**?shared**

Produce a shared object which can then be linked with other objects to form an executable. Not all

systems support this option. For predictable results, you must also specify the same set of options that

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

**?fpic**

Generate position-independent code (PIC) suitable for use in a shared library, if supported for the

target machine. Such code accesses all constant addresses through a global offset table (GOT). The

dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of

GCC; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-

specific maximum size, you get an error message from the linker indicating that ?fpic does not work;

in that case, recompile with **?fPIC** instead. (These maximums are 8k on the SPARC and 32k on the

m68k and RS/6000 . The 386 has no such limit.)

Position-independent code requires special support, and therefore works only on certain machines.

For the 386, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM

RS/6000 is always position-independent.

When this flag is set, the macros \_\_pic\_ \_ and \_\_PIC\_ \_ are defined to 1.

**?fPIC**

If supported for the target machine, emit position-independent code, suitable for dynamic linking and

avoiding any limit on the size of the global offset table. This option makes a difference on the m68k,

PowerPC and SPARC .

Position-independent code requires special support, and therefore works only on certain machines.

When this flag is set, the macros \_\_pic\_ \_ and \_\_PIC\_ \_ are defined to 2.

**?fstrict?aliasing**

Allows the compiler to assume the strictest aliasing rules applicable to the language being compiled.

For C (and C++), this activates optimizations based on the type of expressions. In particular, an object

of one type is assumed never to reside at the same address as an object of a different type, unless the

types are almost the same. For example, an unsigned int can alias an int, but not a void\* or a

double. A character type may alias any other type.

Pay special attention to code like this:

union a\_union {

int i;

double d;

};

int f() {

a\_union t;

t.d = 3.0;

return t.i;

}

The practice of reading from a different union member than the one most recently written to (called

‘‘type-punning’’) is common. Even with ?fstrict?aliasing, type-punning is allowed, provided the

memory is accessed through the union type. So, the code above will work as expected. However,

this code might not:

int f() {

a\_union t;

int\* ip;

t.d = 3.0;

ip = &t.i;

return \*ip;

}

Similarly, access by taking the address, casting the resulting pointer and dereferencing the result has

undefined behavior, even if the cast uses a union type, e.g.:

int f() {

double d = 3.0;

return ((union a\_union \*) &d)?>i;

}

The **?fstrict?aliasing** option is enabled at levels **?O2, ?O3, ?Os**.

**?pipe**

Use pipes rather than temporary files for communication between the various stages of compilation.

This fails to work on some systems where the assembler is unable to read from a pipe; but the GNU

assembler has no trouble.

**?Wformat**

Check calls to printf and scanf, etc., to make sure that the arguments supplied have types

appropriate to the format string specified, and that the conversions specified in the format string make

sense. This includes standard functions, and others specified by format attributes, in the printf,

scanf, strftime and strfmon (an X/Open extension, not in the C standard) families (or other

target-specific families). Which functions are checked without format attributes having been specified

depends on the standard version selected, and such checks of functions without the attribute specified

are disabled by ?ffreestanding or ?fno?builtin.

The formats are checked against the format features supported by GNU libc version 2.2. These include

all ISO C90 and C99 features, as well as features from the Single Unix Specification and some BSD

and GNU extensions. Other library implementations may not support all these features; GCC does not

support warning about features that go beyond a particular library’s limitations. However, if

?pedantic is used with ?Wformat, warnings will be given about format features not in the selected

standard version (but not for strfmon formats, since those are not in any version of the C standard).

Since ?Wformat also checks for null format arguments for several functions, ?Wformat also implies

?Wnonnull.

?Wformat is included in ?Wall. For more control over some aspects of format checking, the options

?Wformat?y2k, ?Wno?format?extra?args, ?Wno?format?zero?length, ?Wformat?nonliteral,

?Wformat?security, and ?Wformat=2 are available, but are not included in ?Wall.

**Options Controlling the Preprocessor**

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

If you use the **?E** option, nothing is done except preprocessing. Some of these options make sense only

together with **?E** because they cause the preprocessor output to be unsuitable for actual compilation.

You can use **?Wp**,*option* to bypass the compiler driver and pass option directly through to the

preprocessor. If option contains commas, it is split into multiple options at the commas. However,

many options are modified, translated or interpreted by the compiler driver before being passed to the

preprocessor, and **?Wp** forcibly bypasses this phase. The preprocessor’s direct interface is

undocumented and subject to change, so whenever possible you should avoid using **?Wp** and let the

driver handle the options instead.

**?fexceptions**

Enable exception handling. Generates extra code needed to propagate exceptions. For some targets,

this implies GCC will generate frame unwind information for all functions, which can produce

significant data size overhead, although it does not affect execution. If you do not specify this option,

GCC will enable it by default for languages like C++ which normally require exception handling, and

disable it for languages like C that do not normally require it. However, you may need to enable this

option when compiling C code that needs to interoperate properly with exception handlers written in

C++. You may also wish to disable this option if you are compiling older C++ programs that don’t use

exception handling.

**?fomit?frame?pointer**

Don’t keep the frame pointer in a register for functions that don’t need one. This avoids the

instructions to save, set up and restore frame pointers; it also makes an extra register available in many

functions. **It also makes debugging impossible on some machines.**

On some machines, such as the VAX, this flag has no effect, because the standard calling sequence

automatically handles the frame pointer and nothing is saved by pretending it doesn’t exist. The

machine-description macro FRAME\_POINTER\_REQIREDcontrols whether a target machine

supports this flag.

Enabled at levels **?O, ?O2, ?O3, ?Os.**

**?fasynchronous?unwind?tables**

Generate unwind table in dwarf2 format, if supported by target machine. The table is exact at each

instruction boundary, so it can be used for stack unwinding from asynchronous events (such as

debugger or garbage collector).

**?mtune=name**

This option is very similar to the ?mcpu= option, except that instead of specifying the actual target

processor type, and hence restricting which instructions can be used, it specifies that GCC should tune

the performance of the code as if the target were of the type specified in this option, but still choosing

the instructions that it will generate based on the cpu specified by a **?mcpu=** *option*. For some ARM

implementations better performance can be obtained by using this option.

**?fstack?protector**

Emit extra code to check for buffer overflows, such as stack smashing attacks. This is done by adding

a guard variable to functions with vulnerable objects. This includes functions that call alloca, and

functions with buffers larger than 8 bytes. The guards are initialized when a function is entered and

then checked when the function exits. If a guard check fails, an error message is printed and the

program exits.

**??param** *name =value*

In some places, GCC uses various constants to control the amount of optimization that is done. For

example, GCC will not inline functions that contain more that a certain number of instructions. You

can control some of these constants on the command-line using the ??param option.

The names of specific parameters, and the meaning of the values, are tied to the internals of the

compiler, and are subject to change without notice in future releases.

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

**ssp-buffer-size**

The minimum size of buffers (i.e. arrays) that will receive stack smashing protection when

?fstack?protection is used.