** and **-L**. These take precedence over places specified using environment variables, which in turn take precedence over those specified by the configuration of GCC.

LANG**LC_CTYPE****LC_MESSAGES****LC_ALL**

These environment variables control the way that GCC uses localization information that allow GCC to work with different national conventions. GCC inspects the locale categories **LC_CTYPE** and **LC_MESSAGES** if it has been configured to do so. These locale categories can be set to any value supported by your installation. A typical value is **en_UK** for English in the United Kingdom.

The **LC_CTYPE** environment variable specifies character classification. GCC uses it to determine the

character boundaries in a string; this is needed for some multibyte encodings that contain quote and escape characters that would otherwise be interpreted as a string end or escape.

The **LC_MESSAGES** environment variable specifies the language to use in diagnostic messages.

If the **LC_ALL** environment variable is set, it overrides the value of **LC_CTYPE** and **LC_MESSAGES**; otherwise, **LC_CTYPE** and **LC_MESSAGES** default to the value of the **LANG** environment variable. If none of these variables are set, GCC defaults to traditional C English behavior.

TMPDIR

If **TMPDIR** is set, it specifies the directory to use for temporary files. GCC uses temporary files to hold the output of one stage of compilation which is to be used as input to the next stage: for example, the output of the preprocessor, which is the input to the compiler proper.

GCC_EXEC_PREFIX

If **GCC_EXEC_PREFIX** is set, it specifies a prefix to use in the names of the subprograms executed by the compiler. No slash is added when this prefix is combined with the name of a subprogram, but you can specify a prefix that ends with a slash if you wish.

If **GCC_EXEC_PREFIX** is not set, GNU CC will attempt to figure out an appropriate prefix to use based on the pathname it was invoked with.

If GCC cannot find the subprogram using the specified prefix, it tries looking in the usual places for the subprogram.

The default value of **GCC_EXEC_PREFIX** is *prefix/lib/gcc-lib/* where *prefix* is the value of *prefix* when you ran the *configure* script.

Other prefixes specified with **-B** take precedence over this prefix.

This prefix is also used for finding files such as *crt0.o* that are used for linking.

In addition, the prefix is used in an unusual way in finding the directories to search for header files. For each of the standard directories whose name normally begins with **/usr/local/lib/gcc-lib** (more precisely, with the value of **GCC_INCLUDE_DIR**), GCC tries replacing that beginning with the specified prefix to produce an alternate directory name. Thus, with **-Bfoo/**, GCC will search *foo/bar* where it would normally search */usr/local/lib/bar*. These alternate directories are searched first; the standard directories come next.

COMPILER_PATH

The value of **COMPILER_PATH** is a colon-separated list of directories, much like **PATH**. GCC tries the directories thus specified when searching for subprograms, if it can't find the subprograms using **GCC_EXEC_PREFIX**.

LIBRARY_PATH

The value of **LIBRARY_PATH** is a colon-separated list of directories, much like **PATH**. When configured as a native compiler, GCC tries the directories thus specified when searching for special linker files, if it can't find them using **GCC_EXEC_PREFIX**. Linking using GCC also uses these directories when searching for ordinary libraries for the **-l** option (but directories specified with **-L** come first).

C_INCLUDE_PATH

CPLUS_INCLUDE_PATH

OBJC_INCLUDE_PATH

These environment variables pertain to particular languages. Each variable's value is a colon-separated list of directories, much like **PATH**. When GCC searches for header files, it tries the directories listed in the variable for the language you are using, after the directories specified with **-I** but before the standard header file directories.

DEPENDENCIES_OUTPUT

If this variable is set, its value specifies how to output dependencies for Make based on the header files processed by the compiler. This output looks much like the output from the **-M** option, but it goes to a separate file, and is in addition to the usual results of compilation.

The value of **DEPENDENCIES_OUTPUT** can be just a file name, in which case the Make rules are written to that file, guessing the target name from the source file name. Or the value can have the form *file target*, in which case the rules are written to file *file* using *target* as the target name.

LANG

This variable is used to pass locale information to the compiler. One way in which this information is used is to determine the character set to be used when character literals, string literals and comments are parsed in C and C++. When the compiler is configured to allow multibyte characters, the following values for **LANG** are recognized:

C-JIS

Recognize JIS characters.

C-SJIS

Recognize SJIS characters.

C-EUCJP

Recognize EUCJP characters.

If **LANG** is not defined, or if it has some other value, then the compiler will use `mblen` and `mbtowc` as defined by the default locale to recognize and translate multibyte characters.

BUGS

For instructions on reporting bugs, see <<http://gcc.gnu.org/bugs.html>>. Use of the **gccbug** script to report bugs is recommended.

FOOTNOTES

1. On some systems, **gcc -shared** needs to build supplementary stub code for constructors to work. On multi-libbed systems, **gcc -shared** must select the correct support libraries to link against. Failing to supply the correct flags may lead to subtle defects. Supplying them in cases where they are not necessary is innocuous.

SEE ALSO

cpp(1), *gcov*(1), *g77*(1), *as*(1), *ld*(1), *gdb*(1), *adb*(1), *dbx*(1), *sdb*(1) and the Info entries for *gcc*, *cpp*, *g77*, *as*, *ld*, *binutils* and *gdb*.

AUTHOR

See the Info entry for *gcc*, or <<http://gcc.gnu.org/thanks.html>>, for contributors to GCC.

COPYRIGHT

Copyright (c) 1988, 1989, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.

NAME

ld – the GNU linker

SYNOPSIS

```
ld      [ -o output ] objfile...
        [ -Aarchitecture ] [ -b input-format ] [ -Bstatic ] [ -Bdynamic ] [ -Bsymbolic ] [ -c commandfile ]
        [ --cref ] [ -d|-dc|-dp ]
        [ -defsym symbol=expression ] [ --demangle ] [ --no-demangle ] [ -e entry ]
        [ -embedded-relocs ] [ -E ] [ -export-dynamic ] [ -f name ] [ --auxiliary name ] [ -F name ]
        [ --filter name ] [ -format input-format ] [ -g ] [ -G size ] [ -h name ] [ -soname name ]
        [ --help ] [ -i ] [ -lar ] [ -Lsearchdir ] [ -M ] [ -Map mapfile ] [ -m emulation ] [ -n|-N ]
        [ -noinhibit-exec ] [ -no-keep-memory ] [ -no-warn-mismatch ] [ -Olevel ]
        [ -oformat output-format ] [ -R filename ] [ -relax ] [ -r|-Ur ] [ -rpath directory ]
        [ -rpath-link directory ] [ -S ] [ -s ] [ -shared ] [ -sort-common ] [ -split-by-reloc count ]
        [ -split-by-file ] [ -T commandfile ] [ --section-start sectionname=sectionorg ]
        [ -Ttext textorg ] [ -Tdata dataorg ] [ -Tbss bssorg ] [ -t ] [ -u sym ] [ -V ] [ -v ] [ --verbose ]
        [ --version ] [ -warn-common ] [ -warn-constructors ] [ -warn-multiple-gp ]
        [ -warn-once ] [ -warn-section-align ] [ --whole-archive ] [ --no-whole-archive ]
        [ --wrap symbol ] [ -X ] [ -x ]
```

DESCRIPTION

ld combines a number of object and archive files, relocates their data and ties up symbol references. Often the last step in building a new compiled program to run is a call to **ld**.

ld accepts Linker Command Language files to provide explicit and total control over the linking process. This man page does not describe the command language; see the ‘**ld**’ entry in ‘**info**’, or the manual *ld: the GNU linker*, for full details on the command language and on other aspects of the GNU linker.

This version of **ld** uses the general purpose BFD libraries to operate on object files. This allows **ld** to read, combine, and write object files in many different formats—for example, COFF or **a.out**. Different formats may be linked together to produce any available kind of object file. You can use ‘**objdump -i**’ to get a list of formats supported on various architectures; see **objdump(1)**.

Aside from its flexibility, the GNU linker is more helpful than other linkers in providing diagnostic information. Many linkers abandon execution immediately upon encountering an error; whenever possible, **ld** continues executing, allowing you to identify other errors (or, in some cases, to get an output file in spite of the error).

The GNU linker **ld** is meant to cover a broad range of situations, and to be as compatible as possible with other linkers. As a result, you have many choices to control its behavior through the command line, and through environment variables.

OPTIONS

The plethora of command-line options may seem intimidating, but in actual practice few of them are used in any particular context. For instance, a frequent use of **ld** is to link standard Unix object files on a standard, supported Unix system. On such a system, to link a file **hello.o**:

```
$ ld -o output /lib/crt0.o hello.o -lc
```

This tells **ld** to produce a file called **output** as the result of linking the file **/lib/crt0.o** with **hello.o** and the library **libc.a** which will come from the standard search directories.

The command-line options to **ld** may be specified in any order, and may be repeated at will. For the most part, repeating an option with a different argument will either have no further effect, or override prior occur-

rences (those further to the left on the command line) of an option.

The exceptions—which may meaningfully be used more than once—are **-A**, **-b** (or its synonym **-format**), **-defsym**, **--section-start**, **-L**, **-l**, **-R**, and **-u**.

The list of object files to be linked together, shown as *objfile*, may follow, precede, or be mixed in with command-line options; save that an *objfile* argument may not be placed between an option flag and its argument.

Usually the linker is invoked with at least one object file, but other forms of binary input files can also be specified with **-l**, **-R**, and the script command language. If *no* binary input files at all are specified, the linker does not produce any output, and issues the message ‘**No input files**’.

Option arguments must either follow the option letter without intervening whitespace, or be given as separate arguments immediately following the option that requires them.

-A*architecture*

In the current release of **ld**, this option is useful only for the Intel 960 family of architectures. In that **ld** configuration, the *architecture* argument is one of the two-letter names identifying members of the 960 family; the option specifies the desired output target, and warns of any incompatible instructions in the input files. It also modifies the linker’s search strategy for archive libraries, to support the use of libraries specific to each particular architecture, by including in the search loop names suffixed with the string identifying the architecture.

For example, if your **ld** command line included ‘**-ACA**’ as well as ‘**-ltry**’, the linker would look (in its built-in search paths, and in any paths you specify with **-L**) for a library with the names

```
try
libtry.a
tryca
libtryca.a
```

The first two possibilities would be considered in any event; the last two are due to the use of ‘**-ACA**’.

Future releases of **ld** may support similar functionality for other architecture families.

You can meaningfully use **-A** more than once on a command line, if an architecture family allows combination of target architectures; each use will add another pair of name variants to search for when **-l** specifies a library.

-b*input-format*

Specify the binary format for input object files that follow this option on the command line. You don’t usually need to specify this, as **ld** is configured to expect as a default input format the most usual format on each machine. *input-format* is a text string, the name of a particular format supported by the BFD libraries. **-format** *input-format* has the same effect, as does the script command **TARGET**.

You may want to use this option if you are linking files with an unusual binary format. You can also use **-b** to switch formats explicitly (when linking object files of different formats), by including **-b** *input-format* before each group of object files in a particular format.

The default format is taken from the environment variable **GNUTARGET**. You can also define

the input format from a script, using the command **TARGET**.

-Bstatic

Do not link against shared libraries. This is only meaningful on platforms for which shared libraries are supported.

-Bdynamic

Link against dynamic libraries. This is only meaningful on platforms for which shared libraries are supported. This option is normally the default on such platforms.

-Bsymbolic

When creating a shared library, bind references to global symbols to the definition within the shared library, if any. Normally, it is possible for a program linked against a shared library to override the definition within the shared library. This option is only meaningful on ELF platforms which support shared libraries.

-c *commandfile*

Directs **ld** to read link commands from the file *commandfile*. These commands will completely override **ld**'s default link format (rather than adding to it); *commandfile* must specify everything necessary to describe the target format.

You may also include a script of link commands directly in the command line by bracketing it between '{' and '}' characters.

--cref Output a cross reference table. If a linker map file is being generated, the cross reference table is printed to the map file. Otherwise, it is printed on the standard output.

-d

-dc

-dp These three options are equivalent; multiple forms are supported for compatibility with other linkers. Use any of them to make **ld** assign space to common symbols even if a relocatable output file is specified (**-r**). The script command **FORCE_COMMON_ALLOCATION** has the same effect.

-defsym *symbol=expression*

Create a global symbol in the output file, containing the absolute address given by *expression*. You may use this option as many times as necessary to define multiple symbols in the command line. A limited form of arithmetic is supported for the *expression* in this context: you may give a hexadecimal constant or the name of an existing symbol, or use + and - to add or subtract hexadecimal constants or symbols. If you need more elaborate expressions, consider using the linker command language from a script.

--demangle

--no-demangle

These options control whether to demangle symbol names in error messages and other output. When the linker is told to demangle, it tries to present symbol names in a readable fashion: it strips leading underscores if they are used by the object file format, and converts C++ mangled symbol names into user readable names. The linker will demangle by default unless the environment variable **COLLECT_NO_DEMANGLE** is set. These options may be used to override the default.

-e *entry*

Use *entry* as the explicit symbol for beginning execution of your program, rather than the default entry point. See the '**ld**' entry in '**info**' for a discussion of defaults and other ways of specifying the entry point.

--embedded-relocs

This option is only meaningful when linking MIPS embedded PIC code, generated by the **--membedded-pic** option to the GNU compiler and assembler. It causes the linker to create a table which may be used at runtime to relocate any data which was statically initialized to pointer values. See the code in `testsuite/ld-empic` for details.

-E**--export-dynamic**

When creating an ELF file, add all symbols to the dynamic symbol table. Normally, the dynamic symbol table contains only symbols which are used by a dynamic object. This option is needed for some uses of *dlopen*.

-f *name***--auxiliary** *name*

When creating an ELF shared object, set the internal DT_AUXILIARY field to the specified name. This tells the dynamic linker that the symbol table of the shared object should be used as an auxiliary filter on the symbol table of the shared object *name*.

-F *name***--filter** *name*

When creating an ELF shared object, set the internal DT_FILTER field to the specified name. This tells the dynamic linker that the symbol table of the shared object should be used as a filter on the symbol table of the shared object *name*.

--format *input-format*

Synonym for **-b** *input-format*.

-g Accepted, but ignored; provided for compatibility with other tools.

-G *size*

Set the maximum size of objects to be optimized using the GP register to *size* under MIPS ECOFF. Ignored for other object file formats.

-h *name***-soname** *name*

When creating an ELF shared object, set the internal DT_SONAME field to the specified name. When an executable is linked with a shared object which has a DT_SONAME field, then when the executable is run the dynamic linker will attempt to load the shared object specified by the DT_SONAME field rather than the using the file name given to the linker.

--help Print a summary of the command-line options on the standard output and exit. This option and **--version** begin with two dashes instead of one for compatibility with other GNU programs. The other options start with only one dash for compatibility with other linkers.

- i** Perform an incremental link (same as option **-r**).
- lar** Add an archive file *ar* to the list of files to link. This option may be used any number of times. **ld** will search its path-list for occurrences of **libar.a** for every *ar* specified.
- Lsearchdir**
This command adds path *searchdir* to the list of paths that **ld** will search for archive libraries. You may use this option any number of times.
- The default set of paths searched (without being specified with **-L**) depends on what emulation mode **ld** is using, and in some cases also on how it was configured. The paths can also be specified in a link script with the **SEARCH_DIR** command.
- M** Print (to the standard output file) a link map—diagnostic information about where symbols are mapped by **ld**, and information on global common storage allocation.
- Map mapfile**
Print to the file *mapfile* a link map—diagnostic information about where symbols are mapped by **ld**, and information on global common storage allocation.
- m emulation**
Emulate the *emulation* linker. You can list the available emulations with the **--verbose** or **-V** options. This option overrides the compiled-in default, which is the system for which you configured **ld**.
- N** specifies readable and writable **text** and **data** sections. If the output format supports Unix style magic numbers, the output is marked as **OMAGIC**.
- When you use the ‘**-N**’ option, the linker does not page-align the data segment.
- n** sets the text segment to be read only, and **NMAGIC** is written if possible.
- noinhibit-exec**
Normally, the linker will not produce an output file if it encounters errors during the link process. With this flag, you can specify that you wish the output file retained even after non-fatal errors.
- no-keep-memory**
The linker normally optimizes for speed over memory usage by caching the symbol tables of input files in memory. This option tells the linker to instead optimize for memory usage, by rereading the symbol tables as necessary. This may be required if the linker runs out of memory space while linking a large executable.
- no-warn-mismatch**
Normally the linker will give an error if you try to link together input files that are mismatched for some reason, perhaps because they have been compiled for different processors or for different endiannesses. This option tells the linker that it should silently permit such possible errors. This option should only be used with care, in cases when you have taken some special action that ensures that the linker errors are inappropriate.

-o *output*

output is a name for the program produced by **ld**; if this option is not specified, the name ‘**a.out**’ is used by default. The script command **OUTPUT** can also specify the output file name.

-O*level*

Generate optimized output files. This might use significantly more time and therefore probably should be enabled only for generating the final binary. *level* is supposed to be a numeric value. Any value greater than zero enables the optimizations.

--oformat *output-format*

Specify the binary format for the output object file. You don’t usually need to specify this, as **ld** is configured to produce as a default output format the most usual format on each machine. *output-format* is a text string, the name of a particular format supported by the BFD libraries. The script command **OUTPUT_FORMAT** can also specify the output format, but this option overrides it.

-R *filename*

Read symbol names and their addresses from *filename*, but do not relocate it or include it in the output. This allows your output file to refer symbolically to absolute locations of memory defined in other programs.

-relax An option with machine dependent effects. Currently this option is only supported on the H8/300.

On some platforms, use this option to perform global optimizations that become possible when the linker resolves addressing in your program, such as relaxing address modes and synthesizing new instructions in the output object file.

On platforms where this is not supported, ‘**-relax**’ is accepted, but has no effect.

-r Generates relocatable output—i.e., generate an output file that can in turn serve as input to **ld**. This is often called *partial* linking. As a side effect, in environments that support standard Unix magic numbers, this option also sets the output file’s magic number to **OMAGIC**. If this option is not specified, an absolute file is produced. When linking C++ programs, this option *will not* resolve references to constructors; **-Ur** is an alternative.

This option does the same as **-i**.

-rpath *directory*

Add a directory to the runtime library search path. This is used when linking an ELF executable with shared objects. All **-rpath** arguments are concatenated and passed to the runtime linker, which uses them to locate shared objects at runtime. The **-rpath** option is also used when locating shared objects which are needed by shared objects explicitly included in the link; see the description of the **-rpath-link** option. If **-rpath** is not used when linking an ELF executable, the contents of the environment variable **LD_RUN_PATH** will be used if it is defined.

The **-rpath** option may also be used on SunOS. By default, on SunOS, the linker will form a runtime search path out of all the **-L** options it is given. If a **-rpath** option is used, the runtime search path will be formed exclusively using the **-rpath** options, ignoring the **-L** options. This can be useful when using gcc, which adds many **-L** options which may be on NFS mounted filesystems.

-rpath-link *directory*

When using ELF or SunOS, one shared library may require another. This happens when an **ld -shared** link includes a shared library as one of the input files.

When the linker encounters such a dependency when doing a non-shared, non-relocateable link, it will automatically try to locate the required shared library and include it in the link, if it is not included explicitly. In such a case, the **-rpath-link** option specifies the first set of directories to search. The **-rpath-link** option may specify a sequence of directory names either by specifying a list of names separated by colons, or by appearing multiple times.

If the required shared library is not found, the linker will issue a warning and continue with the link.

-S Omits debugger symbol information (but not all symbols) from the output file.

-s Omits all symbol information from the output file.

-shared

Create a shared library. This is currently only supported on ELF and SunOS platforms (on SunOS it is not required, as the linker will automatically create a shared library when there are undefined symbols and the **-e** option is not used).

-sort-common

Normally, when **ld** places the global common symbols in the appropriate output sections, it sorts them by size. First come all the one byte symbols, then all the two bytes, then all the four bytes, and then everything else. This is to prevent gaps between symbols due to alignment constraints. This option disables that sorting.

-split-by-reloc *count*

Trys to creates extra sections in the output file so that no single output section in the file contains more than *count* relocations. This is useful when generating huge relocatable for downloading into certain real time kernels with the COFF object file format; since COFF cannot represent more than 65535 relocations in a single section. Note that this will fail to work with object file formats which do not support arbitrary sections. The linker will not split up individual input sections for redistribution, so if a single input section contains more than *count* relocations one output section will contain that many relocations.

-split-by-file

Similar to **-split-by-reloc** but creates a new output section for each input file.

--section-start *sectionname=org*

Locate a section in the output file at the absolute address given by *org*. *org* must be a hexadecimal integer. You may use this option as many times as necessary to locate multiple sections in the command line. If you need more elaborate expressions, consider using the linker command language from a script.

-Tbss *org*

-Tdata *org*

- Ttext** *org*
Use *org* as the starting address for—respectively—the **bss**, **data**, or the **text** segment of the output file. *org* must be a hexadecimal integer.
- T** *commandfile*
Equivalent to **-c** *commandfile*; supported for compatibility with other tools.
- t** Prints names of input files as **ld** processes them.
- u** *sym* Forces *sym* to be entered in the output file as an undefined symbol. This may, for example, trigger linking of additional modules from standard libraries. **-u** may be repeated with different option arguments to enter additional undefined symbols.
- Ur** For anything other than C++ programs, this option is equivalent to **-r**: it generates relocatable output—i.e., an output file that can in turn serve as input to **ld**. When linking C++ programs, **-Ur** will resolve references to constructors, unlike **-r**.
- verbose**
Display the version number for **ld** and list the supported emulations. Display which input files can and can not be opened.
- v, -V** Display the version number for **ld**. The **-V** option also lists the supported emulations.
- version**
Display the version number for **ld** and exit.
- warn-common**
Warn when a common symbol is combined with another common symbol or with a symbol definition. Unix linkers allow this somewhat sloppy practice, but linkers on some other operating systems do not. This option allows you to find potential problems from combining global symbols.
- warn-constructors**
Warn if any global constructors are used. This is only useful for a few object file formats. For formats like COFF or ELF, the linker can not detect the use of global constructors.
- warn-multiple-gp**
Warn if the output file requires multiple global-pointer values. This option is only meaningful for certain processors, such as the Alpha.
- warn-once**
Only warn once for each undefined symbol, rather than once per module which refers to it.
- warn-section-align**
Warn if the address of an output section is changed because of alignment. Typically, the alignment will be set by an input section. The address will only be changed if it not explicitly specified; that is, if the **SECTIONS** command does not specify a start address for the section.

--whole-archive

For each archive mentioned on the command line after the **--whole-archive** option, include every object file in the archive in the link, rather than searching the archive for the required object files. This is normally used to turn an archive file into a shared library, forcing every object to be included in the resulting shared library.

--no-whole-archive

Turn off the effect of the **--whole-archive** option for archives which appear later on the command line.

--wrap *symbol*

Use a wrapper function for *symbol*. Any undefined reference to *symbol* will be resolved to **__wrap_***symbol*. Any undefined reference to **__real_***symbol* will be resolved to *symbol*.

-X Delete all temporary local symbols. For most targets, this is all local symbols whose names begin with 'L'.

-x Delete all local symbols.

ENVIRONMENT

You can change the behavior of **ld** with the environment variable **GNUTARGET**.

GNUTARGET determines the input-file object format if you don't use **-b** (or its synonym **-format**). Its value should be one of the BFD names for an input format. If there is no **GNUTARGET** in the environment, **ld** uses the natural format of the host. If **GNUTARGET** is set to **default** then BFD attempts to discover the input format by examining binary input files; this method often succeeds, but there are potential ambiguities, since there is no method of ensuring that the magic number used to flag object-file formats is unique. However, the configuration procedure for BFD on each system places the conventional format for that system first in the search-list, so ambiguities are resolved in favor of convention.

SEE ALSO

objdump(1)

'**ld**' and '**binutils**' entries in **info**

ld: the GNU linker, Steve Chamberlain and Roland Pesch; *The GNU Binary Utilities*, Roland H. Pesch.

COPYING

Copyright (c) 1991, 92, 93, 94, 95, 96, 97, 1998, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

nlmconv – converts object code into an NLM

SYNOPSIS**nlmconv**

```
[ -I bfdname | --input-target=bfdname ] [ -O bfdname | --output-target=bfdname ]
[ -T headerfile | --header-file=headerfile ] [ -V | --version ] [ --help ] infile outfile
```

DESCRIPTION

nlmconv converts the relocatable object file **infile** into the NetWare Loadable Module **outfile**, optionally reading *headerfile* for NLM header information. For instructions on writing the NLM command file language used in header files, see *The NetWare Tool Maker Specification Manual*, available from Novell, Inc. **nlmconv** currently works with i386 object files in **COFF**, **ELF**, or **a.out** format, and with SPARC object files in **ELF** or **a.out** format.

nlmconv uses the GNU Binary File Descriptor library to read *infile*.

OPTIONS

-I *bfdname*, **--input-target=*bfdname***

Consider the source file's object format to be *bfdname*, rather than attempting to deduce it.

-O *bfdname*, **--output-target=*bfdname***

Write the output file using the object format *bfdname*. **nlmconv** infers the output format based on the input format, e.g. for an i386 input file the output format is *nlm32-i386*.

-T *headerfile*, **--header-file=*headerfile***

Reads *headerfile* for NLM header information. For instructions on writing the NLM command file language used in header files, see *The NetWare Tool Maker Specification Manual*, available from Novell, Inc.

-V, **--version**

Show the version number of **nlmconv** and exit.

-h, **--help**

Show a summary of the options to **nlmconv** and exit.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (June 1993).

COPYING

Copyright (c) 1993, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License. That license is described in the GNU Free Documentation License section.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

nm – list symbols from object files.

SYNOPSIS

```
nm    [ -a|--debug-syms ] [ -g|--extern-only ] [ -B ] [ -C|--demangle ] [ -D|--dynamic ]
      [ -s|--print-armap ] [ -o|--print-file-name ] [ -n|--numeric-sort ] [ -p|--no-sort ]
      [ -r|--reverse-sort ] [ --size-sort ] [ -u|--undefined-only ] [ -l|--line-numbers ] [ --help ]
      [ --version ] [ -t radix | --radix=radix ] [ -P|--portability ] [ -f format | --format=format ]
      [ --target=bfdname ] [ objfile... ]
```

DESCRIPTION

GNU **nm** lists the symbols from object files *objfile*. If no object files are given as arguments, **nm** assumes ‘**a.out**’.

OPTIONS

The long and short forms of options, shown here as alternatives, are equivalent.

-A

-o

--print-file-name

Precede each symbol by the name of the input file where it was found, rather than identifying the input file once only before all of its symbols.

-a

--debug-syms

Display debugger-only symbols; normally these are not listed.

-B The same as **--format=bsd** (for compatibility with the MIPS **nm**).

-C

--demangle

Decode (*demangle*) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable.

-D

--dynamic

Display the dynamic symbols rather than the normal symbols. This is only meaningful for dynamic objects, such as certain types of shared libraries.

-f *format*

Use the output format *format*, which can be “bsd”, “sysv”, or “posix”. The default is “bsd”. Only the first character of *format* is significant; it can be either upper or lower case.

-g

--extern-only

Display only external symbols.

-n

-v

--numeric-sort

Sort symbols numerically by their addresses, not alphabetically by their names.

-p

--no-sort

Don't bother to sort the symbols in any order; just print them in the order encountered.

-P

--portability

Use the POSIX.2 standard output format instead of the default format. Equivalent to “-f posix”.

-s

--print-armap

When listing symbols from archive members, include the index: a mapping (stored in the archive by **ar** or **ranlib**) of what modules contain definitions for what names.

-r

--reverse-sort

Reverse the sense of the sort (whether numeric or alphabetic); let the last come first.

--size-sort

Sort symbols by size. The size is computed as the difference between the value of the symbol and the value of the symbol with the next higher value. The size of the symbol is printed, rather than the value.

-t radix

--radix=radix

Use *radix* as the radix for printing the symbol values. It must be “d” for decimal, “o” for octal, or “x” for hexadecimal.

--target=bfdname

Specify an object code format other than your system's default format. See **objdump(1)**, for information on listing available formats.

-u

--undefined-only

Display only undefined symbols (those external to each object file).

-l

--line-numbers

For each symbol, use debugging information to try to find a filename and line number. For a defined symbol, look for the line number of the address of the symbol. For an undefined symbol, look for the line number of a relocation entry which refers to the symbol. If line number information can be found, print it after the other symbol information.

-V

--version

Show the version number of **nm** and exit.

--help Show a summary of the options to **nm** and exit.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991); **ar(1)**, **objdump(1)**, **ranlib(1)**.

COPYING

Copyright (c) 1991, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

objcopy – copy and translate object files

SYNOPSIS**objcopy**

```
[ -F bfdname | --target=bfdname ] [ -I bfdname | --input-target=bfdname ]
[ -O bfdname | --output-target=bfdname ] [ -j sectionname | --only-section=sectionname ]
[ -R sectionname | --remove-section=sectionname ] [ -S | --strip-all ] [ -g | --strip-debug ]
[ --strip-unnneeded ] [ -K symbolname | --keep-symbol=symbolname ]
[ -N symbolname | --strip-symbol=symbolname ]
[ -L symbolname | --localize-symbol=symbolname ]
[ -W symbolname | --weaken-symbol=symbolname ] [ -x | --discard-all ]
[ -X | --discard-locals ] [ -b byte | --byte=byte ] [ -i interleave | --interleave=interleave ]
[ -p | --preserve-dates ] [ --debugging ] [ --gap-fill=val ] [ --pad-to=address ]
[ --set-start=val ] [ --change-start=incr ] [ --change-addresses=incr ]
[ --change-section-address section{=,+,-}val ] [ --change-section-lma section{=,+,-}val ]
[ --change-section-vma section{=,+,-}val ] [ --change-warnings ]
[ --no-change-warnings ] [ --set-section-flags section=flags ]
[ --add-section sectionname=filename ] [ --change-leading-char ]
[ --remove-leading-char ] [ --srec-len=val ] [ --srec-forceS3 ] [ --redefine-sym old=new ]
[ --weaken ] [ -v | --verbose ] [ -V | --version ] [ --help ] infile [ outfile ]
```

DESCRIPTION

The GNU **objcopy** utility copies the contents of an object file to another. **objcopy** uses the GNU BFD Library to read and write the object files. It can write the destination object file in a format different from that of the source object file. The exact behavior of **objcopy** is controlled by command-line options.

objcopy creates temporary files to do its translations and deletes them afterward. **objcopy** uses BFD to do all its translation work; it knows about all the formats BFD knows about, and thus is able to recognize most formats without being told explicitly.

objcopy can be used to generate S-records by using an output target of **srec** (e.g., use **-O srec**).

objcopy can be used to generate a raw binary file by using an output target of **binary** (e.g., use **-O binary**). When **objcopy** generates a raw binary file, it will essentially produce a memory dump of the contents of the input object file. All symbols and relocation information will be discarded. The memory dump will start at the virtual address of the lowest section copied into the output file.

When generating an S-record or a raw binary file, it may be helpful to use **-S** to remove sections containing debugging information. In some cases **-R** will be useful to remove sections which contain information which is not needed by the binary file.

infile and *outfile* are the source and output files respectively. If you do not specify *outfile*, **objcopy** creates a temporary file and destructively renames the result with the name of the input file.

OPTIONS

-I *bfdname*, --input-target=*bfdname*

Consider the source file's object format to be *bfdname*, rather than attempting to deduce it.

-O *bfdname*, --output-target=*bfdname*

Write the output file using the object format *bfdname*.

-F *bfdname*, --target=*bfdname*

Use *bfdname* as the object format for both the input and the output file; i.e. simply transfer data from source to destination with no translation.

-j *sectionname*, --only-section=*sectionname*

Copy only the named section from the input file to the output file, discarding all other sections.

This option may be given more than once. Note that using this option inappropriately may make

the output file unusable.

-R *sectionname*, --remove-section=*sectionname*

Remove the named section from the file. This option may be given more than once. Note that using this option inappropriately may make the output file unusable.

-S, --strip-all

Do not copy relocation and symbol information from the source file.

-g, --strip-debug

Do not copy debugging symbols from the source file.

--strip-unneeded

Strip all symbols that are not needed for relocation processing.

-K *symbolname*, --keep-symbol=*symbolname*

Copy only symbol *symbolname* from the source file. This option may be given more than once.

-N *symbolname*, --strip-symbol=*symbolname*

Do not copy symbol *symbolname* from the source file. This option may be given more than once.

-L *symbolname*, --localize-symbol=*symbolname*

Make symbol *symbolname* local to the file, so that it is not visible externally. This option may be given more than once.

-W *symbolname*, --weaken-symbol=*symbolname*

Make symbol *symbolname* weak. This option may be given more than once.

-x, --discard-all

Do not copy non-global symbols from the source file.

-X, --discard-locals

Do not copy compiler-generated local symbols. (These usually start with "L" or ".").

-b *byte*, --byte=*byte*

Keep only every *byteth* byte of the input file (header data is not affected). *byte* can be in the range from 0 to the *interleave*-1. This option is useful for creating files to program ROMs. It is typically used with an srec output target.

-i *interleave*, --interleave=*interleave*

Only copy one out of every *interleave* bytes. Which one to copy is selected by the **-b** or **--byte** option. The default is 4. The *interleave* is ignored if neither **-b** nor **--byte** is given.

-p, --preserve-dates

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

--debugging

Convert debugging information, if possible. This is not the default because only certain debugging formats are supported, and the conversion process can be time consuming.

--gap-fill=*val*

Fill gaps between sections with *val*. This operation applies to the *load address* (LMA) of the sections. It is done by increasing the size of the section with the lower address, and filling in the extra space created with *val*.

--pad-to=*address*

Pad the output file up to the load address *address*. This is done by increasing the size of the last section. The extra space is filled in with the value specified by **--gap-fill** (default zero).

--set-start=*val*

Set the start address of the new file to *val*. Not all object file formats support setting the start address.

--change-start=incr, --adjust-start=incr

Changes the start address by adding *incr*. Not all object file formats support setting the start address.

--change-addresses=incr, --adjust-vma=incr

Changes the address of all sections, as well as the start address, by adding *incr*. Some object file formats do not permit section addresses to be changed arbitrarily. Note that this does not relocate the sections; if the program expects sections to be loaded at a certain address, and this option is used to change the sections such that they are loaded at a different address, the program may fail.

--change-section-address section{=,+,-}val, --adjust-section-vma section{=,+,-}val

Set or changes the VMA and LMA addresses of the named *section*. If = is used, the section address is set to *val*. Otherwise, *val* is added to or subtracted from the section address. See the comments under **--change-addresses**, above. If *section* does not exist in the input file, a warning will be issued, unless **--no-change-warnings** is used.

--change-section-lma section{=,+,-}val

Set or change the LMA address of the named *section*. If = is used, the section address is set to *val*. Otherwise, *val* is added to or subtracted from the section address. See the comments under **--change-addresses**, above. If *section* does not exist in the input file, a warning will be issued, unless **--no-change-warnings** is used.

--change-section-vma section{=,+,-}val

Set or change the VMA address of the named *section*. If = is used, the section address is set to *val*. Otherwise, *val* is added to or subtracted from the section address. See the comments under **--change-addresses**, above. If *section* does not exist in the input file, a warning will be issued, unless **--no-change-warnings** is used.

--change-warnings, --adjust-warnings

If **--change-section-XXX** is used, and the named section does not exist, issue a warning. This is the default.

--no-change-warnings, --no-adjust-warnings

Do not issue a warning if **--change-section-XXX** is used, even if the named section does not exist.

--set-section-flags section=flags

Set the flags for the named section. The *flags* argument is a comma separated string of flag names. The recognized names are *alloc*, *contents*, *load*, *noload*, *readonly*, *code*, *data*, *rom*, *share*, and *debug*. Not all flags are meaningful for all object file formats.

--add-section sectionname=filename

Add a new section named *sectionname* while copying the file. The contents of the new section are taken from the file *filename*. The size of the section will be the size of the file. This option only works on file formats which can support sections with arbitrary names.

--change-leading-char

Some object file formats use special characters at the start of symbols. The most common such character is underscore, which compilers often add before every symbol. This option tells **objcopy** to change the leading character of every symbol when it converts between object file formats. If the object file formats use the same leading character, this option has no effect. Otherwise, it will add a character, or remove a character, or change a character, as appropriate.

--remove-leading-char

If the first character of a global symbol is a special symbol leading character used by the object file format, remove the character. The most common symbol leading character is underscore. This option will remove a leading underscore from all global symbols. This can be useful if you want to link together objects of different file formats with different conventions for symbol names. This is different from **--change-leading-char** because it always changes the symbol name when appropriate, regardless of the object file format of the output.

- srec-len=val**
Meaningful only for srec output. Set the length of the Srecords to *val*. This length covers both the address, data and crc fields.
- srec-forceS3**
Meaningful only for srec output. Avoid generation of S1/S2 records, creating S3-only record format.
- redefine-sym old=new**
Change the name of symbol *old* to *new*. This can be useful when one is trying link two things together for which you have no source, and there are name collisions.
- weaken**
Change all global symbols in the file to be weak.
- v, --verbose**
Verbose output: list all object files modified. In the case of archives, "**objcopy -V**" lists all members of the archive.
- V, --version**
Show the version number of **objcopy** and exit.
- help** Show a summary of the options to **objcopy** and exit.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (June 1993).

COPYING

Copyright (c) 1993, 94, 95, 96, 97, 98, 1999, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

objdump – display information from object files.

SYNOPSIS**objdump**

```
[ -a|--archive-headers ] [ -b bfdname | --target=bfdname ] [ -C|--demangle ]
[ --debugging ] [ -d|--disassemble ] [ -D|--disassemble-all ] [ --disassemble-zeroes ]
[ -EB|--EL|--endian={big/little} ] [ -f|--file-headers ] [ -h|--section-headers | --headers ]
[ -i|--info ] [ -j section | --section=section ] [ -l|--line-numbers ] [ -m machine |
--architecture=machine ] [ -p|--private-headers ] [ --prefix-addresses ] [ -r|--reloc ]
[ -R|--dynamic-reloc ] [ -s|--full-contents ] [ -S|--source ] [ --[no-]show-raw-insn ]
[ --stabs ] [ -t|--syms ] [ -T|--dynamic-syms ] [ -x|--all-headers ]
[ --start-address=address ] [ --stop-address=address ] [ --adjust-vma=offset ] [ --version ]
[ --help ] objfile...
```

DESCRIPTION

objdump displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

objfile... are the object files to be examined. When you specify archives, **objdump** shows information on each of the member object files.

OPTIONS

Where long and short forms of an option are shown together, they are equivalent. At least one option besides **-l** (**--line-numbers**) must be given.

-a**--archive-headers**

If any files from *objfile* are archives, display the archive header information (in a format similar to 'ls -l'). Besides the information you could list with 'ar tv', '**objdump -a**' shows the object file format of each archive member.

--adjust-vma=offset

When dumping information, first add *offset* to all the section addresses. This is useful if the section addresses do not correspond to the symbol table, which can happen when putting sections at particular addresses when using a format which can not represent section addresses, such as a.out.

-b bfdname**--target=bfdname**

Specify the object-code format for the object files to be *bfdname*. This may not be necessary; *objdump* can automatically recognize many formats. For example,

```
objdump -b oasys -m vax -h fu.o
```

display summary information from the section headers ('-h') of '**fu.o**', which is explicitly identified ('-m') as a Vax object file in the format produced by Oasys compilers. You can list the formats available with the '-i' option.

-C

--demangle

Decode (*demangle*) low-level symbol names into user-level names. Besides removing any initial underscore prepended by the system, this makes C++ function names readable.

--debugging

Display debugging information. This attempts to parse debugging information stored in the file and print it out using a C like syntax. Only certain types of debugging information have been implemented.

-d**--disassemble**

Display the assembler mnemonics for the machine instructions from *objfile*. This option only disassembles those sections which are expected to contain instructions.

-D**--disassemble-all**

Like **-d**, but disassemble the contents of all sections, not just those expected to contain instructions.

--prefix-addresses

When disassembling, print the complete address on each line. This is the older disassembly format.

--disassemble-zeroes

Normally the disassembly output will skip blocks of zeroes. This option directs the disassembler to disassemble those blocks, just like any other data.

-EB**-EL****--endian={big/little}**

Specify the endianness of the object files. This only affects disassembly. This can be useful when disassembling a file format which does not describe endianness information, such as S-records.

-f**--file-headers**

Display summary information from the overall header of each file in *objfile*.

-h**--section-headers****--headers**

Display summary information from the section headers of the object file.

--help Print a summary of the options to **objdump** and exit.

-i

--info Display a list showing all architectures and object formats available for specification with **-b** or **-m**.

- j** *name*
--section=*name*
Display information only for section *name*.
- l**
--line-numbers
Label the display (using debugging information) with the filename and source line numbers corresponding to the object code shown. Only useful with **-d**, **-D**, or **-r**.
- m** *machine*
--architecture=*machine*
Specify the architecture to use when disassembling object files. This can be useful when disassembling object files which do not describe architecture information, such as S-records. You can list the available architectures with the **-i** option.
- p**
--private-headers
Print information that is specific to the object file format. The exact information printed depends upon the object file format. For some object file formats, no additional information is printed.
- r**
--reloc
Print the relocation entries of the file. If used with **-d** or **-D**, the relocations are printed interspersed with the disassembly.
- R**
--dynamic-reloc
Print the dynamic relocation entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries.
- s**
--full-contents
Display the full contents of any sections requested.
- S**
--source
Display source code intermixed with disassembly, if possible. Implies **-d**.
- show-raw-insn**
When disassembling instructions, print the instruction in hex as well as in symbolic form. This is the default except when **--prefix-addresses** is used.
- no-show-raw-insn**
When disassembling instructions, do not print the instruction bytes. This is the default when **--prefix-addresses** is used.

--stabs

Display the contents of the .stab, .stab.index, and .stab.excl sections from an ELF file. This is only useful on systems (such as Solaris 2.0) in which .stab debugging symbol-table entries are carried in an ELF section. In most other file formats, debugging symbol-table entries are interleaved with linkage symbols, and are visible in the **--syms** output.

--start-address=address

Start displaying data at the specified address. This affects the output of the **-d**, **-r** and **-s** options.

--stop-address=address

Stop displaying data at the specified address. This affects the output of the **-d**, **-r** and **-s** options.

-t**--syms**

Symbol Table. Print the symbol table entries of the file. This is similar to the information provided by the '**nm**' program.

-T**--dynamic-syms**

Dynamic Symbol Table. Print the dynamic symbol table entries of the file. This is only meaningful for dynamic objects, such as certain types of shared libraries. This is similar to the information provided by the '**nm**' program when given the **-D** (**--dynamic**) option.

--version

Print the version number of **objdump** and exit.

-x**--all-headers**

Display all available header information, including the symbol table and relocation entries. Using '**-x**' is equivalent to specifying all of '**-a -f -h -r -t**'.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991); **nm**(1).

COPYING

Copyright (c) 1991, 92, 93, 94, 95, 1996 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

ranlib – generate index to archive.

SYNOPSIS

ranlib [**-v** | **-V**] *archive*

DESCRIPTION

ranlib generates an index to the contents of an archive, and stores it in the archive. The index lists each symbol defined by a member of an archive that is a relocatable object file.

You may use '**nm -s**' or '**nm --print-armap**' to list this index.

An archive with such an index speeds up linking to the library, and allows routines in the library to call each other without regard to their placement in the archive.

The GNU **ranlib** program is another form of GNU **ar**; running **ranlib** is completely equivalent to executing '**ar -s**'.

OPTIONS

-v Print the version number of **ranlib** and exit.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991); **ar(1)**, **nm(1)**.

COPYING

Copyright (c) 1991, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

size – list section sizes and total size.

SYNOPSIS

size [**-A** | **-B** | **--format=compatibility**] [**--help**] [**-d** | **-o** | **-x** | **--radix=number**] [**--target=bfdname**] [**-V** | **--version**] *objfile*...

DESCRIPTION

The GNU **size** utility lists the section sizes—and the total size—for each of the object files *objfile* in its argument list. By default, one line of output is generated for each object file or each module in an archive.

OPTIONS

-A

-B

--format *compatibility*

Using one of these options, you can choose whether the output from GNU **size** resembles output from System V **size** (using '**-A**', or '**--format=sysv**'), or Berkeley **size** (using '**-B**', or '**--format=berkeley**'). The default is the one-line format similar to Berkeley's.

--help Show a summary of acceptable arguments and options.

-d

-o

-x

--radix *number*

Using one of these options, you can control whether the size of each section is given in decimal ('**-d**', or '**--radix 10**'); octal ('**-o**', or '**--radix 8**'); or hexadecimal ('**-x**', or '**--radix 16**'). In '**--radix number**', only the three values (8, 10, 16) are supported. The total size is always given in two radices; decimal and hexadecimal for '**-d**' or '**-x**' output, or octal and hexadecimal if you're using '**-o**'.

--target *bfdname*

You can specify a particular object-code format for *objfile* as *bfdname*. This may not be necessary; *size* can automatically recognize many formats. See **objdump(1)** for information on listing available formats.

-V

--version

Display version number information on **size** itself.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991); **ar(1)**, **objdump(1)**.

COPYING

Copyright (c) 1991, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

strings – print the strings of printable characters in files

SYNOPSIS

strings [**-a** | **--all**] [**-f** **--print-file-name**] [**-o**] [**--help**] [**-v** | **--version**] [**-n** *min len* | *min len*] [**--bytes=***min len*] [**-t** {*o,x,d*} [**--target=***bfdname*] | **--radix={o,x,d}**] *Ýle*

DESCRIPTION

For each *Ýle* given, GNU **strings** prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

strings is mainly useful for determining the contents of non-text files.

OPTIONS

The long and short forms of options, shown here as alternatives, are equivalent.

-a

--all

– Do not scan only the initialized and loaded sections of object files; scan the whole files.

-f

--print-file-name

Print the name of the file before each string.

--help Print a summary of the options to **strings** on the standard output and exit.

-v

--version

Print the version number of **strings** on the standard output and exit.

-n *min len*

–min len

--bytes=*min len*

Print sequences of characters that are at least *min len* characters long, instead of the default 4.

-t {*o,x,d*}

--radix={o,x,d}

Print the offset within the file before each string. The single character argument specifies the radix of the offset—octal, hexadecimal, or decimal.

--target=*bfdname*

Specify an object code format other than your system's default format. See **objdump(1)**, for information on listing available formats.

-o Like **-t o**.

SEE ALSO

'**binutils**' entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991); **ar(1)**, **nm(1)**, **objdump(1)**, **ranlib(1)**.

COPYING

Copyright (c) 1993, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

strip – Discard symbols from object files.

SYNOPSIS

```
strip  [ -F bfdname | --target=bfdname ] [ -I bfdname | --input-target=bfdname ]
        [ -O bfdname | --output-target=bfdname ]
        [ -R sectionname | --remove-section=sectionname ] [ -s | --strip-all ]
        [ -S | -g | --strip-debug ] [ --strip-unneeded ] [ -x | --discard-all ] [ -X | --discard-locals ]
        [ -K symbolname | --keep-symbol=symbolname ]
        [ -N symbolname | --strip-symbol=symbolname ] [ -o file ] [ -p | --preserve-dates ]
        [ -v | --verbose ] [ -V | --version ] [ -V | --help ] objfile . .
```

DESCRIPTION

GNU **strip** discards all symbols from the object files *objfile*. The list of object files may include archives. At least one object file must be given.

strip modifies the files named in its argument, rather than writing modified copies under different names.

OPTIONS

-F *bfdname*

--target=*bfdname*

Treat the original *objfile* as a file with the object code format *bfdname*, and rewrite it in the same format.

--help Show a summary of the options to **strip** and exit.

-I *bfdname*

--input-target=*bfdname*

Treat the original *objfile* as a file with the object code format *bfdname*.

-O *bfdname*

--output-target=*bfdname*

Replace *objfile* with a file in the output format *bfdname*.

-R *sectionname*

--remove-section=*sectionname*

Remove the named section from the file. This option may be given more than once. Note that using this option inappropriately may make the object file unusable.

-s

--strip-all

Remove all symbols.

-S

-g

--strip-debug

Remove debugging symbols only.

--strip-unneeded

Strip all symbols that are not needed for relocation processing.

-N *symbolname***--strip-symbol=*symbolname***

Remove symbol *symbolname* from the source file. This option may be given more than once, and may be combined with other strip options.

-o *file* Put the stripped output in *file*, rather than replacing the existing file. When this argument is used, only one *outfile* argument may be specified.

-p**--preserve-dates**

Preserve the access and modification dates of the file.

-x**--discard-all**

Remove non-global symbols.

-X**--discard-locals**

Remove compiler-generated local symbols. (These usually start with “L” or “.”.)

-K *symbolname*, --keep-symbol=*symbolname*

Copy only symbol *symbolname* from the source file. This option may be given more than once.

-N *symbolname*, --strip-symbol=*symbolname*

Do not copy symbol *symbolname* from the source file. This option may be given more than once, and may be combined with strip options other than **-K**.

-v**--verbose**

Verbose output: list all object files modified. In the case of archives, **strip -v** lists all members of the archive.

-V**--version**

Show the version number for **strip** and exit.

SEE ALSO

‘**binutils**’ entry in **info**; *The GNU Binary Utilities*, Roland H. Pesch (October 1991).

COPYING

Copyright (c) 1991, 2000 Free Software Foundation, Inc.

This document is distributed under the terms of the GNU Free Documentation License, version 1.1. That license is described in the sources for this manual page, but it is not displayed here in order to make this manual more concise. Copies of this license can also be obtained from: <http://www.gnu.org/copyleft/>.

NAME

cpp – The C Preprocessor

SYNOPSIS

```
cpp [-P] [-C] [-gcc] [-traditional]
    [-undef] [-trigraphs] [-pedantic]
    [-Wwarn...] [-Idir...]
    [-Dmacro[=defn]...] [-Umacro]
    [-Apredicate(answer)]
    [-M|-MM][-MG][-MFfilename]
    [-MP][-MQtarget...][-MTtarget...]
    [-x language] [-std=standard]
    infile outfile
```

Only the most useful options are listed here; see below for the remainder.

DESCRIPTION

The C preprocessor is a *macro processor* that is used automatically by the C compiler to transform your program before actual compilation. It is called a macro processor because it allows you to define *macros*, which are brief abbreviations for longer constructs.

The C preprocessor is intended only for macro processing of C, C++ and Objective C source files. For macro processing of other files, you are strongly encouraged to use alternatives like M4, which will likely give you better results and avoid many problems. For example, normally the C preprocessor does not preserve arbitrary whitespace verbatim, but instead replaces each sequence with a single space.

For use on C-like source files, the C preprocessor provides four separate facilities that you can use as you see fit:

- Inclusion of header files. These are files of declarations that can be substituted into your program.
- Macro expansion. You can define *macros*, which are abbreviations for arbitrary fragments of C code, and then the C preprocessor will replace the macros with their definitions throughout the program.
- Conditional compilation. Using special preprocessing directives, you can include or exclude parts of the program according to various conditions.
- Line control. If you use a program to combine or rearrange source files into an intermediate file which is then compiled, you can use line control to inform the compiler of where each source line originally came from.

C preprocessors vary in some details. This manual discusses the GNU C preprocessor, which provides a small superset of the features of ISO Standard C.

In its default mode, the GNU C preprocessor does not do a few things required by the standard. These are features which are rarely, if ever, used, and may cause surprising changes to the meaning of a program which does not expect them. To get strict ISO Standard C, you should use the **-std=c89** or **-std=c99** options, depending on which version of the standard you want. To get all the mandatory diagnostics, you must also use **-pedantic**.

OPTIONS

The C preprocessor expects two file names as arguments, *infile* and *outfile*. The preprocessor reads *infile* together with any other files it specifies with **#include**. All the output generated by the combined input files is written in *outfile*.

Either *infile* or *outfile* may be -, which as *infile* means to read from standard input and as *outfile* means to write to standard output. Also, if either file is omitted, it means the same as if - had been specified for that file.

Here is a table of command options accepted by the C preprocessor. These options can also be given when compiling a C program; they are passed along automatically to the preprocessor when it is invoked by the compiler.

- P** Inhibit generation of `#`-lines with line-number information in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code and will be sent to a program which might be confused by the `#`-lines.
- C** Do not discard comments. All comments are passed through to the output file, except for comments in processed directives, which are deleted along with the directive. Comments appearing in the expansion list of a macro will be preserved, and appear in place wherever the macro is invoked.

You should be prepared for side effects when using **-C**; it causes the preprocessor to treat comments as tokens in their own right. For example, macro redefinitions that were trivial when comments were replaced by a single space might become significant when comments are retained. Also, comments appearing at the start of what would be a directive line have the effect of turning that line into an ordinary source line, since the first token on the line is no longer a `#`.

-traditional

Try to imitate the behavior of old-fashioned C, as opposed to ISO C.

- Traditional macro expansion pays no attention to single-quote or double-quote characters; macro argument symbols are replaced by the argument values even when they appear within apparent string or character constants.
- Traditionally, it is permissible for a macro expansion to end in the middle of a string or character constant. The constant continues into the text surrounding the macro call.
- However, traditionally the end of the line terminates a string or character constant, with no error.
- In traditional C, a comment is equivalent to no text at all. (In ISO C, a comment counts as whitespace.)
- Traditional C does not have the concept of a “preprocessing number”. It considers **1.0e+4** to be three tokens: **1.0e**, **+**, and **4**.
- A macro is not suppressed within its own definition, in traditional C. Thus, any macro that is used recursively inevitably causes an error.
- The character `#` has no special meaning within a macro definition in traditional C.
- In traditional C, the text at the end of a macro expansion can run together with the text after the macro call, to produce a single token. (This is impossible in ISO C.)
- None of the GNU extensions to the preprocessor are available in **-traditional** mode.

Use the **-traditional** option when preprocessing Fortran code, so that single-quotes and double-quotes within Fortran comment lines (which are generally not recognized as such by the preprocessor) do not cause diagnostics about unterminated character or string constants.

However, this option does not prevent diagnostics about unterminated comments when a C-style comment appears to start, but not end, within Fortran-style commentary.

So, the following Fortran comment lines are accepted with **-traditional**:

```
C This isn't an unterminated character constant
C Neither is "20000000000, an octal constant
C in some dialects of Fortran
```

However, this type of comment line will likely produce a diagnostic, or at least unexpected output from the preprocessor, due to the unterminated comment:

```
C Some Fortran compilers accept /* as starting
C an inline comment.
```

Note that `g77` automatically supplies the **-traditional** option when it invokes the preprocessor. However, a future version of `g77` might use a different, more-Fortran-aware preprocessor in place of `cpp`.

-trigraphs

Process ISO standard trigraph sequences. These are three-character sequences, all starting with **??**, that are defined by ISO C to stand for single characters. For example, **??/** stands for ****, so **'??/n'** is a character constant for a newline. By default, GCC ignores trigraphs, but in standard-conforming modes it converts them. See the **-std** option.

The nine trigraph sequences are

??(-> [

??) -> **]**

??<

-> **{**

??>

-> **}**

??=

-> **#**

??/ -> ****

??' -> **^**

??! -> **|**

??- -> **~**

Trigraph support is not popular, so many compilers do not implement it properly. Portable code should not rely on trigraphs being either converted or ignored.

-pedantic

Issue warnings required by the ISO C standard in certain cases such as when text other than a comment follows **#else** or **#endif**.

-pedantic-errors

Like **-pedantic**, except that errors are produced rather than warnings.

-Wcomment**-Wcomments**

(Both forms have the same effect). Warn whenever a comment-start sequence **/*** appears in a **/*** comment, or whenever a backslash-newline appears in a **//** comment.

-Wtrigraphs

Warn if any trigraphs are encountered. This option used to take effect only if **-trigraphs** was also specified, but now works independently. Warnings are not given for trigraphs within comments, as we feel this is obnoxious.

-Wwhite-space

Warn about possible white space confusion, e.g. white space between a backslash and a newline.

-Wall

Requests **-Wcomment**, **-Wtrigraphs**, and **-Wwhite-space** (but not **-Wtraditional** or **-Wundef**).

-Wtraditional

Warn about certain constructs that behave differently in traditional and ISO C.

- Macro parameters that appear within string literals in the macro body. In traditional C macro replacement takes place within string literals, but does not in ISO C.
- In traditional C, some preprocessor directives did not exist. Traditional preprocessors would only consider a line to be a directive if the **#** appeared in column 1 on the line. Therefore **-Wtraditional** warns about directives that traditional C understands but would ignore because the **#** does not appear as the first character on the line. It also suggests you hide directives like **#pragma** not understood by traditional C by indenting them. Some traditional implementations would not

recognise **#elif**, so it suggests avoiding it altogether.

- A function-like macro that appears without arguments.
- The unary plus operator.
- The 'U' integer constant suffix. (Traditional C does support the 'L' suffix on integer constants.) Note, these suffixes appear in macros defined in the system headers of most modern systems, e.g. the `_MIN/_MAX` macros in `limits.h`. Use of these macros can lead to spurious warnings as they do not necessarily reflect whether the code in question is any less portable to traditional C given that suitable backup definitions are provided.

-Wundef

Warn if an undefined identifier is evaluated in an **#if** directive.

-I directory

Add the directory *directory* to the head of the list of directories to be searched for header files. This can be used to override a system header file, substituting your own version, since these directories are searched before the system header file directories. If you use more than one **-I** option, the directories are scanned in left-to-right order; the standard system directories come after.

- I-** Any directories specified with **-I** options before the **-I-** option are searched only for the case of **#include "file"**; they are not searched for **#include <file>**.

If additional directories are specified with **-I** options after the **-I-**, these directories are searched for all **#include** directives.

In addition, the **-I-** option inhibits the use of the current directory as the first search directory for **#include "file"**. Therefore, the current directory is searched only if it is requested explicitly with **-I.** Specifying both **-I-** and **-I.** allows you to control precisely which directories are searched before the current one and which are searched after.

-nostdinc

Do not search the standard system directories for header files. Only the directories you have specified with **-I** options (and the current directory, if appropriate) are searched.

By using both **-nostdinc** and **-I-**, you can limit the include-file search path to only those directories you specify explicitly.

-nostdinc++

Do not search for header files in the C++-specific standard directories, but do still search the other standard directories. (This option is used when building the C++ library.)

-remap

When searching for a header file in a directory, remap file names if a file named *header.gcc* exists in that directory. This can be used to work around limitations of file systems with file name restrictions. The *header.gcc* file should contain a series of lines with two tokens on each line: the first token is the name to map, and the second token is the actual name to use.

-D name

Predefine *name* as a macro, with definition **1**.

-D name=definition

Predefine *name* as a macro, with definition *definition*. There are no restrictions on the contents of *definition*, but if you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax. If you use more than one **-D** for the same *name*, the rightmost definition takes effect.

Any **-D** and **-U** options on the command line are processed in order, and always before **-imacros file**, regardless of the order in which they are written.

-U *name*

Do not predefine *name*.

Any **-D** and **-U** options on the command line are processed in order, and always before **-imacros *file***, regardless of the order in which they are written.

-undef

Do not predefine any nonstandard macros.

-gcc

Define the macros `__GNUC__`, `__GNUC_MINOR__` and `__GNUC_PATCHLEVEL__`. These are defined automatically when you use **gcc -E**; you can turn them off in that case with **-no-gcc**.

-A *predicate=answer*

Make an assertion with the predicate *predicate* and answer *answer*. This form is preferred to the older form **-A *predicate(answer)***, which is still supported, because it does not use shell special characters.

-A -*predicate=answer*

Disable an assertion with the predicate *predicate* and answer *answer*. Specifying no predicate, by **-A-** or **-A -**, disables all predefined assertions and all assertions preceding it on the command line; and also undefines all predefined macros and all macros preceding it on the command line.

-dM

Instead of outputting the result of preprocessing, output a list of **#define** directives for all the macros defined during the execution of the preprocessor, including predefined macros. This gives you a way of finding out what is predefined in your version of the preprocessor; assuming you have no file **foo.h**, the command

```
touch foo.h; cpp -dM foo.h
```

will show the values of any predefined macros.

-dD

Like **-dM** except in two respects: it does *not* include the predefined macros, and it outputs *both* the **#define** directives and the result of preprocessing. Both kinds of output go to the standard output file.

-dN

Like **-dD**, but emit only the macro names, not their expansions.

-dI

Output **#include** directives in addition to the result of preprocessing.

-M

Instead of outputting the result of preprocessing, output a rule suitable for make describing the dependencies of the main source file. The preprocessor outputs one make rule containing the object file name for that source file, a colon, and the names of all the included files, including those coming from **-include** or **-imacros** command line options. Unless specified explicitly (with **-MT** or **-MQ**), the object file name consists of the basename of the source file with any suffix replaced with object file suffix. If there are many included files then the rule is split into several lines using `\-newline`.

-MM

Like **-M**, but mention only the files included with **#include "file"** or with **-include** or **-imacros** command line options. System header files included with **#include <file>** are omitted.

-MF *file*

When used with **-M** or **-MM**, specifies a file to write the dependencies to. This allows the preprocessor to write the preprocessed file to stdout normally. If no **-MF** switch is given, CPP sends the rules to stdout and suppresses normal preprocessed output.

-MG

When used with **-M** or **-MM**, **-MG** says to treat missing header files as generated files and assume they live in the same directory as the source file. It suppresses preprocessed output, as a missing header file is ordinarily an error.

This feature is used in automatic updating of makefiles.

-MP

This option instructs CPP to add a phony target for each dependency other than the main file, causing each to depend on nothing. These dummy rules work around errors make gives if you remove header files without updating the Makefile to match.

This is typical output:-

```
/tmp/test.o: /tmp/test.c /tmp/test.h

/tmp/test.h:
```

-MQ *target*

-MT *target*

By default CPP uses the main file name, including any path, and appends the object suffix, normally “.o”, to it to obtain the name of the target for dependency generation. With **-MT** you can specify a target yourself, overriding the default one.

If you want multiple targets, you can specify them as a single argument to **-MT**, or use multiple **-MT** options.

The targets you specify are output in the order they appear on the command line. **-MQ** is identical to **-MT**, except that the target name is quoted for Make, but with **-MT** it isn't. For example, **-MT** '\$(objpfx)foo.o' gives

```
$(objpfx)foo.o: /tmp/foo.c
```

but **-MQ** '\$(objpfx)foo.o' gives

```
$$$(objpfx)foo.o: /tmp/foo.c
```

The default target is automatically quoted, as if it were given with **-MQ**.

-H Print the name of each header file used, in addition to other normal activities.

-imacros *file*

Process *file* as input, discarding the resulting output, before processing the regular input file. Because the output generated from *file* is discarded, the only effect of **-imacros** *file* is to make the macros defined in *file* available for use in the main input.

-include *file*

Process *file* as input, and include all the resulting output, before processing the regular input file.

-idirafter *dir*

Add the directory *dir* to the second include path. The directories on the second include path are searched when a header file is not found in any of the directories in the main include path (the one that **-I** adds to).

-iprefix *prefix*

Specify *prefix* as the prefix for subsequent **-iwithprefix** options. If the prefix represents a directory, you should include the final /.

-iwithprefix *dir*

Add a directory to the second include path. The directory's name is made by concatenating *prefix* and *dir*, where *prefix* was specified previously with **-iprefix**.

-isystem *dir*

Add a directory to the beginning of the second include path, marking it as a system directory, so that it gets the same special treatment as is applied to the standard system directories.

-x c

-x c++

-x objective-c

-x assembler-with-cpp

Specify the source language: C, C++, Objective-C, or assembly. This has nothing to do with standards conformance or extensions; it merely selects which base syntax to expect. If you give none of these options, cpp will deduce the language from the extension of the source file: **.c**, **.cc**, **.m**, or **.S**. Some other common extensions for C++ and assembly are also recognized. If cpp does not recognize the extension, it will treat the file as C; this is the most generic mode.

Note: Previous versions of cpp accepted a **-lang** option which selected both the language and the standards conformance level. This option has been removed, because it conflicts with the **-l** option.

-std=standard

-ansi

Specify the standard to which the code should conform. Currently cpp only knows about the standards for C; other language standards will be added in the future.

standard may be one of:

The ISO C standard from 1990. **c89** is the customary shorthand for this version of the standard.

The **-ansi** option is equivalent to **-std=c89**. The 1990 C standard, as amended in 1994. The revised ISO C standard, published in December 1999. Before publication, this was known as C9X. The 1990 C standard plus GNU extensions. This is the default. The 1999 C standard plus GNU extensions.

-ftabstop=NUMBER

Set the distance between tab stops. This helps the preprocessor report correct column numbers in warnings or errors, even if tabs appear on the line. Values less than 1 or greater than 100 are ignored. The default is 8.

-\$ Forbid the use of **\$** in identifiers. The C standard allows implementations to define extra characters that can appear in identifiers. By default the GNU C preprocessor permits **\$**, a common extension.

SEE ALSO

gcc(1), *as*(1), *ld*(1), and the Info entries for *cpp*, *gcc*, and *binutils*.

COPYRIGHT

Copyright (c) 1987, 1989, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

NAME

gcov – coverage testing tool

SYNOPSIS

gcov [**-b**] [**-c**] [**-v**] [**-n**] [**-l**] [**-f**] [**-o** *directory*] *sourcefile*

DESCRIPTION

gcov is a test coverage program. Use it in concert with GNU CC to analyze your programs to help create more efficient, faster running code. You can use gcov as a profiling tool to help discover where your optimization efforts will best affect your code. You can also use gcov along with the other profiling tool, gprof, to assess which parts of your code use the greatest amount of computing time.

Profiling tools help you analyze your code's performance. Using a profiler such as gcov or gprof, you can find out some basic performance statistics, such as:

- how often each line of code executes
- what lines of code are actually executed
- how much computing time each section of code uses

Once you know these things about how your code works when compiled, you can look at each module to see which modules should be optimized. gcov helps you determine where to work on optimization.

Software developers also use coverage testing in concert with testsuites, to make sure software is actually good enough for a release. Testsuites can verify that a program works as expected; a coverage program tests to see how much of the program is exercised by the testsuite. Developers can then determine what kinds of test cases need to be added to the testsuites to create both better testing and a better final product.

You should compile your code without optimization if you plan to use gcov because the optimization, by combining some lines of code into one function, may not give you as much information as you need to look for 'hot spots' where the code is using a great deal of computer time. Likewise, because gcov accumulates statistics by line (at the lowest resolution), it works best with a programming style that places only one statement on each line. If you use complicated macros that expand to loops or to other control structures, the statistics are less helpful—they only report on the line where the macro call appears. If your complex macros behave like functions, you can replace them with inline functions to solve this problem.

gcov creates a logfile called *sourcefile.gcov* which indicates how many times each line of a source file *sourcefile.c* has executed. You can use these logfiles along with gprof to aid in fine-tuning the performance of your programs. gprof gives timing information you can use along with the information you get from gcov.

gcov works only on code compiled with GNU CC. It is not compatible with any other profiling or test coverage mechanism.

OPTIONS

Write branch frequencies to the output file, and write branch summary info to the standard output. This option allows you to see how often each branch in your program was taken. Write branch frequencies as the number of branches taken, rather than the percentage of branches taken. Display the gcov version number (on the standard error stream). Do not create the gcov output file. Create long file names for included source files. For example, if the header file **x.h** contains code, and was included in the file **a.c**, then running gcov on the file **a.c** will produce an output file called **a.c.x.h.gcov** instead of **x.h.gcov**. This can be useful if **x.h** is included in multiple source files. Output summaries for each function in addition to the file level summary. The directory where the object files live. Gcov will search for .bb, .bbg, and .da files in this directory.

When using gcov, you must first compile your program with two special GNU CC options: **-fprofile-arcs** **-ftest-coverage**. This tells the compiler to generate additional information needed by gcov (basically a flow graph of the program) and also includes additional code in the object files for generating the extra profiling information needed by gcov. These additional files are placed in the directory where the source code is located.

Running the program will cause profile output to be generated. For each source file compiled with

`-fprofile-arcs`, an accompanying `.da` file will be placed in the source directory.

Running `gcov` with your program's source file names as arguments will now produce a listing of the code along with frequency of execution for each line. For example, if your program is called **tmp.c**, this is what you see when you use the basic `gcov` facility:

```
$ gcc -fprofile-arcs -ftest-coverage tmp.c
$ a.out
$ gcov tmp.c
 87.50% of 8 source lines executed in file tmp.c
Creating tmp.c.gcov.
```

The file *tmp.c.gcov* contains output from `gcov`. Here is a sample:

```

                                main()
                                {
1      int i, total;

1      total = 0;

11     for (i = 0; i < 10; i++)
10     total += i;

1      if (total != 45)
#####    printf ("Failure\n");
        else
1      printf ("Success\n");
1      }

```

When you use the `-b` option, your output looks like this:

```
$ gcov -b tmp.c
 87.50% of 8 source lines executed in file tmp.c
 80.00% of 5 branches executed in file tmp.c
 80.00% of 5 branches taken at least once in file tmp.c
 50.00% of 2 calls executed in file tmp.c
Creating tmp.c.gcov.
```

Here is a sample of a resulting *tmp.c.gcov* file:

```

                                main()
                                {
1      int i, total;

1      total = 0;

11     for (i = 0; i < 10; i++)
branch 0 taken = 91%
branch 1 taken = 100%
branch 2 taken = 100%
10     total += i;

```

```

        1      if (total != 45)
branch 0 taken = 100%
        #####      printf ("Failure\n");
call 0 never executed
branch 1 never executed
        else
        1      printf ("Success\n");
call 0 returns = 100%
        1      }

```

For each basic block, a line is printed after the last line of the basic block describing the branch or call that ends the basic block. There can be multiple branches and calls listed for a single source line if there are multiple basic blocks that end on that line. In this case, the branches and calls are each given a number. There is no simple way to map these branches and calls back to source constructs. In general, though, the lowest numbered branch or call will correspond to the leftmost construct on the source line.

For a branch, if it was executed at least once, then a percentage indicating the number of times the branch was taken divided by the number of times the branch was executed will be printed. Otherwise, the message “never executed” is printed.

For a call, if it was executed at least once, then a percentage indicating the number of times the call returned divided by the number of times the call was executed will be printed. This will usually be 100%, but may be less for functions call `exit` or `longjmp`, and thus may not return every time they are called.

The execution counts are cumulative. If the example program were executed again without removing the `.da` file, the count for the number of times each line in the source was executed would be added to the results of the previous *run*(s). This is potentially useful in several ways. For example, it could be used to accumulate data over a number of program runs as part of a test verification suite, or to provide more accurate long-term information over a large number of program runs.

The data in the `.da` files is saved immediately before the program exits. For each source file compiled with `-fprofile-arcs`, the profiling code first attempts to read in an existing `.da` file; if the file doesn’t match the executable (differing number of basic block counts) it will ignore the contents of the file. It then adds in the new execution counts and finally writes the data to the file. If you plan to use `gcov` to help optimize your code, you must first compile your program with two special GNU CC options: **`-fprofile-arcs -ftest-coverage`**. Aside from that, you can use any other GNU CC options; but if you want to prove that every single line in your program was executed, you should not compile with optimization at the same time. On some machines the optimizer can eliminate some simple code lines by combining them with other lines. For example, code like this:

```

if (a != b)
    c = 1;
else
    c = 0;

```

can be compiled into one instruction on some machines. In this case, there is no way for `gcov` to calculate separate execution counts for each line because there isn’t separate code for each line. Hence the `gcov` output looks like this if you compiled the program with optimization:

```

100  if (a != b)
100    c = 1;
100  else
100    c = 0;

```

The output shows that this block of code, combined by optimization, executed 100 times. In one sense this result is correct, because there was only one instruction representing all four of these lines. However, the output does not indicate how many times the result was 0 and how many times the result was 1.

SEE ALSO

gcc(1) and the Info entry for *gcc*.

COPYRIGHT

Copyright (c) 1996, 1997, 1999, 2000 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be included in translations approved by the Free Software Foundation instead of in the original English.