NAME

gcc, g++ - GNU project C and C++ Compiler (egcs-1.1.2)

SYNOPSIS

gcc [*option* | *filename*]...
g++ [*option* | *filename*]...

WARNING

The information in this man page is an extract from the full documentation of the GNU C compiler, and is limited to the meaning of the options.

This man page is not kept up to date except when volunteers want to maintain it. If you find a discrepancy between the man page and the software, please check the Info file, which is the authoritative documentation.

If we find that the things in this man page that are out of date cause significant confusion or complaints, we will stop distributing the man page.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

For complete and current documentation, refer to the Info file **'gcc'** or the manual *Using and Porting GNU CC (for version 2.0)*. Both are made from the Texinfo source file **gcc.texinfo**.

DESCRIPTION

The C and C++ compilers are integrated. Both process input files through one or more of four stages: preprocessing, compilation, assembly, and linking. Source filename suffixes identify the source language, but which name you use for the compiler governs default assumptions:

gcc assumes preprocessed (**.i**) files are C and assumes C style linking.

g++ assumes preprocessed (**.i**) files are C++ and assumes C++ style linking.

Suffixes of source file names indicate the language and kind of processing to be done:

.c C source; preprocess, compile, assemble
.C C++ source; preprocess, compile, assemble
.cc C++ source; preprocess, compile, assemble
.cxx C++ source; preprocess, compile, assemble
.m Objective-C source; preprocess, compile, assemble
.i preprocessed C; compile, assemble
.ii preprocessed C++; compile, assemble

.s Assembler source; assemble
.S Assembler source; preprocess, assemble
.h Preprocessor file; not usually named on command line

Files with other suffixes are passed to the linker. Common cases include:

.o Object file
.a Archive file

Linking is always the last stage unless you use one of the **-c**, **-S**, or **-E** options to avoid it (or unless compilation errors stop the whole process). For the link stage, all **.o** files corresponding to source files, **-l** libraries, unrecognized filenames (including named **.o** object files and **.a** archives) are passed to the linker in command-line order.

OPTIONS

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

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

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

Overall Options

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

Language Options

-ansi -fall-virtual -fcond-mismatch
-fdollars-in-identifiers -fenum-int-equiv
-fexternal-templates -fno-asm -fno-builtin -fhosted
-fno-hosted -ffreestanding -fno-freestanding
-fno-strict-prototype -fsigned-bitfields
-fsigned-char -fthis-is-variable
-funsigned-bitfields -funsigned-char
-fwritable-strings -traditional -traditional-cpp
-trigraphs

Warning Options

-fsyntax-only -pedantic -pedantic-errors -w -W
-Wall -Waggregate-return -Wcast-align -Wcast-qual
-Wchar-subscript -Wcomment -Wconversion
-Wenum-clash -Werror -Wformat -Wid-clash-*len*
-Wimplicit -Wimplicit-int
-Wimplicit-function-declaration -Winline
-Wlong-long -Wmain -Wmissing-prototypes
-Wmissing-declarations -Wnested-externs -Wno-import
-Wparentheses -Wpointer-arith -Wredundant-decls
-Wreturn-type -Wshadow -Wstrict-prototypes -Wswitch

```
-Wtemplate-debugging -Wtraditional -Wtrigraphs
-Wuninitialized -Wunused -Wwrite-strings
```

Debugging Options

-a -d*letters* -fpretend-float -g -g*level* -gcoff
-gxcoff -gxcoff+ -gdwarf -gdwarf+ -gstabs -gstabs+
-ggdb -p -pg -save-temps -print-file-name=*library*
-print-libgcc-file-name -print-prog-name=*program*

Optimization Options

-fcaller-saves -fcse-follow-jumps -fcse-skip-blocks
-fdelayed-branch -felide-constructors
-fexpensive-optimizations -ffast-math -ffloat-store
-fforce-addr -fforce-mem -finline-functions
-fkeep-inline-functions -fmemoize-lookups
-fno-default-inline -fno-defer-pop
-fno-function-cse -fno-inline -fno-peephole
-fomit-frame-pointer -frerun-cse-after-loop
-fschedule-insns -fschedule-insns2
-fstrength-reduce -fthread-jumps -funroll-all-loops
-funroll-loops -O -O2 -O3

Preprocessor Options

-A*assertion* -C -dD -dM -dN -D*macro*[=*defn*] -E -H
-idirafter *dir* -include *file* -imacros *file* -iprefix
file -iwithprefix *dir* -M -MD -MM -MMD -nostdinc -P
-U*macro* -undef

Assembler Option

-Wa,*option*

Linker Options

-l*library* -nostartfiles -nostdlib -static -shared
-symbolic -Xlinker *option* -Wl,*option* -u *symbol*

Directory Options

-B*prefix* -I*dir* -I- -L*dir*

Target Options

-b *machine* -V *version*

Configuration Dependent Options

M680x0 Options
-m68000 -m68020 -m68020-40 -m68030 -m68040 -m68881
-mbitfield -mc68000 -mc68020 -mfpa -mnobitfield
-mrtd -mshort -msoft-float

VAX Options
-mg -mgnu -munix

SPARC Options
-mepilogue -mfpu -mhard-float -mno-fpu
-mno-epilogue -msoft-float -msparclite -mv8
-msupersparc -mcypress

Convex Options
-margcount -mc1 -mc2 -mnoargcount

AMD29K Options
-m29000 -m29050 -mbw -mdw -mkernel-registers
-mlarge -mnbw -mnodw -msmall -mstack-check
-muser-registers

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

RS6000 Options
-mfp-in-toc -mno-fop-in-toc

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
-mcpu=*cpu type* -mips2 -mips3 -mint64 -mlong64
-mmips-as -mgas -mrnames -mno-rnames -mgpopt
-mno-gpopt -mstats -mno-stats -mmemcpy -mno-memcpy
-mno-mips-tfile -mmips-tfile -msoft-float
-mhard-float -mabicalls -mno-abicalls -mhalf-pic
-mno-half-pic -G *num* -nocpp

i386 Options
-m486 -mno-486 -msoft-float -mno-fp-ret-in-387

HPPA Options
-mpa-risc-1-0 -mpa-risc-1-1 -mkernel -mshared-libs
-mno-shared-libs -mlong-calls -mdisable-fpregs
-mdisable-indexing -mtrailing-colon

i960 Options
-m*cpu-type* -mnumerics -msoft-float
-mleaf-procedures -mno-leaf-procedures -mtail-call
-mno-tail-call -mcomplex-addr -mno-complex-addr
-mcode-align -mno-code-align -mic-compat
-mic2.0-compat -mic3.0-compat -masm-compat
-mintel-asm -mstrict-align -mno-strict-align
-mold-align -mno-old-align

DEC Alpha Options

-mfp-regs -mno-fp-regs -mno-soft-float -msoft-float

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

Code Generation Options
-fcall-saved-*reg* -fcall-used-*reg* -ffixed-*reg*
-finhibit-size-directive -fnonnull-objects
-fno-common -fno-ident -fno-gnu-linker
-fpcc-struct-return -fpic -fPIC -freg-struct-return
-fshared-data -fshort-enums -fshort-double
-fvolatile -fvolatile-global -fverbose-asm

OVERALL OPTIONS

-x *language*
Specify explicitly the *language* for the following input files (rather than choosing a default based on the file name suffix) . This option applies to all following input files until the next **'-x'** option. Possible values of *language* are **'c'**, **'objective-c'**, **'c-header'**, **'c++'**, **'cpp-output'**, **'assembler'**, and **'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 want only some of the four stages (preprocess, compile, assemble, link), 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 compiler output is an object file corresponding to each source file.

By default, GCC makes the object file name for a source file by replacing the suffix **'.c'**, **'.i'**, **'.s'**, etc., with **'.o'**. Use **-o** to select another name.

GCC ignores any unrecognized input files (those that do not require compilation or assembly) with the **-c** option.

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

By default, GCC makes the assembler file name for a source file by replacing the suffix **'.c'**, **'.i'**, etc., with **'.s'**. Use **-o** to select another name.

GCC ignores any input files that don't require compilation.

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

GCC ignores input files which don't require preprocessing.

-o *file*
Place output in file *file*. This applies regardless to whatever sort of output GCC is producing, 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 you do not specify **'-o'**, 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 cannot read from a pipe; but the GNU assembler has no trouble.

LANGUAGE OPTIONS

The following options control the dialect of C that the compiler accepts:

-ansi Support all ANSI standard C programs.

This turns off certain features of GNU C that are incompatible with ANSI C, such as the **asm**, **inline** 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, and disallows

'$' as part of identifiers.

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'**.

The preprocessor predefines a macro **__STRICT_ANSI__** when you use the **'-ansi'** option. 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.

-fno-asm
Do not recognize **asm**, **inline** or **typeof** as a keyword. These words may then be used as identifiers. You can use **__asm__**, **__inline__** and **__typeof__** instead. **'-ansi'** implies **'-fno-asm'**.

-fno-builtin
Don't recognize built-in functions that do not begin with two leading underscores. Currently, the functions affected include **_exit**, **abort**, **abs**, **alloca**, **cos**, **exit**, **fabs**, **labs**, **memcmp**, **memcpy**, **sin**, **sqrt**, **strcmp**, **strcpy**, and **strlen**.

The **'-ansi'** option prevents **alloca** and **_exit** from being builtin functions.

-fhosted
Compile for a hosted environment; this implies the **'-fbuiltin'** option, and implies that suspicious declarations of **main** should be warned about.

-ffreestanding
Compile for a freestanding environment; this implies the **'-fno-builtin'** option, and implies that **main** has no special requirements.

-fno-strict-prototype
Treat a function declaration with no arguments, such as **'int foo ();'**, as C would treat it--as saying nothing about the number of arguments or their types (C++ only). Normally, such a declaration in

C++ means that the function **foo** takes no arguments.

-trigraphs
Support ANSI C trigraphs. The **'-ansi'** option implies **'-trigraphs'**.

-traditional
Attempt to support some aspects of traditional C compilers. For details, see the GNU C Manual; the duplicate list here has been deleted so that we won't get complaints when it is out of date.

But one note about C++ programs only (not C). **'-traditional'** has one additional effect for C++: assignment to **this** is permitted. This is the same as the effect of **'-fthis-is-variable'**.

-traditional-cpp
Attempt to support some aspects of traditional C preprocessors. This includes the items that specifically mention the preprocessor above, but none of the other effects of **'-traditional'**.

-fdollars-in-identifiers
Permit the use of **'$'** in identifiers (C++ only). You can also use **'-fno-dollars-in-identifiers'** to explicitly prohibit use of **'$'**. (GNU C++ allows **'$'** by default on some target systems but not others.)

-fenum-int-equiv
Permit implicit conversion of **int** to enumeration types (C++ only). Normally GNU C++ allows conversion of **enum** to **int**, but not the other way around.

-fexternal-templates
Produce smaller code for template declarations, by generating only a single copy of each template function where it is defined (C++ only). To use this option successfully, you must also mark all files that use templates with either **'#pragma implementation'** (the definition) or **'#pragma interface'** (declarations).

When your code is compiled with **'-fexternal-templates'**, all template instantiations are external. You must arrange for all necessary instantiations to appear in the implementation file; you can do this with a **typedef** that references each instantiation needed. Conversely, when you compile using the default option **'-fno-external-templates'**, all template instantiations are explicitly internal.

-fall-virtual
Treat all possible member functions as virtual, implicitly. All member functions (except for constructor functions and **new** or **delete** member operators) are treated as virtual functions of the class where they appear.

This does not mean that all calls to these member functions will be made through the internal table of virtual functions. Under some circumstances, the compiler can determine that a call to a given virtual function can be made directly; in these cases the calls are direct in any case.

-fcond-mismatch
Allow conditional expressions with mismatched types in the second and third arguments. The value of such an expression is void.

-fthis-is-variable
Permit assignment to **this** (C++ only). The incorporation of user-defined free store management into C++ has made assignment to '**this**' an anachronism. Therefore, by default it is invalid to assign to **this** within a class member function. However, for backwards compatibility, you can make it valid with '**-fthis-is-variable**'.

-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** and **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, '**-fno-signed-char**' is

equivalent to '**-funsigned-char**'.

-fsigned-bitfields

-funsigned-bitfields

-fno-signed-bitfields

-fno-unsigned-bitfields
These options control whether a bitfield is signed or unsigned, when declared with no explicit '**signed**' or '**unsigned**' qualifier. By default, such a bitfield is signed, because this is consistent: the basic integer types such as **int** are signed types.

However, when you specify '**-traditional**', 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. '**-traditional**' also has this effect.

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

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

If you use the '**-E**' option, GCC does nothing 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. The preprocessor evaluates any **'-D'** and **'-U'** options on the command line before processing **'-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'**.

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

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

-nostdinc++
Do not search for header files in the C++-specific standard directories, but do still search the other standard directories. (This option is used when building **'libg++'**.)

-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'** commands. Used with the **'-E'** option.

-M [-MG]
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 files **'#include'**d in it. 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'**.

'-MG' says to treat missing header files as generated files and assume they live in the same directory as the source file. It must be specified in addition to **'-M'**.

-MM [-MG]
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 files with names made by replacing **'.o'** with **'.d'** at the end of the output file names. This is in addition to compiling the file as specified--**'-MD'** does not inhibit ordinary compilation the way **'-M'** does.

The Mach utility **'md'** can be used to merge the **'.d'** 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.

-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 preprocessor conditional such as **'#if #**_question_**(**_answer_**)'**. **'-A-'** disables the standard assertions that normally describe the target machine.

-A_question_
(_answer_) Assert the answer _answer_ for _question_, in case it is tested with a preprocessor 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 preprocessor 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.

ASSEMBLER OPTION

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

LINKER OPTIONS

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 GCC does a link step, these object files are used as input to the linker.

-l*library*
Use the library named *library* when linking.

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. However, if the linker finds an ordinary object file rather than a library, the object file 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 libraries are used normally.

-nostdlib
Don't use the standard system libraries and startup files when linking. Only the files you specify will be passed to the linker.

-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. Only a few systems support 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 GNU CC 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 **'-Xlink-**

er -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.

DIRECTORY OPTIONS

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

-I*dir* Append directory *dir* to the list of directories searched for include files.

-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 and data files of the compiler itself.

The compiler driver program runs one or more of the subprograms **'cpp'**, **'cc1'** (or, for C++, **'cc1plus'**), **'as'** and **'ld'**. It tries *prefix* as a prefix for each program it tries to run, both with and without **'***machine***/***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 compiler driver searches for the unmodified program name, using the directories specified in your **'PATH'** environment variable.

The run-time support file **'libgcc.a'** is also 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. Most of the time, on most machines, **'libgcc.a'** is not actually necessary.

You can get a similar result from the environment variable **GCC_EXEC_PREFIX**; if it is defined, its value is used as a prefix in the same way. If both the **'-B'** option and the **GCC_EXEC_PREFIX** variable are present, the **'-B'** option is used first and the environment variable value second.

WARNING OPTIONS

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

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

-fsyntax-only
Check the code for syntax errors, but don't emit any output.

-w Inhibit all warning messages.

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

-pedantic
Issue all the warnings demanded by strict ANSI standard C; reject all programs that use forbidden extensions.

Valid ANSI standard 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 features are supported as well. With this option, they are rejected. There is no reason to *use* this option; it exists only to satisfy pedants.

'-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.

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

-W Print extra warning messages for these events:

- A nonvolatile automatic variable might be changed by a call to **longjmp**. These warnings 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.

- 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;
}
```

  Spurious warnings can occur because GNU CC does not realize that certain functions (including **abort** and **longjmp**) will never return.

- 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 **'<='**.

-Wimplicit-int
Warn whenever a declaration does not specify a type.

-Wimplicit-function-declaration
Warn whenever a function is used before being declared.

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

-Wmain Warn if the **main** function is declared or defined with a suspicious type. Typically, it is a function with external linkage, returning **int**, and taking zero or two arguments.

-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**.

-Wunused
Warn whenever a local variable is unused aside from its declaration, whenever a function is declared static but never defined, and whenever a statement computes a result that is explicitly not used.

-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.

-Wcomment
Warn whenever a comment-start sequence **'/*'** appears in a comment.

-Wtrigraphs
Warn if any trigraphs are encountered (assuming they are enabled).

-Wformat
Check calls to **printf** and **scanf**, etc., to make sure that the arguments supplied have types appropriate to the format string specified.

-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.

-Wuninitialized
An automatic variable is used without first being initialized.

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 GNU CC 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 GNU CC 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.

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

-Wparentheses
Warn if parentheses are omitted in certain contexts.

-Wtemplate-debugging
When using templates in a C++ program, warn if debugging is not yet fully available (C++ only).

-Wall All of the above **'-W'** options combined. These are all the options which pertain to usage that we recommend avoiding and that we believe is easy to avoid, even in conjunction with macros.

The remaining **'-W...'** options are not implied by **'-Wall'** because they warn about constructions that we consider reasonable to use, on occasion, in clean programs.

-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**.

-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.

-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.

-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.

-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.

-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 an function.

-Wenum-clash
Warn about conversion between different enumeration types (C++ only).

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

-Woverloaded-virtual
(C++ only.) In a derived class, the definitions of virtual functions must match the type signature of a virtual function declared in the base class. Use this option to request warnings when a derived class declares a function that may be an erroneous attempt to define a virtual function: that is, warn when a function with the same name as a virtual function in the base class, but with a type signature that doesn't match any virtual functions from the base class.

-Winline
Warn if a function can not be inlined, and either it was declared as inline, or else the **-finline-functions** option was given.

-Werror
Treat warnings as errors; abort compilation after any warning.

DEBUGGING OPTIONS

GNU CC 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+'**, or **'-gdwarf'** (see below).

Unlike most other C compilers, GNU CC 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 GNU CC is generated with the capability for more than one debugging format.

-ggdb Produce debugging information in the native format (if that is 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.

-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.

-gdwarf
Produce debugging information in DWARF 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 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.

-g*level*
-ggdb*level*
-gstabs*level*
-gcoff*level* **-gxcoff***level*

-gdwarf*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**.

-pg Generate extra code to write profile information suitable for the analysis program **gprof**.

-a Generate extra code to write profile information for basic blocks, which will record the number of times each basic block is executed. 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.

-ax Generate extra code to read basic block profiling parameters from file ‘bb.in’ and write profiling results to file ‘bb.out’. ‘bb.in’ contains a list of functions. Whenever a function on the list is entered, profiling is turned on. When the outmost function is left, profiling is turned off. If a function name is prefixed with ‘-’ the function is excluded from profiling. If a function name is not unique it can be disambiguated by writing ‘/path/filename.d:functionname’. ‘bb.out’ will list some available filenames. Four function names have a special meaning: ‘__bb_jumps__’ will cause jump frequencies to be written to ‘bb.out’. ‘__bb_trace__’ will cause the sequence of basic blocks to be piped into ‘gzip’ and written to file ‘bbtrace.gz’. ‘__bb_hidecall__’ will cause call instructions to be excluded from the trace. ‘__bb_showret__’ will cause return instructions to be included in the trace.

-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’**).

-dM Dump all macro definitions, at the end of preprocessing, and write no output.

-dN Dump all macro names, at the end of preprocessing.

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

-dy Dump debugging information during parsing, to standard error.

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

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

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

-ds Dump after CSE (including the jump optimization

that sometimes follows CSE), to ‘*file*.**cse**’.

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

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

-df Dump after flow analysis, to ‘*file*.**flow**’.

-dc Dump after instruction combination, to ‘*file*.**combine**’.

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

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

-dg Dump after global register allocation, to ‘*file*.**greg**’.

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

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

-dd Dump after delayed branch scheduling, to ‘*file*.**dbr**’.

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

-da Produce all the dumps listed above.

-dm Print statistics on memory usage, at the end of the run, to standard error.

-dp Annotate the assembler output with a comment indicating which pattern and alternative was used.

-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 GNU CC 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, compil-

ing **'foo.c'** with **'-c -save-temps'** would produce files **'foo.cpp'** and **'foo.s'**, as well as **'foo.o'**.

-print-file-name=*library*
Print the full absolute name of the library file *library* that would be used when linking--and do not do anything else. With this option, GNU CC does not compile or link anything; it just prints the file name.

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

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

OPTIMIZATION OPTIONS

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'**, only variables declared **register** are allocated 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 two options **'-fthread-jumps'** and **'-fdefer-pop'** are turned on. On machines that have delay slots, the **'-fdelayed-branch'** option is turned on. For those machines that can support debugging even without a frame pointer, the **'-fomit-frame-pointer'** option is turned on. On some machines other flags may also be turned on.

-O2 Optimize even more. Nearly all supported optimizations that do not involve a space-speed tradeoff

are performed. Loop unrolling and function inlining are not done, for example. As compared to **-O**, this option increases both compilation time and the performance of the generated code.

-O3 Optimize yet more. This turns on everything **-O2** does, along with also turning on **-finline-functions**.

-O0 Do not optimize.

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'**. The following list shows only one form--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. This 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.

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.

-fmemoize-lookups

-fsave-memoized
Use heuristics to compile faster (C++ only). These heuristics are not enabled by default, since they are only effective for certain input files. Other input files compile more slowly.

The first time the compiler must build a call to a member function (or reference to a data member), it must (1) determine whether the class implements member functions of that name; (2) resolve which member function to call (which involves figuring out what sorts of type conversions need to be made); and (3) check the visibility of the member function to the caller. All of this adds up to slower compilation. Normally, the second time a call is made to that member function (or reference to that data member), it must go through the same lengthy process again. This means that code like this

```
cout << "This " << p << " has " << n << " legs.\n";
```

makes six passes through all three steps. By using a software cache, a "hit" significantly reduces this cost. Unfortunately, using the cache introduces another layer of mechanisms which must be implemented, and so incurs its own overhead. **'-fmemoize-lookups'** enables the software cache.

Because access privileges (visibility) to members and member functions may differ from one function context to the next, **g++** may need to flush the cache. With the **'-fmemoize-lookups'** flag, the cache is flushed after every function that is compiled. The '-fsave-memoized' flag enables the same software cache, but when the compiler determines that the context of the last function compiled would yield the same access privileges of the next function to compile, it preserves the cache. This is most helpful when defining many member functions for the same class: with the exception of member functions which are friends of other classes, each member function has exactly the same access privileges as every other, and the cache need not be flushed.

-fno-default-inline
Don't make member functions inline by default merely because they are defined inside the class scope (C++ only).

-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 may produce better code by making all memory references potential common subexpressions. When they are not common subexpressions, instruction combination should eliminate the separate register-load. I am interested in hearing about the difference this makes.

-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. I am interested in hearing about the difference this makes.

-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 most 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.

-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 GCC normally does not output the function as assembler code in its own right.

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

-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.

-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.

-fno-peephole
Disable any machine-specific peephole optimizations.

-ffast-math
This option allows GCC to violate some ANSI or IEEE rules/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.

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.

The following options control specific optimizations. The **'-O2'** option turns on all of these optimizations except **'-funroll-loops'** and **'-funroll-all-loops'**.

The **'-O'** option usually turns on the **'-fthread-jumps'** and **'-fdelayed-branch'** options, but specific machines may change the default optimizations.

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.

-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-all-loops
Perform the optimization of loop unrolling. This is done for all loops. This usually makes programs run more slowly.

-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.

-felide-constructors
Elide constructors when this seems plausible (C++ only). With this flag, GNU C++ initializes **y** directly from the call to **foo** without going through a temporary in the following code:

```
A foo (); A y = foo ();
```

Without this option, GNU C++ first initializes **y** by calling the appropriate constructor for type **A;** then assigns the result of **foo** to a temporary; and, finally, replaces the initial value of **'y'** with the temporary.

The default behavior (**'-fno-elide-constructors'**) is specified by the draft ANSI C++ standard. If your program's constructors have side effects, using **'-felide-constructors'** can make your program act differently, since some constructor calls may be omitted.

-fexpensive-optimizations
Perform a number of minor optimizations that are relatively expensive.

-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.

TARGET OPTIONS

By default, GNU CC 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 GNU CC, 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 GNU CC 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 GNU CC as a cross-compiler.

The value to use for *machine* is the same as was specified as the machine type when configuring GNU CC 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 GNU CC to run. This is useful when multiple versions are installed. For example, *version* might be **'2.0'**, meaning to run GNU CC version 2.0.

The default version, when you do not specify **'-V'**, is controlled by the way GNU CC is installed. Normally, it will be a version that is recommended for general use.

MACHINE DEPENDENT OPTIONS

Each of the 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 command-line compatibility with other compilers on the same platform.

These are the **'-m'** options defined for the 68000 series:

-m68000

-mc68000
Generate output for a 68000. This is the default when the compiler is configured for 68000-based systems.

-m68020

-mc68020
Generate output for a 68020 (rather than a 68000). 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-based 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.

-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.

-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 part of GNU CC. 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.

-mshort
Consider type **int** to be 16 bits wide, like **short int**.

-mnobitfield
Do not use the bit-field instructions. **'-m68000'** implies **'-mnobitfield'**.

-mbitfield
Do use the bit-field instructions. **'-m68020'** implies **'-mbitfield'**. This is the default if you use the unmodified sources.

-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 and 68020 processors, but not by the 68000.

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.

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

-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:* there is no GNU floating-point library for SPARC. 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.

-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-v8

-mv8

-msparclite
These three 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.

-mcypress

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

With **-mcypress** (the default), the compiler optimises 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 optimises 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 **'-m'** options are defined for the Convex:

-mc1 Generate output for a C1. This is the default when the compiler is configured for a C1.

-mc2 Generate output for a C2. This is the default when the compiler is configured for a C2.

-margcount
Generate code which puts an argument count in the word preceding each argument list. Some non-portable Convex and Vax programs need this word. (Debuggers don't, except for functions with variable-length argument lists; this info is in the symbol table.)

-mnoargcount
Omit the argument count word. This is the default if you use the unmodified sources.

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.

-mnodw 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 de-

fault.

-mnbw Generate code that assumes the systems does not support byte and halfword write operations. This implies **'-mnodw'**.

-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.

-mlarge
Do not assume that the **call** instruction can be used; this is the default.

-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 **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
Insert a call to **__msp_check** after each stack adjustment. This is often used for kernel code.

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.

-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.

-mno-check-zero-division

-mcheck-zero-division
Early models of the 88K architecture had problems with division by zero; in particular, many of them didn't trap. Use these options to avoid including (or to include explicitly) additional code to detect division by zero and signal an exception. All GCC configurations for the 88K use **'-mcheck-zero-division'** by default.

-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 is not needed by GDB. 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

-mno-ocs-frame-position
Force (or do not require) register values to be stored in a particular place in stack frames, as specified in OCS. The DG/UX, Delta88 SVr3.2, and BCS configurations use **'-mocs-frame-position'**; other 88k configurations have the default **'-mno-ocs-frame-position'**.

-moptimize-arg-area

-mno-optimize-arg-area
Control how to store function arguments in stack frames. **'-moptimize-arg-area'** saves space, but may break some debuggers (not GDB). **'-mno-optimize-arg-area'** conforms better to stan-

dards. By default GCC does not optimize the argument area.

-mshort-data-*num*
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 do not, generate code to guarantee sequential consistency of volatile memory references.

GNU CC always guarantees consistency by default, for the preferred processor submodel. How this is done depends on the submodel.

The m88100 processor does not reorder memory references and so always provides sequential consistency. If you use **'-m88100'**, GNU CC does not generate any special instructions for sequential consistency.

The order of memory references made by the m88110 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 you use **'-m88000'** or **'-m88110'**, GNU CC generates special instructions when appropriate, to force execution in the proper order.

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 the option **'-mno-serialize-volatile'**.

If you use the **'-m88100'** option but require sequential consistency when running on the m88110 processor, you should use **'-mserialize-volatile'**.

-msvr4

-msvr3 Turn on (**'-msvr4'**) or off (**'-msvr3'**) compiler extensions related to System V release 4 (SVr4).

This controls the following:

- Which variant of the assembler syntax to emit (which you can select independently using **'-mversion-03.00'**).
- **'-msvr4'** makes the C preprocessor recognize **'#pragma weak'**
- **'-msvr4'** makes GCC issue additional declaration directives used in SVr4.

'-msvr3' is the default for all m88K configurations except the SVr4 configuration.

-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.

-muse-div-instruction
Very early models of the 88K architecture didn't have a divide instruction, so GCC avoids that instruction by default. Use this option to specify that it's safe to use the divide instruction.

-mversion-03.00
In the DG/UX configuration, there are two flavors of SVr4. This option modifies **-msvr4** to select whether the hybrid-COFF or real-ELF flavor is used. All other configurations ignore this option.

-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.

These options are defined for the IBM RS6000:

-mfp-in-toc

-mno-fp-in-toc
Control whether or not floating-point constants go in the Table of Contents (TOC), a table of all global variable and function addresses. By default GCC puts floating-point constants there; if the TOC overflows, **'-mno-fp-in-toc'** will reduce the size of the TOC, which may avoid the overflow.

These **'-m'** options are defined for the IBM RT PC:

-min-line-mul
Use an in-line code sequence for integer multiplies. This is the default.

-mcall-lib-mul
Call **lmul$$** for integer multiples.

-mfull-fp-blocks
Generate full-size floating point data blocks, including the minimum amount of scratch space recommended by IBM. This is the default.

-mminimum-fp-blocks
Do not include extra scratch space in floating point data blocks. This results in smaller code, but slower execution, since scratch space must be allocated dynamically.

-mfp-arg-in-fpregs
Use a calling sequence incompatible with the IBM calling convention in which floating point arguments are passed in floating point registers. Note that **varargs.h** and **stdargs.h** will not work with floating point operands if this option is specified.

-mfp-arg-in-gregs
Use the normal calling convention for floating point arguments. This is the default.

-mhc-struct-return
Return structures of more than one word in memory, rather than in a register. This provides compatibility with the MetaWare HighC (hc) compiler. Use **'-fpcc-struct-return'** for compatibility with the Portable C Compiler (pcc).

-mnohc-struct-return
Return some structures of more than one word in registers, when convenient. This is the default. For compatibility with the IBM-supplied compilers, use either **'-fpcc-struct-return'** or **'-mhc-struct-return'**.

These **'-m'** options are defined for the MIPS family of computers:

-mcpu=*cpu-type*
Assume the defaults for the machine type *cpu-type* when scheduling instructions. The default *cpu-type* is **default**, which picks the longest cycles times for any of the machines, in order that the code run

at reasonable rates on all MIPS cpu's. Other choices for *cpu-type* are **r2000**, **r3000**, **r4000**, and **r6000**. While picking a specific *cpu-type* will schedule things appropriately for that particular chip, the compiler will not generate any code that does not meet level 1 of the MIPS ISA (instruction set architecture) without the **-mips2** or **-mips3** switches being used.

-mips2 Issue instructions from level 2 of the MIPS ISA (branch likely, square root instructions). The **-mcpu=r4000** or **-mcpu=r6000** switch must be used in conjunction with **-mips2**.

-mips3 Issue instructions from level 3 of the MIPS ISA (64 bit instructions). The **-mcpu=r4000** switch must be used in conjunction with **-mips2**.

-mint64

-mlong64

-mlonglong128
These options don't work at present.

-mmips-as
Generate code for the MIPS assembler, and invoke **mips-tfile** to add normal debug information. This is the default for all platforms except for the OSF/1 reference platform, using the OSF/rose object format. If any of the **-ggdb**, **-gstabs**, or **-gstabs+** switches are used, the **mips-tfile** program will encapsulate the stabs within MIPS ECOFF.

-mgas Generate code for the GNU assembler. This is the default on the OSF/1 reference platform, using the OSF/rose object format.

-mrnames

-mno-rnames
The **-mrnames** switch says to output code using the MIPS software names for the registers, instead of the hardware names (ie, **a0** instead of **$4**). The GNU assembler does not support the **-mrnames** switch, and the MIPS assembler will be instructed to run the MIPS C preprocessor over the source file. The **-mno-rnames** switch is default.

-mgpopt

-mno-gpopt
The **-mgpopt** switch says to write all of the data declarations before the instructions in the text

section, to all the MIPS assembler to generate one word memory references instead of using two words for short global or static data items. This is on by default if optimization is selected.

-mstats

-mno-stats
For each non-inline function processed, the **-mstats** switch causes the compiler to emit one line to the standard error file to print statistics about the program (number of registers saved, stack size, etc.).

-mmemcpy

-mno-memcpy
The **-mmemcpy** switch makes all block moves call the appropriate string function (**memcpy** or **bcopy**) instead of possibly generating inline code.

-mmips-tfile

-mno-mips-tfile
The **-mno-mips-tfile** switch causes the compiler not postprocess the object file with the **mips-tfile** program, after the MIPS assembler has generated it to add debug support. If **mips-tfile** is not run, then no local variables will be available to the debugger. In addition, **stage2** and **stage3** objects will have the temporary file names passed to the assembler embedded in the object file, which means the objects will not compare the same.

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

-mhard-float
Generate output containing floating point instructions. This is the default if you use the unmodified sources.

-mfp64 Assume that the **FR** bit in the status word is on, and that there are 32 64-bit floating point registers, instead of 32 32-bit floating point registers. You must also specify the **-mcpu=r4000** and **-mips3** switches.

-mfp32 Assume that there are 32 32-bit floating point registers. This is the default.

-mabicalls

-mno-abicalls
Emit (or do not emit) the **.abicalls**, **.cpload**, and **.cprestore** pseudo operations that some System V.4 ports use for position independent code.

-mhalf-pic

-mno-half-pic
The **-mhalf-pic** switch says to put pointers to extern references into the data section and load them up, rather than put the references in the text section. This option does not work at present. **-G***num* Put global and static items less than or equal to *num* bytes into the small data or bss sections instead of the normal data or bss section. This allows the assembler to emit one word memory reference instructions based on the global pointer (**gp** or **$28**), instead of the normal two words used. By default, *num* is 8 when the MIPS assembler is used, and 0 when the GNU assembler is used. The **-G***num* switch is also passed to the assembler and linker. All modules should be compiled with the same **-G***num* value.

-nocpp Tell the MIPS assembler to not run its preprocessor over user assembler files (with a '**.s**' suffix) when assembling them.

These '**-m**' options are defined for the Intel 80386 family of computers: **-m486**

-mno-486
Control whether or not code is optimized for a 486 instead of an 386. Code generated for a 486 will run on a 386 and vice versa.

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

On machines where a function returns floating point results in the 80387 register stack, some floating point opcodes may be emitted even if '**-msoft-float**' is used.

-mno-fp-ret-in-387
Do not use the FPU registers for return values of functions.

The usual calling convention has functions return values of types **float** and **double** in an FPU register, even if there is no FPU. The idea is that the operating system should emulate an FPU.

The option '**-mno-fp-ret-in-387**' causes such values to be returned in ordinary CPU registers instead.

These '**-m**' options are defined for the HPPA family of computers:

-mpa-risc-1-0
Generate code for a PA 1.0 processor.

-mpa-risc-1-1
Generate code for a PA 1.1 processor.

-mkernel
Generate code which is suitable for use in kernels. Specifically, avoid **add** instructions in which one of the arguments is the DP register; generate **addil** instructions instead. This avoids a rather serious bug in the HP-UX linker.

-mshared-libs
Generate code that can be linked against HP-UX shared libraries. This option is not fully function yet, and is not on by default for any PA target. Using this option can cause incorrect code to be generated by the compiler.

-mno-shared-libs
Don't generate code that will be linked against shared libraries. This is the default for all PA targets.

-mlong-calls
Generate code which allows calls to functions greater than 256K away from the caller when the caller and callee are in the same source file. Do not turn this option on unless code refuses to link with "branch out of range errors from the linker.

-mdisable-fpregs
Prevent floating point registers from being used in any manner. This is necessary for compiling kernels which perform lazy context switching of floating point registers. If you use this option and attempt to perform floating point operations, the compiler will abort.

-mdisable-indexing
Prevent the compiler from using indexing address modes. This avoids some rather obscure problems when compiling MIG generated code under MACH.

-mtrailing-colon
Add a colon to the end of label definitions (for ELF assemblers).

These **'-m'** options are defined for the Intel 80960 family of computers:

-m*cpu-type*
Assume the defaults for the machine type *cpu-type* for instruction and addressing-mode availability and alignment. The default *cpu-type* is **kb;** other choices are **ka, mc, ca, cf, sa,** and **sb.**

-mnumerics

-msoft-float
The **-mnumerics** option indicates that the processor does support floating-point instructions. The **-msoft-float** option indicates that floating-point support should not be assumed.

-mleaf-procedures

-mno-leaf-procedures
Do (or do not) attempt to alter leaf procedures to be callable with the *bal* instruction as well as *call*. This will result in more efficient code for explicit calls when the *bal* instruction can be substituted by the assembler or linker, but less efficient code in other cases, such as calls via function pointers, or using a linker that doesn't support this optimization.

-mtail-call

-mno-tail-call
Do (or do not) make additional attempts (beyond those of the machine-independent portions of the compiler) to optimize tail-recursive calls into branches. You may not want to do this because the detection of cases where this is not valid is not totally complete. The default is **-mno-tail-call.**

-mcomplex-addr

-mno-complex-addr
Assume (or do not assume) that the use of a complex addressing mode is a win on this implementation of the i960. Complex addressing modes may not be

worthwhile on the K-series, but they definitely are on the C-series. The default is currently **-mcomplex-addr** for all processors except the CB and CC.

-mcode-align

-mno-code-align
Align code to 8-byte boundaries for faster fetching (or don't bother). Currently turned on by default for C-series implementations only.

-mic-compat

-mic2.0-compat

-mic3.0-compat
Enable compatibility with iC960 v2.0 or v3.0.

-masm-compat

-mintel-asm
Enable compatibility with the iC960 assembler.

-mstrict-align

-mno-strict-align
Do not permit (do permit) unaligned accesses.

-mold-align
Enable structure-alignment compatibility with Intel's gcc release version 1.3 (based on gcc 1.37). Currently this is buggy in that **#pragma align 1** is always assumed as well, and cannot be turned off.

These **'-m'** options are defined for the DEC Alpha implementations:

-mno-soft-float

-msoft-float
Use (do not use) the hardware floating-point instructions for floating-point operations. When **-msoft-float** is specified, functions in **'libgcc1.c'** will be used to perform floating-point operations. Unless they are replaced by routines that emulate the floating-point operations, or compiled in such a way as to call such emulations routines, these routines will issue floating-point operations. If you are compiling for an Alpha without floating-point operations, you must ensure that the library is built so as not to call them.

Note that Alpha implementations without floating-point operations are required to have floating-point registers.

-mfp-reg

-mno-fp-regs
Generate code that uses (does not use) the floating-point register set. **-mno-fp-regs** implies **-msoft-float.** If the floating-point register set is not used, floating point operands are passed in integer registers as if they were integers and floating-point results are passed in $0 instead of $f0. This is a non-standard calling sequence, so any function with a floating-point argument or return value called by code compiled with **-mno-fp-regs** must also be compiled with that option.

A typical use of this option is building a kernel that does not use, and hence need not save and restore, any floating-point registers.

These additional options are available on System V Release 4 for compatibility with other compilers on those systems:

-G On SVr4 systems, **gcc** accepts the option **'-G'** (and passes it to the system linker), for compatibility with other compilers. However, we suggest you use **'-symbolic'** or **'-shared'** as appropriate, instead of supplying linker options on the **gcc** command line.

-Qy Identify the versions of each tool used by the compiler, in a **.ident** assembler directive in the output.

-Qn Refrain from adding **.ident** directives to the output file (this is the default).

-YP,*dirs*
Search the directories *dirs*, and no others, for libraries specified with **'-l'**. You can separate directory entries in *dirs* from one another with colons.

-Ym,*dir*
Look in the directory *dir* to find the M4 preprocessor. The assembler uses this option.

CODE GENERATION OPTIONS
These machine-independent options control the interface conventions used in code generation.

Most of them begin with '-f'. These options 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.

-fnonnull-objects
Assume that objects reached through references are not null (C++ only).

Normally, GNU C++ makes conservative assumptions about objects reached through references. For example, the compiler must check that **a** is not null in code like the following:

```
obj &a = g (); a.f (2);
```

Checking that references of this sort have non-null values requires extra code, however, and it is unnecessary for many programs. You can use **'-fnonnull-objects'** to omit the checks for null, if your program doesn't require checking.

-fpcc-struct-return
Use the same convention for returning **struct** and **union** values that is used by the usual C compiler on your system. This convention is less efficient for small structures, and on many machines it fails to be reentrant; but it has the advantage of allowing intercallability between GCC-compiled code and PCC-compiled code.

-freg-struct-return
Use the convention that **struct** and **union** values are returned in registers when possible. This is more efficient for small structures than **-fpcc-struct-return.**

If you specify neither **-fpcc-struct-return** nor **-freg-struct-return,** GNU CC defaults to whichever convention is standard for the target. If there is no standard convention, GNU CC defaults to **-fpcc-struct-return.**

-fshort-enums
Allocate to an **enum** type only as many bytes as it needs for the declared range of possible values. Specifically, the **enum** type will be equivalent to the smallest integer type which has enough room.

-fshort-double
Use the same size for **double** as for **float** .

-fshared-data
Requests that the data and non-**const** variables of this compilation be shared data rather than private data. The distinction makes sense only on certain

operating systems, where shared data is shared between processes running the same program, while private data exists in one copy per process.

-fno-common
Allocate even uninitialized global variables in the bss section of the object file, rather than generating them as common blocks. This has the effect that if the same variable is declared (without **extern**) in two different compilations, you will get an error when you link them. The only reason this might be useful is if you wish to verify that the program will work on other systems which always work this way.

-fno-ident
Ignore the '**#ident**' directive.

-fno-gnu-linker
Do not output global initializations (such as C++ constructors and destructors) in the form used by the GNU linker (on systems where the GNU linker is the standard method of handling them). Use this option when you want to use a non-GNU linker, which also requires using the **collect2** program to make sure the system linker includes constructors and destructors. (**collect2** is included in the GNU CC distribution.) For systems which *must* use **collect2**, the compiler driver **gcc** is configured to do this automatically.

-finhibit-size-directive
Don't output a **.size** assembler directive, or anything else that would cause trouble if the function is split in the middle, and the two halves are placed at locations far apart in memory. This option is used when compiling '**crtstuff.c**'; you should not need to use it for anything else.

-fverbose-asm
Put extra commentary information in the generated assembly code to make it more readable. This option is generally only of use to those who actually need to read the generated assembly code (perhaps while debugging the compiler itself).

-fvolatile
Consider all memory references through pointers to be volatile.

-fvolatile-global
Consider all memory references to extern and global data items to be volatile.

-fpic If supported for the target machines, generate position-independent code, suitable for use in a shared library.

-fPIC If supported for the target machine, emit position-independent code, suitable for dynamic linking, even if branches need large displacements.

-ffixed-*reg*
Treat the register named *reg* as a fixed register; generated code should never refer to it (except perhaps as a stack pointer, frame pointer or in some other fixed role).

reg must be the name of a register. The register names accepted are machine-specific and are defined in the **REGISTER_NAMES** macro in the machine description macro file.

This flag does not have a negative form, because it specifies a three-way choice.

-fcall-used-*reg*
Treat the register named *reg* as an allocable register that is clobbered by function calls. It may be allocated for temporaries or variables that do not live across a call. Functions compiled this way will not save and restore the register *reg*.

Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

This flag does not have a negative form, because it specifies a three-way choice.

-fcall-saved-*reg*
Treat the register named *reg* as an allocable register saved by functions. It may be allocated even for temporaries or variables that live across a call. Functions compiled this way will save and restore the register *reg* if they use it.

Use of this flag for a register that has a fixed pervasive role in the machine's execution model, such as the stack pointer or frame pointer, will produce disastrous results.

A different sort of disaster will result from the use of this flag for a register in which function values may be returned.

This flag does not have a negative form, because it specifies a three-way choice.

PRAGMAS

Two **'#pragma'** directives are supported for GNU C++, to permit using the same header file for two purposes: as a definition of interfaces to a given object class, and as the full definition of the contents of that object class.

#pragma interface

(C++ only.) Use this directive in header files that define object classes, to save space in most of the object files that use those classes. Normally, local copies of certain information (backup copies of inline member functions, debugging information, and the internal tables that implement virtual functions) must be kept in each object file that includes class definitions. You can use this pragma to avoid such duplication. When a header file containing **'#pragma interface'** is included in a compilation, this auxiliary information will not be generated (unless the main input source file itself uses **'#pragma implementation'**). Instead, the object files will contain references to be resolved at link time.

#pragma implementation

#pragma implementation "*objects***.h"**

(C++ only.) Use this pragma in a main input file, when you want full output from included header files to be generated (and made globally visible). The included header file, in turn, should use **'#pragma interface'**. Backup copies of inline member functions, debugging information, and the internal tables used to implement virtual functions are all generated in implementation files.

If you use **'#pragma implementation'** with no argument, it applies to an include file with the same basename as your source file; for example, in **'allclass.cc'**, **'#pragma implementation'** by itself is equivalent to **'#pragma implementation "allclass.h"'**. Use the string argument if you want a single implementation file to include code from multiple header files.

There is no way to split up the contents of a single header file into multiple implementation files.

FILES

file.c	C source file
file.h	C header (preprocessor) file
file.i	preprocessed C source file
file.C	C++ source file

file.cc	C++ source file
file.cxx	C++ source file
file.m	Objective-C source file
file.s	assembly language file
file.o	object file
a.out	link edited output
TMPDIR/cc*	temporary files
LIBDIR/cpp	preprocessor
LIBDIR/cc1	compiler for C
LIBDIR/cc1plus	compiler for C++
LIBDIR/collect	linker front end needed on some machines
LIBDIR/libgcc.a	GCC subroutine library
/lib/crt[01n].o	start-up routine
LIBDIR/ccrt0	additional start-up routine for C++
/lib/libc.a	standard C library, see *intro*(3)
/usr/include	standard directory for **#include** files
LIBDIR/include	standard gcc directory for **#include** files
LIBDIR/g++-include	additional g++ directory for **#include**

LIBDIR is usually **/usr/local/lib/***machine*/*version*.
TMPDIR comes from the environment variable **TMPDIR** (default **/usr/tmp** if available, else **/tmp**).

SEE ALSO

cpp(1), as(1), ld(1), gdb(1), adb(1), dbx(1), sdb(1).
'gcc', **'cpp'**, **'as'**, **'ld'**, and **'gdb'** entries in **info.**
Using and Porting GNU CC (for version 2.0), Richard M. Stallman; *The C Preprocessor*, Richard M. Stallman; *Debugging with GDB: the GNU Source-Level Debugger*, Richard M. Stallman and Roland H. Pesch; *Using as: the GNU Assembler*, Dean Elsner, Jay Fenlason & friends; *ld: the GNU linker*, Steve Chamberlain and Roland Pesch.

BUGS

For instructions on reporting bugs, see the GCC manual.

COPYING

Free Software Foundation instead of in the original English.

AUTHORS

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