GCC Command Options

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.

See section Compiling C++ Programs, for a summary of special options for compiling C++ programs.

The `gcc` program accepts options and file names as operands. Many options have multiletter 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.

- Option Summary: Brief list of all options, without explanations.
- Overall Options: Controlling the kind of output: an executable, object files, assembler files, or preprocessed source.
- Invoking G++: Compiling C++ programs.
- C Dialect Options: Controlling the variant of C language compiled.
- C++ Dialect Options: Variations on C++.
- Warning Options: How picky should the compiler be?
- Debugging Options: Symbol tables, measurements, and debugging dumps.
- Optimize Options: How much optimization?
- Preprocessor Options: Controlling header files and macro definitions. Also, getting dependency information for Make.
- Assembler Options: Passing options to the assembler.
- Link Options: Specifying libraries and so on.
- Directory Options: Where to find header files and libraries. Where to find the compiler executable files.
- Spec Files: How to pass switches to sub-processes.
- Target Options: Running a cross-compiler, or an old version of GCC.
- Submodel Options: Specifying minor hardware or convention variations, such as 68010 vs 68020.
- Code Gen Options: Specifying conventions for function calls, data layout and register usage.
- Environment Variables: Env vars that affect GCC.
- Running Protoize: Automatically adding or removing function prototypes.

Option Summary

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

Overall Options

See section Options Controlling the Kind of Output.

`-c -S -E -o` *`file`* `-pipe -v --help -x` *`language`*

C Language Options

See section Options Controlling C Dialect.

`-ansi -flang-isoc9x -fallow-single-precision -fcond-mismatch -fno-asm -fno-builtin -ffreestanding -fhosted -fsigned-bitfields -fsigned-char -funsigned-bitfields -funsigned-char -fwritable-strings -traditional -traditional-cpp -trigraphs`

C++ Language Options

See section Options Controlling C++ Dialect.

`-fno-access-control -fcheck-new -fconserve-space -fdollars-in-identifiers -fno-elide-constructors -fexternal-templates -ffor-scope -fno-for-scope -fno-gnu-keywords -fguiding-decls -fhonor-std -fhuge-objects -fno-implicit-templates -finit-priority -fno-implement-inlines -fname-mangling-version-`*`n`* `-fno-default-inline -foperator-names -fno-optional-diags -fpermissive -frepo -fstrict-prototype -fsquangle -ftemplate-depth-`*`n`* `-fvtable-thunks -nostdinc++ -Wctor-dtor-privacy -Wno-deprecated -Weffc++ -Wno-non-template-friend -Wnon-virtual-dtor -Wold-style-cast -Woverloaded-virtual -Wno-pmf-conversions -Wreorder -Wsign-promo -Wsynth`

Warning Options

See section Options to Request or Suppress Warnings.

`-fsyntax-only -pedantic -pedantic-errors -w -W -Wall -Waggregate-return -Wbad-function-cast -Wcast-align -Wcast-qual -Wchar-subscripts -Wcomment -Wconversion -Werror -Wformat -Wid-clash-`*`len`* `-Wimplicit -Wimplicit-int -Wimplicit-function-declaration -Wimport -Werror-implicit-function-declaration -Winline -Wlarger-than-`*`len`* `-Wlong-long -Wmain -Wmissing-declarations -Wmissing-noreturn -Wmissing-prototypes -Wmultichar -Wnested-externs -Wno-import -Wparentheses -Wpointer-arith -Wredundant-decls -Wreturn-type -Wshadow -Wsign-compare -Wstrict-prototypes -Wswitch -Wtraditional -Wtrigraphs -Wundef -Wuninitialized -Wunknown-pragmas -Wunreachable-code -Wunused -Wwrite-strings`

Debugging Options

See section Options for Debugging Your Program or GCC.

`-a -ax -d`*`letters`* `-fdump-unnumbered -fdump-translation-unit-`*`file`* `-fpretend-float -fprofile-arcs -ftest-coverage -g -g`*`level`* `-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 -save-temps -time`

Optimization Options

See section Options That Control Optimization.

```
-falign-functions=n  -falign-labels=n  -falign-loops=n
-falign-jumps=n  -fbranch-probabilities
-fcaller-saves  -fcse-follow-jumps  -fcse-skip-blocks
-fdelayed-branch   -fexpensive-optimizations
-ffast-math  -ffloat-store  -fforce-addr  -fforce-mem -fno-math-errno
-fdata-sections  -ffunction-sections  -fgcse
-finline-functions  -finline-limit=n  -fkeep-inline-functions
-fmove-all-movables  -fno-default-inline  -fno-defer-pop
-fno-function-cse  -fno-inline  -fno-peephole
-fomit-frame-pointer  -foptimize-register-moves  -fregmove
-frerun-cse-after-loop  -frerun-loop-opt  -freduce-all-givs
-fschedule-insns  -fschedule-insns2  -fstrength-reduce
-fstrict-aliasing  -fthread-jumps  -funroll-all-loops
-funroll-loops
-O  -O0  -O1  -O2  -O3 -Os
```

Preprocessor Options

See section Options Controlling the Preprocessor.

```
-Aquestion(answer)  -C  -dD  -dM  -dN
-Dmacro[=defn]  -E  -H
-idirafter dir
-include file  -imacros file
-iprefix file  -iwithprefix dir
-iwithprefixbefore dir  -isystem dir -isystem-c++ dir
-M  -MD  -MM  -MMD  -MG  -nostdinc  -P  -trigraphs
-undef  -Umacro  -Wp,option
```

Assembler Option

See section Passing Options to the Assembler.

```
-Wa,option
```

Linker Options

See section Options for Linking.

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

Directory Options

See section Options for Directory Search.

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

Target Options

See section Specifying Target Machine and Compiler Version.

```
-b machine  -V version
```

Machine Dependent Options

See section Hardware Models and Configurations.

```
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

VAX Options
-mg -mgnu -munix

SPARC Options
-mcpu=*cpu type*
-mtune=*cpu type*
-mcmodel=*code model*
-m32 -m64
-mapp-regs -mbroken-saverestore -mcypress -mepilogue
-mflat -mfpu -mhard-float -mhard-quad-float
-mimpure-text -mlive-g0 -mno-app-regs -mno-epilogue
-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
-mshort-load-bytes -mno-short-load-bytes -mshort-load-words -mno-short-load-words
-msoft-float -mhard-float -mfpe
-mthumb-interwork -mno-thumb-interwork
-mcpu= -march= -mfpe=
-mstructure-size-boundary=
-mbsd -mxopen -mno-symrename
-mabort-on-noreturn
-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
-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 -mno-new-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* -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
-membedded-pic -mfp32 -mfp64 -mgas -mgp32 -mgp64
-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-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

i386 Options
-mcpu=*cpu type*
-march=*cpu type*
-mieee-fp -mno-fancy-math-387
-mno-fp-ret-in-387 -msoft-float -msvr3-shlib
-mno-wide-multiply -mrtd -malign-double
-mreg-alloc=*list* -mregparm=*num*
-malign-jumps=*num* -malign-loops=*num*
-malign-functions=*num* -mpreferred-stack-boundary=*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
-m*cpu type* -masm-compat -mclean-linkage
-mcode-align -mcomplex-addr -mleaf-procedures
-mic-compat -mic2.0-compat -mic3.0-compat
-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 -mb -ml -mdalign -mrelax

System V Options
-Qy -Qn -YP,*paths* -Ym,*dir*

ARC Options
-EB -EL
-mmangle-cpu -mcpu=*cpu* -mtext=*text section*
-mdata=*data section* -mrodata=*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 -mnohimem

Code Generation Options

See section Options for Code Generation Conventions.

-fcall-saved-*reg* -fcall-used-*reg*
-fexceptions -ffixed-*reg* -finhibit-size-directive
-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
-fargument-alias -fargument-noalias
-fargument-noalias-global
-fleading-underscore

- Overall Options: Controlling the kind of output: an executable, object files, assembler files, or preprocessed source.
- C Dialect Options: Controlling the variant of C language compiled.
- C++ Dialect Options: Variations on C++.
- Warning Options: How picky should the compiler be?
- Debugging Options: Symbol tables, measurements, and debugging dumps.
- Optimize Options: How much optimization?
- Preprocessor Options: Controlling header files and macro definitions. Also, getting dependency information for Make.
- Assembler Options: Passing options to the assembler.
- Link Options: Specifying libraries and so on.
- Directory Options: Where to find header files and libraries. Where to find the compiler executable files.
- Spec Files: How to pass switches to sub-processes.
- Target Options: Running a cross-compiler, or an old version of GCC.

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.h

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

file.cc

file.cxx

file.cpp

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.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  objective-c  c++
c-header  cpp-output  c++-cpp-output
assembler  assembler-with-cpp
```

-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).

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.

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, the script 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. See section Options Controlling C Dialect, for explanations of options for languages related to C. See section Options Controlling C++ Dialect, for explanations of 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 ANSI standard C programs. In C++ mode, remove GNU extensions that conflict with ANSI C++. This turns off certain features of GCC that are incompatible with ANSI C (when compiling C code), or of ANSI 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 ANSI trigraph feature. For the C compiler, it disables recognition of C++ style `'//'` comments as well as the `inline` keyword. For the C++ compiler, `'-foperator-names'` is enabled as well. The alternate keywords `__asm__`, `__extension__`, `__inline__` and `__typeof__` continue to work despite `'-ansi'`. You would not want to use them in an ANSI 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-ANSI programs to be rejected gratuitously. For that, `'-pedantic'` is required in addition to `'-ansi'`. See section Options to Request or Suppress Warnings. 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 ANSI standard doesn't call for; this is to avoid interfering with any programs that might use these names for other things. The functions `alloca`, `abort`, `exit`, and `_exit` are not builtin functions when `'-ansi'` is used.

`-flang-isoc9x`

Enable support for features found in the C9X standard. In particular, enable support for the C9X `restrict` keyword. Even when this option is not specified, you can still use some C9X features in so far as they do not conflict with previous C standards. For example, you may use `__restrict__` even when -flang-isoc9x is not specified.

`-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, as it also disables the other, C++-specific, extension keywords such as `headof`.

`-fno-builtin`

Don't recognize builtin functions that do not begin with `'__builtin_'` as prefix. Currently, the functions affected include `abort`, `abs`, `alloca`, `cos`, `exit`, `fabs`, `ffs`, `labs`, `memcmp`, `memcpy`, `sin`, `sqrt`, `strcmp`, `strcpy`, and `strlen`. 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. The `'-ansi'` option prevents `alloca` and `ffs` from being builtin functions, since these

functions do not have an ANSI standard meaning.

`-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 ANSI C trigraphs. You don't want to know about this brain-damage. The `'-ansi'` option 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 ANSI 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 ANSI 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 ANSI C features. Some vendors are starting to ship systems with ANSI 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 ANSI C compilers, and other old C compilers. The predefined macro __STDC_VERSION__ is also not defined when you use '-traditional'. See section `Standard Predefined Macros' in *The C Preprocessor*, for more discussion of these and other predefined macros.
- 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.

-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 ANSI C features. Some vendors are starting to ship systems with ANSI 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 ANSI or GNU C conventions (the default).

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, ANSI 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.

`-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. See section Where's the Template?, for more information. 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. See section Where's the Template?, for more information. 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 draft 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 `classof`, `headof`, or `typeof` as a keyword, so that code can use these words as identifiers. You can use the keywords `__classof__`, `__headof__`, and `__typeof__` instead. `'-ansi'` implies `'-fno-gnu-keywords'`.

`-fguiding-decls`

Treat a function declaration with the same type as a potential function template instantiation as though it declares that instantiation, not a normal function. If a definition is given for the function later in the translation unit (or another translation unit if the target supports weak symbols), that definition will be used; otherwise the template will be instantiated. This behavior reflects the C++ language prior to September 1996, when guiding declarations were removed. This option implies `'-fname-mangling-version-0'`, and will not work with other name mangling versions. Like all options that change the ABI, all C++ code, *including libgcc.a* must be built with the same setting of this option.

`-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. See section Where's the Template?, for more information.

`-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.

`-finit-priority`

Support `'__attribute__ ((init_priority (n)))'` for controlling the order of initialization of file-scope objects. On ELF targets, this requires GNU ld 2.10 or later.

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

`-fname-mangling-version-`*n*

Control the way in which names are mangled. Version 0 is compatible with versions of g++ before 2.8. Version 1 is the default. Version 1 will allow correct mangling of function templates. For example, version 0 mangling does not mangle foo<int, double> and foo<int, char> given this declaration:

```
template <class T, class U> void foo(T t);
```

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

`-foperator-names`

Recognize the operator name keywords `and`, `bitand`, `bitor`, `compl`, `not`, `or` and `xor` as synonyms for the symbols they refer to. `'-ansi'` implies `'-foperator-names'`.

`-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'`. See section Where's the Template?, for more information.

`-fno-rtti`

Disable generation of the information used by C++ runtime type identification features (`'dynamic_cast'` and `'typeid'`). If you don't use those parts of the language (or exception handling, which uses `'dynamic_cast'` internally), you can save some space by using this flag.

`-fstrict-prototype`

Within an ‘`extern "C"`’ linkage specification, treat a function declaration with no arguments, such as ‘`int foo ();`’, as declaring the function to take no arguments. Normally, such a declaration means that the function `foo` can take any combination of arguments, as in C. ‘`-pedantic`’ implies ‘`-fstrict-prototype`’ unless overridden with ‘`-fno-strict-prototype`’. Specifying this option will also suppress implicit declarations of functions. This flag no longer affects declarations with C++ linkage.

`-fsquangle`

`-fno-squangle`

‘`-fsquangle`’ will enable a compressed form of name mangling for identifiers. In particular, it helps to shorten very long names by recognizing types and class names which occur more than once, replacing them with special short ID codes. This option also requires any C++ libraries being used to be compiled with this option as well. The compiler has this disabled (the equivalent of ‘`-fno-squangle`’) by default. Like all options that change the ABI, all C++ code, *including libgcc.a* must be built with the same setting of this option.

`-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.

`-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. See section Options That Control Optimization. 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. See section Deprecated Features.

-Wno-non-template-friend (C++ only)

Disable warnings when non-templatized 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 templatized 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 also be turned off with the flag ‘-fguiding-decls’, which activates the older, non-specification compiler code, or 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 derived class function declaration may be an error in defining a virtual function. In a derived class, the definitions of virtual functions must match the type signature of a virtual function declared in the base class. With this option, the compiler warns when you define a function with the same name as a virtual function, but with a type signature that does not match any declarations from the base class.

-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 enumeral 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 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 ANSI C and ISO C++; reject all programs that use forbidden extensions. Valid ANSI C and ISO C++ programs should compile properly with or without this option (though a rare few will require `'-ansi'`). 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. See section Alternate Keywords. This option is not intended to be *useful*; it exists only to satisfy pedants who would otherwise claim that GCC fails to support the ANSI standard. Some users try to use `'-pedantic'` to check programs for strict ANSI C conformance. They soon find that it does not do quite what they want: it finds some non-ANSI practices, but not all--only those for which ANSI C *requires* a diagnostic. A feature to report any failure to conform to ANSI 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.

-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 ();
    }
}
```

`-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`.

`-Wswitch`

Warn whenever a `switch` statement has an index of enumeral 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 (assuming they are enabled).

`-Wunused`

Warn whenever a variable is unused aside from its declaration, whenever a function is declared static but never defined, whenever a label is declared but not used, and whenever a statement computes a result that is explicitly not used. In order to get a warning about an unused function parameter, you must specify both `'-W'` and `'-Wunused'`. To suppress this warning for an expression, simply cast it to void. For unused variables, parameters and labels, use the `'unused'` attribute (see section Specifying Attributes of Variables).

`-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 nonvolatile 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`. See section Declaring Attributes of Functions.

`-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.

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.
- 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 };
```

`-Wtraditional`

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

- Macro arguments occurring within string constants in the macro body. These would substitute the argument in traditional C, but are part of the constant in ANSI C.
- 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.

`-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`

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`

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`

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.

`-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`

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.

`-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.

`-g`*`level`*

`-ggdb`*`level`*

`-gstabs`*`level`*

`-gcoff`*`level`*

`-gxcoff`*level*

`-gdwarf`*level*

`-gdwarf-2`*level*

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:

`__bb_jumps__`

Write source, target and frequency of jumps to file `'bb.out'`.

`__bb_hidecall__`

Exclude function calls from frequency count.

`__bb_showret__`

Include function returns in frequency count.

`__bb_trace__`

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'` (see section Options That Control Optimization) when recompiling, to optimize using estimated branch probabilities.

`-ftest-coverage`

Create data files for the `gcov` code-coverage utility (see section gcov: a Test Coverage Program). 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'`).

`-Q`

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

`-d`*letters*

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 word to the source file name (e.g. `'foo.c.rtl'` or `'foo.c.jump'`). Here are the possible letters for use in *letters*, and their meanings:

`'b'`

Dump after computing branch probabilities, to '*file*`.bp`'.

`'c'`

Dump after instruction combination, to the file '*file*`.combine`'.

`'d'`

Dump after delayed branch scheduling, to '*file*`.dbr`'.

`'D'`

Dump all macro definitions, at the end of preprocessing, in addition to normal output.

`'r'`

Dump after RTL generation, to '*file*`.rtl`'.

`'j'`

Dump after first jump optimization, to '*file*`.jump`'.

`'F'`

Dump after purging ADDRESSOF, to '*file*`.addressof`'.

`'f'`

Dump after flow analysis, to '*file*`.flow`'.

`'g'`

Dump after global register allocation, to '*file*`.greg`'.

`'G'`

Dump after GCSE, to '*file*`.gcse`'.

`'j'`

Dump after first jump optimization, to ‘*file*.jump’.

‘J’

Dump after last jump optimization, to ‘*file*.jump2’.

‘k’

Dump after conversion from registers to stack, to ‘*file*.stack’.

‘l’

Dump after local register allocation, to ‘*file*.lreg’.

‘L’

Dump after loop optimization, to ‘*file*.loop’.

‘M’

Dump after performing the machine dependent reorganisation pass, to ‘*file*.mach’.

‘N’

Dump after the register move pass, to ‘*file*.regmove’.

‘r’

Dump after RTL generation, to ‘*file*.rtl’.

‘R’

Dump after the second instruction scheduling pass, to ‘*file*.sched2’.

‘s’

Dump after CSE (including the jump optimization that sometimes follows CSE), to ‘*file*.cse’.

‘S’

Dump after the first instruction scheduling pass, to ‘*file*.sched’.

‘t’

Dump after the second CSE pass (including the jump optimization that sometimes follows CSE), to ‘*file*.cse2’.

‘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.

‘x’

Just generate RTL for a function instead of compiling it. Usually used with ‘r’.

‘y’

Dump debugging information during parsing, to standard error.

‘A’

Annotate the assembler output with miscellaneous debugging information.

-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 invokations with different options, in particular with and without -g.

-fdump-translation-unit-*file* (C++ only)

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

-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’.

-time

Report the CPU time taken by each subprocess in the compilation sequence. For C source files, this is the preprocessor, compiler proper, and assembler. The output looks like this:

```
# cpp 0.04 0.04
# 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 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 cpp: No such file or directory’. To resolve this you either need to put ‘cpp’ 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 '/'. See section Environment Variables Affecting GCC.

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 and function inlining. 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 ‘inline-functions’ option.

-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. See section Register Usage.

`-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 an 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 ANSI 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 ANSI 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.

`-fexpensive-optimizations`

Perform a number of minor optimizations that are relatively expensive.

`-foptimize-register-moves`

`-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-moves` 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'` (see section Options for Debugging Your Program or GCC), you can compile it a second time using `'-fbranch-probabilities'`, to improve optimizations based on guessing the path a branch might take. With `'-fbranch-probabilities'`, GCC puts a `'REG_EXEC_COUNT'` note on the first instruction of each basic block, and a `'REG_BR_PROB'` note on each `'JUMP_INSN'` and `'CALL_INSN'`. These can be used to improve optimization. Currently, they are only used in one place: in `'reorg.c'`, instead of guessing which path a branch is mostly to take, the `'REG_BR_PROB'` values are used to exactly determine which path is taken more often.

`-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;
}
```

Every language that wishes to perform language-specific alias analysis should define a function that computes, given an `tree` node, an alias set for the node. Nodes in different alias sets are not allowed to alias. For an example, see the C front-end function `c_get_alias_set`.

`-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.

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. Any ‘`-D`’ and ‘`-U`’ options on the command line are always processed before ‘`-imacros` *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.

`-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. See section Options for Directory Search, for information on '`-I`'. By using both '`-nostdinc`' and '`-I-`', you can limit the include-file search path to only those directories you specify explicitly.

`-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`

Tell the preprocessor to output a rule suitable for `make` describing the dependencies of each object file. For each source file, the preprocessor outputs one `make`-rule whose target is the object file name for that source file and whose dependencies are all the `#include` header files it uses. This rule may be a single line or may be continued with '`\`'-newline if it is long. The list of rules is printed on standard output instead of the preprocessed C program. '`-M`' implies '`-E`'. Another way to specify output of a `make` rule is by setting the environment variable `DEPENDENCIES_OUTPUT` (see section Environment Variables Affecting GCC).

`-MM`

Like '`-M`' but the output mentions only the user header 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 made by replacing ".c" with ".d" at the end of the input file names. This is in addition to compiling the file as specified---'`-MD`' does not inhibit ordinary compilation the way '`-M`' does. In 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.

`-MG`

Treat missing header files as generated files and assume they live in the same directory as the source file. If you specify '`-MG`', you must also specify either '`-M`' or '`-MM`'. '`-MG`' is not supported with '`-MD`' or '`-MMD`'.

`-H`

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

`-A`*`question`*`(`*`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.

`-D`*`macro`*

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

`-D`*`macro`*`=`*`defn`*

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

`-U`*`macro`*

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

`-dM`

Tell the preprocessor to output only a list of the macro definitions that are in effect at the end of preprocessing. 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.

`-trigraphs`

Support ANSI C trigraphs. The `'-ansi'` option also has this effect.

`-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. See section Options Controlling the Kind of Output.

`-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 `'lib`*`library`*`.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 memcmp, memset, and memcpy for System V (and ANSI 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 memcmp, memset, and memcpy for System V (and ANSI 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. (See section Interfacing to GCC Output, for more discussion of `'libgcc.a'`.) 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; see section

collect2.)

`-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. You must also specify ‘`-fpic`’ or ‘`-fPIC`’ on some systems when you specify this option.

`-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:

`-I`*dir*

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. 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 '-nostdinc' are independent.

-L*dir*

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

-B*prefix*

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 '*machine*/*version*/' (see section Specifying Target Machine and Compiler Version). 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. See section Environment Variables Affecting GCC.

-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 subprocesses and the switches to pass to them

GCC is a driver program. It performs its job by invoking a sequence of other programs to do the work of compiling, assembling and linking. GCC interprets its command-line parameters and uses these to deduce which programs it should invoke, and which command-line options it ought to place on their command lines. This behaviour is controlled by **spec strings**. In most cases there is one spec string for each program that GCC can invoke, but a few programs have multiple spec strings to control their behaviour. The spec strings built into GCC can be overridden by using the '-specs=' command-line switch to specify a spec file.

Spec files are plaintext files that are used to construct spec strings. They consist of a sequence of directives separated by blank lines. The type of directive is determined by the first non-whitespace character on the line and it can be one of the following:

%*command*

Issues a *command* to the spec file processor. The commands that can appear here are:

%include <*file*>

Search for *file* and insert its text at the current point in the specs file.

`%include_noerr <file>`

Just like '`%include`', but do not generate an error message if the include file cannot be found.

`%rename old_name new_name`

Rename the spec string *old_name* to *new_name*.

`*[spec_name]:`

This tells the compiler to create, override or delete the named spec string. All lines after this directive up to the next directive or blank line are considered to be the text for the spec string. If this results in an empty string then the spec will be deleted. (Or, if the spec did not exist, then nothing will happened.) Otherwise, if the spec does not currently exist a new spec will be created. If the spec does exist then its contents will be overridden by the text of this directive, unless the first character of that text is the '`+`' character, in which case the text will be appended to the spec.

`[suffix]:`

Creates a new '`[suffix] spec`' pair. All lines after this directive and up to the next directive or blank line are considered to make up the spec string for the indicated suffix. When the compiler encounters an input file with the named suffix, it will processes the spec string in order to work out how to compile that file. For example:

```
.ZZ:
z-compile -input %i
```

This says that any input file whose name ends in '`.ZZ`' should be passed to the program '`z-compile`', which should be invoked with the command-line switch '`-input`' and with the result of performing the '`%i`' substitution. (See below.) As an alternative to providing a spec string, the text that follows a suffix directive can be one of the following:

`@language`

This says that the suffix is an alias for a known *language*. This is similar to using the `-x` command-line switch to GCC to specify a language explicitly. For example:

```
.ZZ:
@c++
```

Says that .ZZ files are, in fact, C++ source files.

`#name`

This causes an error messages saying:

```
name compiler not installed on this system.
```

GCC already has an extensive list of suffixes built into it. This directive will add an entry to the end of the list of suffixes, but since the list is searched from the end backwards, it is effectively possible to override earlier entries using this technique.

GCC has the following spec strings built into it. Spec files can override these strings or create their own. Note that individual targets can also add their own spec strings to this list.

```
asm          Options to pass to the assembler
asm_final    Options to pass to the assembler post-processor
cpp          Options to pass to the C preprocessor
cc1          Options to pass to the C compiler
cc1plus      Options to pass to the C++ compiler
```

```
endfile      Object files to include at the end of the link
link         Options to pass to the linker
lib          Libraries to include on the command line to the linker
libgcc       Decides which GCC support library to pass to the linker
linker       Sets the name of the linker
predefines   Defines to be passed to the C preprocessor
signed_char  Defines to pass to CPP to say whether char is signed by default
startfile    Object files to include at the start of the link
```

Here is a small example of a spec file:

```
%rename lib                 old_lib

*lib:
--start-group -lgcc -lc -leval1 --end-group %(old_lib)
```

This example renames the spec called ‘`lib`’ to ‘`old_lib`’ and then overrides the previous definition of ‘`lib`’ with a new one. The new definition adds in some extra command-line options before including the text of the old definition.

Spec strings are a list of command-line options to be passed to their corresponding program. In addition, the spec strings can contain ‘`%`’-prefixed sequences to substitute variable text or to conditionally insert text into the command line. Using these constructs it is possible to generate quite complex command lines.

Here is a table of all defined ‘`%`’-sequences for spec strings. Note that spaces are not generated automatically around the results of expanding these sequences. Therefore you can concatenate them together or combine them with constant text in a single argument.

`%%`

Substitute one ‘`%`’ into the program name or argument.

`%i`

Substitute the name of the input file being processed.

`%b`

Substitute the basename of the input file being processed. This is the substring up to (and not including) the last period and not including the directory.

`%d`

Marks the argument containing or following the ‘`%d`’ as a temporary file name, so that that file will be deleted if GCC exits successfully. Unlike ‘`%g`’, this contributes no text to the argument.

`%g`*`suffix`*

Substitute a file name that has suffix *suffix* and is chosen once per compilation, and mark the argument in the same way as ‘`%d`’. To reduce exposure to denial-of-service attacks, the file name is now chosen in a way that is hard to predict even when previously chosen file names are known. For example, ‘`%g.s ... %g.o ... %g.s`’ might turn into ‘`ccUVUUAU.s ccXYAXZ12.o ccUVUUAU.s`’. *suffix* matches the regexp ‘`[.A-Za-z]*`’ or the special string ‘`%O`’, which is treated exactly as if ‘`%O`’ had been preprocessed. Previously, ‘`%g`’ was simply substituted with a file name chosen once per compilation, without regard to any appended suffix (which was therefore treated just like ordinary text), making such attacks more likely to succeed.

`%u`*`suffix`*

Like ‘`%g`’, but generates a new temporary file name even if ‘`%u`*`suffix`*’ was already seen.

%U*suffix*

Substitutes the last file name generated with '%u*suffix*', generating a new one if there is no such last file name. In the absence of any '%u*suffix*', this is just like '%g*suffix*', except they don't share the same suffix *space*, so '%g.s ... %U.s ... %g.s ... %U.s' would involve the generation of two distinct file names, one for each '%g.s' and another for each '%U.s'. Previously, '%U' was simply substituted with a file name chosen for the previous '%u', without regard to any appended suffix.

%w

Marks the argument containing or following the '%w' as the designated output file of this compilation. This puts the argument into the sequence of arguments that '%o' will substitute later.

%o

Substitutes the names of all the output files, with spaces automatically placed around them. You should write spaces around the '%o' as well or the results are undefined. '%o' is for use in the specs for running the linker. Input files whose names have no recognized suffix are not compiled at all, but they are included among the output files, so they will be linked.

%O

Substitutes the suffix for object files. Note that this is handled specially when it immediately follows '%g, %u, or %U', because of the need for those to form complete file names. The handling is such that '%O' is treated exactly as if it had already been substituted, except that '%g, %u, and %U' do not currently support additional *suffix* characters following '%O' as they would following, for example, '.o'.

%p

Substitutes the standard macro predefinitions for the current target machine. Use this when running cpp.

%P

Like '%p', but puts '__' before and after the name of each predefined macro, except for macros that start with '__' or with '_*L*', where *L* is an uppercase letter. This is for ANSI C.

%I

Substitute a '-iprefix' option made from GCC_EXEC_PREFIX.

%s

Current argument is the name of a library or startup file of some sort. Search for that file in a standard list of directories and substitute the full name found.

%e*str*

Print *str* as an error message. *str* is terminated by a newline. Use this when inconsistent options are detected.

%|

Output '-' if the input for the current command is coming from a pipe.

%(*name*)

Substitute the contents of spec string *name* at this point.

%[*name*]

Like ‘%(...)’ but put ‘__’ around ‘-D’ arguments.

%x{*option*}

Accumulate an option for ‘%X’.

%X

Output the accumulated linker options specified by ‘-Wl’ or a ‘%x’ spec string.

%Y

Output the accumulated assembler options specified by ‘-Wa’.

%Z

Output the accumulated preprocessor options specified by ‘-Wp’.

%v1

Substitute the major version number of GCC. (For version 2.9.5, this is 2.)

%v2

Substitute the minor version number of GCC. (For version 2.9.5, this is 9.)

%a

Process the asm spec. This is used to compute the switches to be passed to the assembler.

%A

Process the asm_final spec. This is a spec string for passing switches to an assembler post-processor, if such a program is needed.

%l

Process the link spec. This is the spec for computing the command line passed to the linker. Typically it will make use of the ‘%L %G %S %D and %E’ sequences.

%D

Dump out a ‘-L’ option for each directory that GCC believes might contain startup files. If the target supports multilibs then the current multilib directory will be prepended to each of these paths.

%L

Process the lib spec. This is a spec string for deciding which libraries should be included on the command line to the linker.

%G

Process the libgcc spec. This is a spec string for deciding which GCC support library should be included on the command line to the linker.

%S

Process the startfile spec. This is a spec for deciding which object files should be the first ones passed to the linker. Typically this might be a file named ‘crt0.o’.

%E

Process the endfile spec. This is a spec string that specifies the last object files that will be passed

to the linker.

%C

Process the cpp spec. This is used to construct the arguments to be passed to the C preprocessor.

%c

Process the signed_char spec. This is intended to be used to tell cpp whether a char is signed. It typically has the definition:

```
%{funsigned-char:-D__CHAR_UNSIGNED__}
```

%1

Process the cc1 spec. This is used to construct the options to be passed to the actual C compiler (‘cc1’).

%2

Process the cc1plus spec. This is used to construct the options to be passed to the actual C++ compiler (‘cc1plus’).

%*

Substitute the variable part of a matched option. See below. Note that each comma in the substituted string is replaced by a single space.

%{S}

Substitutes the -S switch, if that switch was given to GCC. If that switch was not specified, this substitutes nothing. Note that the leading dash is omitted when specifying this option, and it is automatically inserted if the substitution is performed. Thus the spec string ‘%{foo}’ would match the command-line option ‘-foo’ and would output the command line option ‘-foo’.

%W{S}

Like %{S} but mark last argument supplied within as a file to be deleted on failure.

%{S*}

Substitutes all the switches specified to GCC whose names start with -S, but which also take an argument. This is used for switches like ‘-o, -D, -I’, etc. GCC considers ‘-o foo’ as being one switch whose names starts with ‘o’. %{o*} would substitute this text, including the space. Thus two arguments would be generated.

%{^S*}

Like %{S*}, but don't put a blank between a switch and its argument. Thus %{^o*} would only generate one argument, not two.

%{S*:X}

Substitutes X if one or more switches whose names start with -S are specified to GCC. Note that the tail part of the -S option (i.e. the part matched by the ‘*’) will be substituted for each occurrence of ‘%*’ within X.

%{S:X}

Substitutes X, but only if the ‘-S’ switch was given to GCC.

%{!S:X}

Substitutes X, but only if the ‘-S’ switch was *not* given to GCC.

%{|S:X}

Like %{S:X}, but if no S switch, substitute ‘-’.

%{|!S:X}

Like %{!S:X}, but if there is an S switch, substitute ‘-’.

%{.S:X}

Substitutes X, but only if processing a file with suffix S.

%{!.S:X}

Substitutes X, but only if *not* processing a file with suffix S.

%{S|P:X}

Substitutes X if either -S or -P was given to GCC. This may be combined with ‘!’ and ‘.’ sequences as well, although they have a stronger binding than the ‘|’. For example a spec string like this:

```
%{.c:-foo} %{!.c:-bar} %{.c|d:-baz} %{!.c|d:-boggle}
```

will output the following command-line options from the following input command-line options:

```
fred.c        -foo -baz
jim.d         -bar -boggle
-d fred.c     -foo -baz -boggle
-d jim.d      -bar -baz -boggle
```

The conditional text X in a %{S:X} or %{!S:X} construct may contain other nested ‘%’ constructs or spaces, or even newlines. They are processed as usual, as described above.

The ‘-O, -f, -m, and -W’ switches are handled specifically in these constructs. If another value of ‘-O’ or the negated form of a ‘-f, -m, or -W’ switch is found later in the command line, the earlier switch value is ignored, except with {S*} where S is just one letter, which passes all matching options.

The character ‘|’ at the beginning of the predicate text is used to indicate that a command should be piped to the following command, but only if ‘-pipe’ is specified.

It is built into GCC which switches take arguments and which do not. (You might think it would be useful to generalize this to allow each compiler's spec to say which switches take arguments. But this cannot be done in a consistent fashion. GCC cannot even decide which input files have been specified without knowing which switches take arguments, and it must know which input files to compile in order to tell which compilers to run).

GCC also knows implicitly that arguments starting in ‘-l’ are to be treated as compiler output files, and passed to the linker in their proper position among the other output files.

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 and version.

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 compiler version.

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.

These options are defined by the macro `TARGET_SWITCHES` in the machine description. The default for the options is also defined by that macro, which enables you to change the defaults.

- M680x0 Options
- VAX Options
- SPARC Options
- Convex Options
- AMD29K Options
- ARM Options
- Thumb Options
- MN10200 Options
- MN10300 Options
- M32R/D Options
- M88K Options
- RS/6000 and PowerPC Options
- RT Options
- MIPS Options
- i386 Options
- HPPA Options
- Intel 960 Options
- DEC Alpha Options
- Clipper Options
- H8/300 Options
- SH Options
- System V Options
- TMS320C3x/C4x Options
- V850 Options
- ARC Options
- NS32K Options

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.

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 ‘sparclite-*-*’ 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.

`-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.

`-mcmodel=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.

`-mcmodel=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.

`-mcmodel=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.

`-mcmodel=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 systems 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 mtsrim 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 if 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.

-mshort-load-bytes

Do not try to load half-words (eg 'short's) by loading a word from an unaligned address. For some targets the MMU is configured to trap unaligned loads; use this option to generate code that is safe in these environments.

-mno-short-load-bytes

Use unaligned word loads to load half-words (eg 'short's). This option produces more efficient code, but the MMU is sometimes configured to trap these instructions.

-mshort-load-words

This is a synonym for '-mno-short-load-bytes'.

-mno-short-load-words

This is a synonym for '-mshort-load-bytes'.

-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 specifes 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.

`-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 produced 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.

`-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 `r0`, 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 `'-power'` 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.

-m64

-m32

Enable 64-bit PowerPC ABI and calling convention: 64-bit pointers, 64-bit `long` type, and the infrastructure needed to support them. Specifying ‘-m64’ implies ‘-mpowerpc64’ and ‘-mpowerpc’, while ‘-m32’ disables the 64-bit ABI and implies ‘-mno-powerpc64’. GCC defaults

to ‘-m32’.

`-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 **pthreads** 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 t