NAME

gcc − GNU project C and C++ compiler

SYNOPSIS

gcc [−**c**|−**S**|−**E**] [−**std=***standard*]
  [−**g**] [−**pg**] [−**O***level*]
  [−**W***warn*...] [−**Wpedantic**]
  [−**I***dir*...] [−**L***dir*...]
  [−**D***macro*[=*defn*]...] [−**U***macro*]
  [−**f***option*...] [−**m***machine−option*...]
  [−**o** *outfile*] [@*file*] *infile*...

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

DESCRIPTION

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

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

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

The usual way to run GCC is to run the executable called **gcc**, or *machine−***gcc** when cross−compiling, or *machine−***gcc−***version* to run a specific version of GCC. When you compile C++ programs, you should invoke GCC as **g++** instead.

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

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

Many options have long names starting with −**f** or with −**W**−−−for example, −**fmove−loop−invariants**, −**Wformat** and so on. Most of these have both positive and negative forms; the negative form of −**ffoo** is −**fno−foo**. This manual documents only one of these two forms, whichever one is not the default.

Some options take one or more arguments typically separated either by a space or by the equals sign (**=**) from the option name. Unless documented otherwise, an argument can be either numeric or a string. Numeric arguments must typically be small unsigned decimal or hexadecimal integers. Hexadecimal arguments must begin with the **0x** prefix. Arguments to options that specify a size threshold of some sort may be arbitrarily large decimal or hexadecimal integers followed by a byte size suffix designating a multiple of bytes such as kB and KiB for kilobyte and kibibyte, respectively, MB and MiB for megabyte and mebibyte, GB and GiB for gigabyte and gigibyte, and so on. Such arguments are designated by *byte−size* in the following text. Refer to the NIST, IEC, and other relevant national and international standards for the full listing and explanation of the binary and decimal byte size prefixes.

OPTIONS

Option Summary

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

Overall Options

−**c** −**S** −**E** −**o** *file* −**dumpbase** *dumpbase* −**dumpbase−ext** *auxdropsuf* −**dumpdir** *dumppfx* −**x** *language* −**v** −**###** −−**help**[=*class*[**,...**]] −−**target−help** −−**version** −**pass−exit−codes** −**pipe** −**specs=***file* −**wrapper** @*file* −**ffile−prefix−map=***old*=*new* −**fcanon−prefix−map** −**fplugin=***file*

      −fplugin−arg−*name*=*arg*   **−fdump−ada−spec**[**−slim**]   **−fada−spec−parent=***unit*
      −fdump−go−spec=*file*

C Language Options
      −ansi **−std=***standard* **−aux−info** *filename* **−fno−asm** **−fno−builtin** **−fno−builtin−***function*
      −fcond−mismatch **−ffreestanding** **−fgimple** **−fgnu−tm** **−fgnu89−inline** **−fhosted**
      −flax−vector−conversions **−fms−extensions** **−fpermitted−flt−eval−methods=***standard*
      −fplan9−extensions **−fsigned−bitfields** **−funsigned−bitfields** **−fsigned−char** **−funsigned−char**
      −fstrict−flex−arrays[=*n*] **−fsso−struct=***endianness*

C++ Language Options
      −fabi−version=*n* **−fno−access−control** **−faligned−new=***n*
      −fno−assume−sane−operators−new−delete **−fchar8_t** **−fcheck−new** **−fconcepts**
      −fconstexpr−depth=*n* **−fconstexpr−cache−depth=***n* **−fconstexpr−loop−limit=***n*
      −fconstexpr−ops−limit=*n* **−fno−elide−constructors** **−fno−enforce−eh−specs** **−fno−gnu−keywords**
      −fno−immediate−escalation **−fno−implicit−templates** **−fno−implicit−inline−templates**
      −fno−implement−inlines **−fmodule−header**[=*kind*] **−fmodule−only** **−fmodules**
      −fmodule−implicit−inline **−fno−module−lazy** **−fmodule−mapper=***specification*
      −fmodule−version−ignore **−fms−extensions** **−fnew−inheriting−ctors** **−fnew−ttp−matching**
      −fno−nonansi−builtins **−fnothrow−opt** **−fno−operator−names** **−fno−optional−diags**
      −fno−pretty−templates **−frange−for−ext−temps** **−fno−rtti** **−fsized−deallocation**
      −fstrong−eval−order[=*kind*] **−ftemplate−backtrace−limit=***n* **−ftemplate−depth=***n*
      −fno−threadsafe−statics **−fuse−cxa−atexit** **−fno−weak** **−nostdinc++** **−fvisibility−inlines−hidden**
      −fvisibility−ms−compat **−fext−numeric−literals** **−flang−info−include−translate**[=*header*]
      −flang−info−include−translate−not **−flang−info−module−cmi**[=*module*] **−stdlib=***libstdc++,libc++*
      −Wabi−tag **−Wcatch−value** **−Wcatch−value=***n* **−Wno−class−conversion** **−Wclass−memaccess**
      −Wcomma−subscript **−Wconditionally−supported** **−Wno−conversion−null**
      −Wctad−maybe−unsupported **−Wctor−dtor−privacy** **−Wdangling−reference**
      −Wno−defaulted−function−deleted **−Wno−delete−incomplete** **−Wdelete−non−virtual−dtor**
      −Wno−deprecated−array−compare **−Wdeprecated−copy** **−Wdeprecated−copy−dtor**
      −Wno−deprecated−enum−enum−conversion **−Wno−deprecated−enum−float−conversion**
      −Weffc++ **−Wno−elaborated−enum−base** **−Wno−exceptions** **−Wextra−semi**
      −Wno−global−module **−Wno−inaccessible−base** **−Wno−inherited−variadic−ctor**
      −Wno−init−list−lifetime **−Winvalid−constexpr** **−Winvalid−imported−macros**
      −Wno−invalid−offsetof **−Wno−literal−suffix** **−Wmismatched−new−delete** **−Wmismatched−tags**
      −Wmultiple−inheritance **−Wnamespaces** **−Wnarrowing** **−Wnoexcept** **−Wnoexcept−type**
      −Wnon−virtual−dtor **−Wpessimizing−move** **−Wno−placement−new** **−Wplacement−new=***n*
      −Wrange−loop−construct **−Wredundant−move** **−Wredundant−tags** **−Wreorder** **−Wregister**
      −Wstrict−null−sentinel **−Wno−subobject−linkage** **−Wtemplates** **−Wno−non−template−friend**
      −Wold−style−cast **−Woverloaded−virtual** **−Wno−pmf−conversions** **−Wself−move**
      −Wsign−promo **−Wsized−deallocation** **−Wsuggest−final−methods** **−Wsuggest−final−types**
      −Wsuggest−override **−Wno−template−body** **−Wno−template−id−cdtor**
      −Wtemplate−names−tu−local **−Wno−terminate** **−Wno−vexing−parse** **−Wvirtual−inheritance**
      −Wno−virtual−move−assign **−Wvolatile**

Objective−C and Objective−C++ Language Options
      −fconstant−string−class=*class−name* **−fgnu−runtime** **−fnext−runtime** **−fno−nil−receivers**
      −fobjc−abi−version=*n* **−fobjc−call−cxx−cdtors** **−fobjc−direct−dispatch** **−fobjc−exceptions**
      −fobjc−gc **−fobjc−nilcheck** **−fobjc−std=objc1** **−fno−local−ivars**
      −fivar−visibility=[**public**|**protected**|**private**|**package**] **−freplace−objc−classes** **−fzero−link**
      −gen−decls **−Wassign−intercept** **−Wno−property−assign−default** **−Wno−protocol**
      −Wobjc−root−class **−Wselector** **−Wstrict−selector−match** **−Wundeclared−selector**

OpenMP and OpenACC Options
      −foffload=*arg* **−foffload−options=***arg* **−fopenacc** **−fopenacc−dim=***geom* **−fopenmp**
      −fopenmp−simd **−fopenmp−target−simd−clone**[=*device−type*]

Diagnostic Message Formatting Options

 −fmessage−length=*n* **−fdiagnostics−plain−output −fdiagnostics−show−location=[once|every−line]**
 −fdiagnostics−color=[auto|never|always]   **−fdiagnostics−urls=[auto|never|always]**
 −fdiagnostics−format=[text|sarif−stderr|sarif−file|json|json−stderr|json−file]
 −fdiagnostics−add−output=*DIAGNOSTICS−OUTPUT−SPEC*
 −fdiagnostics−set−output=*DIAGNOSTICS−OUTPUT−SPEC* **−fno−diagnostics−json−formatting**
 −fno−diagnostics−show−option   **−fno−diagnostics−show−caret**
 −fno−diagnostics−show−event−links   **−fno−diagnostics−show−labels**
 −fno−diagnostics−show−line−numbers   **−fno−diagnostics−show−cwe**
 −fno−diagnostics−show−rules   **−fno−diagnostics−show−highlight−colors**
 −fdiagnostics−minimum−margin−width=*width*   **−fdiagnostics−parseable−fixits**
 −fdiagnostics−generate−patch **−fdiagnostics−show−template−tree** **−fno−elide−type**
 −fdiagnostics−path−format=[none|separate−events|inline−events]
 −fdiagnostics−show−path−depths −fno−show−column −fdiagnostics−column−unit=[display|byte]
 −fdiagnostics−column−origin=*origin*   **−fdiagnostics−escape−format=[unicode|bytes]**
 −fdiagnostics−text−art−charset=[none|ascii|unicode|emoji]

Warning Options

 −fsyntax−only −fmax−errors=*n* **−Wpedantic −pedantic−errors −fpermissive −w −Wextra**
 −Wall −Wabi=*n* **−Waddress −Wno−address−of−packed−member −Waggregate−return**
 −Walloc−size   **−Walloc−size−larger−than=***byte−size* **−Walloc−zero**   **−Walloca**
 −Walloca−larger−than=*byte−size* **−Wno−aggressive−loop−optimizations −Warith−conversion**
 −Warray−bounds **−Warray−bounds=***n* **−Warray−compare** **−Warray−parameter**
 −Warray−parameter=*n* **−Wno−attributes −Wattribute−alias=***n* **−Wno−attribute−alias**
 −Wno−attribute−warning **−Wbidi−chars=[none|unpaired|any|ucn]** **−Wbool−compare**
 −Wbool−operation −Wno−builtin−declaration−mismatch −Wno−builtin−macro−redefined
 −Wc90−c99−compat **−Wc99−c11−compat** **−Wc11−c23−compat** **−Wc23−c2y−compat**
 −Wc++−compat −Wc++11−compat −Wc++14−compat −Wc++17−compat −Wc++20−compat
 −Wno−c++11−extensions   **−Wno−c++14−extensions**   **−Wno−c++17−extensions**
 −Wno−c++20−extensions −Wno−c++23−extensions −Wcalloc−transposed−args −Wcast−align
 −Wcast−align=strict −Wcast−function−type −Wcast−qual −Wchar−subscripts −Wclobbered
 −Wcomment −Wcompare−distinct−pointer−types −Wno−complain−wrong−lang −Wconversion
 −Wno−coverage−mismatch **−Wno−cpp** **−Wdangling−else** **−Wdangling−pointer**
 −Wdangling−pointer=*n* **−Wdate−time −Wno−deprecated −Wno−deprecated−declarations**
 −Wno−designated−init **−Wdisabled−optimization** **−Wno−discarded−array−qualifiers**
 −Wno−discarded−qualifiers   **−Wno−div−by−zero**   **−Wdouble−promotion**
 −Wduplicated−branches **−Wduplicated−cond** **−Wempty−body** **−Wno−endif−labels**
 −Wenum−compare **−Wenum−conversion** **−Wenum−int−mismatch** **−Werror** **−Werror=***
 −Wexpansion−to−defined   **−Wfatal−errors**   **−Wflex−array−member−not−at−end**
 −Wfloat−conversion −Wfloat−equal −Wformat −Wformat=2 −Wno−format−contains−nul
 −Wno−format−extra−args −Wformat−nonliteral −Wformat−overflow=*n* **−Wformat−security**
 −Wformat−signedness **−Wformat−truncation=***n* **−Wformat−y2k** **−Wframe−address**
 −Wframe−larger−than=*byte−size* **−Wno−free−nonheap−object** **−Wheader−guard**
 −Wno−if−not−aligned **−Wno−ignored−attributes** **−Wignored−qualifiers**
 −Wno−incompatible−pointer−types −Whardened −Wimplicit −Wimplicit−fallthrough
 −Wimplicit−fallthrough=*n* **−Wno−implicit−function−declaration** **−Wno−implicit−int**
 −Winfinite−recursion −Winit−self −Winline −Wno−int−conversion −Wint−in−bool−context
 −Wno−int−to−pointer−cast −Wno−invalid−memory−model −Winvalid−pch −Winvalid−utf8
 −Wno−unicode −Wjump−misses−init −Wlarger−than=*byte−size* **−Wleading−whitespace=***kind*
 −Wlogical−not−parentheses −Wlogical−op −Wlong−long −Wno−lto−type−mismatch −Wmain
 −Wmaybe−uninitialized **−Wmemset−elt−size**   **−Wmemset−transposed−args**
 −Wmisleading−indentation **−Wmissing−attributes** **−Wmissing−braces**
 −Wmissing−field−initializers **−Wmissing−format−attribute** **−Wmissing−include−dirs**
 −Wmissing−noreturn **−Wmusttail−local−addr** **−Wmaybe−musttail−local−addr**
 −Wno−missing−profile **−Wno−multichar** **−Wmultistatement−macros** **−Wnonnull**

−Wnonnull−compare   −Wnormalized=[none|id|nfc|nfkc]   −Wnull−dereference   −Wno−odr
−Wopenacc−parallelism −Wopenmp −Wopenmp−simd −Wno−overflow −Woverlength−strings
−Wno−override−init−side−effects   −Wpacked   −Wno−packed−bitfield−compat
−Wpacked−not−aligned   −Wpadded   −Wparentheses   −Wno−pedantic−ms−format
−Wpointer−arith   −Wno−pointer−compare   −Wno−pointer−to−int−cast   −Wno−pragmas
−Wno−pragma−once−outside−header −Wno−prio−ctor−dtor −Wno−psabi −Wredundant−decls
−Wrestrict   −Wno−return−local−addr   −Wreturn−type   −Wno−scalar−storage−order
−Wsequence−point   −Wshadow   −Wshadow=global   −Wshadow=local
−Wshadow=compatible−local   −Wno−shadow−ivar   −Wno−shift−count−negative
−Wno−shift−count−overflow   −Wshift−negative−value   −Wno−shift−overflow
−Wshift−overflow=*n* −Wsign−compare −Wsign−conversion −Wno−sizeof−array−argument
−Wsizeof−array−div −Wsizeof−pointer−div −Wsizeof−pointer−memaccess −Wstack−protector
−Wstack−usage=*byte−size*   −Wstrict−aliasing   −Wstrict−aliasing=n   −Wstrict−overflow
−Wstrict−overflow=*n* −Wstring−compare −Wno−stringop−overflow −Wno−stringop−overread
−Wno−stringop−truncation   −Wstrict−flex−arrays   −Wsuggest−attribute=*attribute−name*
−Wswitch   −Wno−switch−bool   −Wswitch−default   −Wswitch−enum
−Wno−switch−outside−range −Wno−switch−unreachable −Wsync−nand −Wsystem−headers
−Wtautological−compare −Wtrailing−whitespace −Wtrailing−whitespace=*kind* −Wtrampolines
−Wtrigraphs −Wtrivial−auto−var−init −Wno−tsan −Wtype−limits −Wundef −Wuninitialized
−Wunknown−pragmas −Wunsuffixed−float−constants −Wunterminated−string−initialization
−Wunused   −Wunused−but−set−parameter   −Wunused−but−set−variable
−Wunused−const−variable −Wunused−const−variable=*n* −Wunused−function −Wunused−label
−Wunused−local−typedefs −Wunused−macros −Wunused−parameter   −Wno−unused−result
−Wunused−value −Wunused−variable −Wuse−after−free −Wuse−after−free=*n* −Wuseless−cast
−Wno−varargs   −Wvariadic−macros   −Wvector−operation−performance   −Wvla
−Wvla−larger−than=*byte−size*   −Wno−vla−larger−than   −Wvolatile−register−var
−Wwrite−strings   −Wno−xor−used−as−pow   −Wzero−as−null−pointer−constant
−Wzero−length−bounds

Static Analyzer Options
−fanalyzer −fanalyzer−call−summaries −fanalyzer−checker=*name* −fno−analyzer−feasibility
−fanalyzer−fine−grained   −fanalyzer−show−events−in−system−headers
−fno−analyzer−state−merge   −fno−analyzer−state−purge   −fno−analyzer−suppress−followups
−fanalyzer−transitivity   −fno−analyzer−undo−inlining   −fanalyzer−verbose−edges
−fanalyzer−verbose−state−changes   −fanalyzer−verbosity=*level*   −fdump−analyzer
−fdump−analyzer−callgraph   −fdump−analyzer−exploded−graph
−fdump−analyzer−exploded−nodes   −fdump−analyzer−exploded−nodes−2
−fdump−analyzer−exploded−nodes−3   −fdump−analyzer−exploded−paths
−fdump−analyzer−feasibility   −fdump−analyzer−infinite−loop   −fdump−analyzer−json
−fdump−analyzer−state−purge   −fdump−analyzer−stderr   −fdump−analyzer−supergraph
−fdump−analyzer−untracked   −Wno−analyzer−double−fclose   −Wno−analyzer−double−free
−Wno−analyzer−exposure−through−output−file
−Wno−analyzer−exposure−through−uninit−copy   −Wno−analyzer−fd−access−mode−mismatch
−Wno−analyzer−fd−double−close   −Wno−analyzer−fd−leak
−Wno−analyzer−fd−phase−mismatch   −Wno−analyzer−fd−type−mismatch
−Wno−analyzer−fd−use−after−close   −Wno−analyzer−fd−use−without−check
−Wno−analyzer−file−leak   −Wno−analyzer−free−of−non−heap
−Wno−analyzer−imprecise−fp−arithmetic   −Wno−analyzer−infinite−loop
−Wno−analyzer−infinite−recursion   −Wno−analyzer−jump−through−null
−Wno−analyzer−malloc−leak   −Wno−analyzer−mismatching−deallocation
−Wno−analyzer−null−argument   −Wno−analyzer−null−dereference
−Wno−analyzer−out−of−bounds   −Wno−analyzer−overlapping−buffers
−Wno−analyzer−possible−null−argument   −Wno−analyzer−possible−null−dereference
−Wno−analyzer−putenv−of−auto−var   −Wno−analyzer−shift−count−negative
−Wno−analyzer−shift−count−overflow   −Wno−analyzer−stale−setjmp−buffer

−Wno−analyzer−tainted−allocation−size   **−Wno−analyzer−tainted−assertion**
−Wno−analyzer−tainted−array−index   **−Wno−analyzer−tainted−divisor**
−Wno−analyzer−tainted−offset   **−Wno−analyzer−tainted−size**
−Wanalyzer−symbol−too−complex   **−Wanalyzer−too−complex**
−Wno−analyzer−undefined−behavior−ptrdiff   **−Wno−analyzer−undefined−behavior−strtok**
−Wno−analyzer−unsafe−call−within−signal−handler   **−Wno−analyzer−use−after−free**
−Wno−analyzer−use−of−pointer−in−stale−stack−frame
−Wno−analyzer−use−of−uninitialized−value   **−Wno−analyzer−va−arg−type−mismatch**
−Wno−analyzer−va−list−exhausted   **−Wno−analyzer−va−list−leak**
−Wno−analyzer−va−list−use−after−va−end   **−Wno−analyzer−write−to−const**
−Wno−analyzer−write−to−string−literal

C and Objective−C−only Warning Options
−Wbad−function−cast −Wdeprecated−non−prototype −Wfree−labels −Wmissing−declarations
−Wmissing−parameter−name   **−Wmissing−parameter−type**
−Wdeclaration−missing−parameter−type   **−Wmissing−prototypes**
−Wmissing−variable−declarations   **−Wnested−externs**   **−Wold−style−declaration**
−Wold−style−definition   **−Wstrict−prototypes**   **−Wtraditional**   **−Wtraditional−conversion**
−Wdeclaration−after−statement −Wpointer−sign

Debugging Options
−g **−g***level* **−gdwarf** **−gdwarf−***version* **−gbtf** **−gctf** **−gctf***level* **−gprune−btf** **−gno−prune−btf**
−ggdb **−grecord−gcc−switches** **−gno−record−gcc−switches** **−gstrict−dwarf** **−gno−strict−dwarf**
−gas−loc−support **−gno−as−loc−support** **−gas−locview−support** **−gno−as−locview−support**
−gcodeview **−gcolumn−info** **−gno−column−info** **−gdwarf32** **−gdwarf64** **−gstatement−frontiers**
−gno−statement−frontiers   **−gvariable−location−views**   **−gno−variable−location−views**
−ginternal−reset−location−views   **−gno−internal−reset−location−views**   **−ginline−points**
−gno−inline−points **−gvms** **−gz**[=*type*] **−gsplit−dwarf** **−gdescribe−dies** **−gno−describe−dies**
−fdebug−prefix−map=*old*=*new*   **−fdebug−types−section**   **−fno−eliminate−unused−debug−types**
−femit−struct−debug−baseonly   **−femit−struct−debug−reduced**
−femit−struct−debug−detailed[=*spec−list*]   **−fno−eliminate−unused−debug−symbols**
−femit−class−debug−always **−fno−merge−debug−strings** **−fno−dwarf2−cfi−asm** **−fvar−tracking**
−fvar−tracking−assignments

Optimization Options
−faggressive−loop−optimizations   **−falign−functions**[=*n*[**:***m*:[*n2*[**:***m2*]]]]
−falign−jumps[=*n*[**:***m*:[*n2*[**:***m2*]]]]   **−falign−labels**[=*n*[**:***m*:[*n2*[**:***m2*]]]]
−falign−loops[=*n*[**:***m*:[*n2*[**:***m2*]]]]   **−fmin−function−alignment=**[*n*]   **−fno−allocation−dce**
−fallow−store−data−races   **−fassociative−math**   **−fauto−profile**   **−fauto−profile**[=*path*]
−fauto−inc−dec   **−fbranch−probabilities**   **−fcaller−saves**   **−fcombine−stack−adjustments**
−fconserve−stack **−ffold−mem−offsets** **−fcompare−elim** **−fcprop−registers** **−fcrossjumping**
−fcse−follow−jumps **−fcse−skip−blocks** **−fcx−fortran−rules** **−fcx−limited−range** **−fdata−sections**
−fdce   **−fdelayed−branch**   **−fdelete−null−pointer−checks**   **−fdevirtualize**
−fdevirtualize−speculatively **−fdevirtualize−at−ltrans** **−fdse** **−fearly−inlining** **−fipa−sra**
−fexpensive−optimizations **−ffat−lto−objects** **−ffast−math** **−ffinite−math−only** **−ffloat−store**
−fexcess−precision=*style*   **−ffinite−loops**   **−fforward−propagate**   **−ffp−contract=***style*
−ffunction−sections **−fgcse** **−fgcse−after−reload** **−fgcse−las** **−fgcse−lm** **−fgraphite−identity**
−fgcse−sm **−fhoist−adjacent−loads** **−fif−conversion** **−fif−conversion2** **−findirect−inlining**
−finline−stringops[=*fn*] **−finline−functions** **−finline−functions−called−once** **−finline−limit=***n*
−finline−small−functions **−fipa−modref** **−fipa−cp** **−fipa−cp−clone** **−fipa−bit−cp** **−fipa−vrp**
−fipa−pta **−fipa−profile** **−fipa−pure−const** **−fipa−reference** **−fipa−reference−addressable**
−fipa−reorder−for−locality **−fipa−stack−alignment** **−fipa−icf** **−fira−algorithm=***algorithm*
−flate−combine−instructions **−flifetime−dse** **−flive−patching=***level* **−fira−region=***region*
−fira−hoist−pressure   **−fira−loop−pressure**   **−fno−ira−share−save−slots**
−fno−ira−share−spill−slots   **−fisolate−erroneous−paths−dereference**
−fisolate−erroneous−paths−attribute −fivopts −fkeep−inline−functions −fkeep−static−functions

−fkeep−static−consts   −flimit−function−alignment   −flive−range−shrinkage   −floop−block
−floop−interchange   −floop−strip−mine   −floop−unroll−and−jam   −floop−nest−optimize
−floop−parallelize−all   −flra−remat   −flto   −flto−compression−level   −flto−partition=*alg*
−flto−incremental=*path*   −flto−incremental−cache−size=*n*   −fmalloc−dce   −fmerge−all−constants
−fmerge−constants   −fmodulo−sched   −fmodulo−sched−allow−regmoves
−fmove−loop−invariants   −fmove−loop−stores   −fno−branch−count−reg   −fno−defer−pop
−fno−fp−int−builtin−inexact   −fno−function−cse   −fno−guess−branch−probability   −fno−inline
−fno−math−errno   −fno−peephole   −fno−peephole2   −fno−printf−return−value
−fno−sched−interblock   −fno−sched−spec   −fno−signed−zeros   −fno−toplevel−reorder
−fno−trapping−math   −fno−zero−initialized−in−bss   −fomit−frame−pointer   −foptimize−crc
−foptimize−sibling−calls   −fpartial−inlining   −fpeel−loops   −fpredictive−commoning
−fprefetch−loop−arrays   −fprofile−correction   −fprofile−use   −fprofile−use=*path*
−fprofile−partial−training   −fprofile−values   −fprofile−reorder−functions   −freciprocal−math
−free   −frename−registers   −freorder−blocks   −freorder−blocks−algorithm=*algorithm*
−freorder−blocks−and−partition   −freorder−functions   −frerun−cse−after−loop
−freschedule−modulo−scheduled−loops   −frounding−math   −fsave−optimization−record
−fsched2−use−superblocks   −fsched−pressure   −fsched−spec−load
−fsched−spec−load−dangerous   −fsched−stalled−insns−dep[=*n*]   −fsched−stalled−insns[=*n*]
−fsched−group−heuristic   −fsched−critical−path−heuristic   −fsched−spec−insn−heuristic
−fsched−rank−heuristic   −fsched−last−insn−heuristic   −fsched−dep−count−heuristic
−fschedule−fusion   −fschedule−insns   −fschedule−insns2   −fsection−anchors
−fselective−scheduling   −fselective−scheduling2   −fsel−sched−pipelining
−fsel−sched−pipelining−outer−loops   −fsemantic−interposition   −fshrink−wrap
−fshrink−wrap−separate   −fsignaling−nans   −fsingle−precision−constant   −fsplit−ivs−in−unroller
−fsplit−loops   −fsplit−paths   −fsplit−wide−types   −fsplit−wide−types−early   −fssa−backprop
−fssa−phiopt   −fstdarg−opt   −fstore−merging   −fstrict−aliasing   −fipa−strict−aliasing
−fthread−jumps   −ftracer   −ftree−bit−ccp   −ftree−builtin−call−dce   −ftree−ccp   −ftree−ch
−ftree−coalesce−vars   −ftree−copy−prop   −ftree−cselim   −ftree−dce   −ftree−dominator−opts
−ftree−dse   −ftree−forwprop   −ftree−fre   −fcode−hoisting   −ftree−loop−if−convert
−ftree−loop−im   −ftree−phiprop   −ftree−loop−distribution   −ftree−loop−distribute−patterns
−ftree−loop−ivcanon   −ftree−loop−linear   −ftree−loop−optimize   −ftree−loop−vectorize
−ftree−parallelize−loops=*n*   −ftree−pre   −ftree−partial−pre   −ftree−pta   −ftree−reassoc
−ftree−scev−cprop   −ftree−sink   −ftree−slsr   −ftree−sra   −ftree−switch−conversion
−ftree−tail−merge   −ftree−ter   −ftree−vectorize   −ftree−vrp   −ftrivial−auto−var−init
−funconstrained−commons   −funit−at−a−time   −funroll−all−loops   −funroll−loops
−funsafe−math−optimizations   −funswitch−loops   −fipa−ra   −fvariable−expansion−in−unroller
−fvect−cost−model   −fvpt   −fweb   −fwhole−program   −fwpa   −fuse−linker−plugin
−fzero−call−used−regs −−param *name*=*value* −O −O0 −O1 −O2 −O3 −Os −Ofast −Og −Oz

Program Instrumentation Options
−p −pg −fprofile−arcs −−coverage −ftest−coverage −fcondition−coverage −fpath−coverage
−fprofile−abs−path   −fprofile−dir=*path*   −fprofile−generate   −fprofile−generate=*path*
−fprofile−info−section   −fprofile−info−section=*name*   −fprofile−note=*path*
−fprofile−prefix−path=*path*   −fprofile−update=*method*   −fprofile−filter−files=*regex*
−fprofile−exclude−files=*regex*   −fprofile−reproducible=[**multithreaded|parallel−runs|serial**]
−fsanitize=*style*   −fsanitize−recover   −fsanitize−recover=*style*   −fsanitize−trap
−fsanitize−trap=*style*   −fasan−shadow−offset=*number*   −fsanitize−sections=*s1*,*s2*,...
−fsanitize−undefined−trap−on−error   −fbounds−check   −fcf−protection
−fcf−protection=[**full|branch|return|none|check**]   −fharden−compares
−fharden−conditional−branches   −fhardened   −fharden−control−flow−redundancy
−fhardcfr−skip−leaf   −fhardcfr−check−exceptions   −fhardcfr−check−returning−calls
−fhardcfr−check−noreturn−calls=[**always|no−xthrow|nothrow|never**]   −fstack−protector
−fstack−protector−all   −fstack−protector−strong   −fstack−protector−explicit   −fstack−check
−fstack−limit−register=*reg*   −fstack−limit−symbol=*sym*   −fno−stack−limit   −fsplit−stack
−fstrub=disable −fstrub=strict −fstrub=relaxed −fstrub=all −fstrub=at−calls −fstrub=internal

−fvtable−verify=[std|preinit|none]   −fvtv−counts   −fvtv−debug   −finstrument−functions
−finstrument−functions−once   −finstrument−functions−exclude−function−list=*sym***,***sym***,...**
−finstrument−functions−exclude−file−list=*file***,***file***,...   −fprofile−prefix−map=***old***=***new*
−fpatchable−function−entry=*N*[**,***M*]

Preprocessor Options
    −A*question***=***answer* **−A−***question*[**=***answer*] **−C −CC −D***macro*[**=***defn*] **−dD −dI −dM −dN −dU**
    −fdebug−cpp   −fdirectives−only   −fdollars−in−identifiers   −fexec−charset=*charset*
    −fextended−identifiers   −finput−charset=*charset*   **−fmacro−prefix−map=***old***=***new*
    −fmax−include−depth=*depth* **−fno−canonical−system−headers −fpch−deps −fpch−preprocess**
    −fpreprocessed   −ftabstop=*width* **−ftrack−macro−expansion −fwide−exec−charset=***charset*
    −fworking−directory −H −imacros *file* **−include** *file* **−M −MD −MF −MG −MM −MMD**
    −MP −MQ −MT −Mno−modules −no−integrated−cpp −P −pthread −remap −traditional
    −traditional−cpp −trigraphs −U*macro* **−undef −Wp,***option* **−Xpreprocessor** *option*

Assembler Options
    −Wa,*option* **−Xassembler** *option*

Linker Options
    object−file−name **−fuse−ld=***linker* **−l***library* **−nostartfiles −nodefaultlibs −nolibc −nostdlib**
    −nostdlib++ −e *entry* **−−entry=***entry* **−pie −pthread −r −rdynamic −s −static −static−pie**
    −static−libgcc   −static−libstdc++   −static−libasan   −static−libtsan   −static−liblsan
    −static−libubsan −shared −shared−libgcc −symbolic −T *script* **−Wl,***option* **−Xlinker** *option* **−u**
    symbol **−z** *keyword*

Directory Options
    −B*prefix* **−I***dir* **−I− −idirafter** *dir* **−imacros** *file* **−imultilib** *dir* **−iplugindir=***dir* **−iprefix** *file*
    −iquote *dir* **−isysroot** *dir* **−isystem** *dir* **−iwithprefix** *dir* **−iwithprefixbefore** *dir* **−−embed−dir=***dir*
    −L*dir* **−no−canonical−prefixes −−no−sysroot−suffix −nostdinc −nostdinc++ −−sysroot=***dir*

Code Generation Options
    −fcall−saved−*reg*   **−fcall−used−***reg*   **−ffixed−***reg*   **−fexceptions   −fnon−call−exceptions**
    −fdelete−dead−exceptions −funwind−tables −fasynchronous−unwind−tables −fno−gnu−unique
    −finhibit−size−directive −fcommon −fno−ident −fpcc−struct−return −fpic −fPIC −fpie −fPIE
    −fno−plt −fno−jump−tables −fno−bit−tests −frecord−gcc−switches −freg−struct−return
    −fshort−enums −fshort−wchar −fverbose−asm −fpack−struct[=*n*] **−fleading−underscore**
    −ftls−model=*model* **−fstack−reuse=***reuse_level* **−ftrampolines −ftrampoline−impl=[stack|heap]**
    −ftrapv   −fwrapv   −fvisibility=[default|internal|hidden|protected]   −fstrict−volatile−bitfields
    −fsync−libcalls −fzero−init−padding−bits=*value*

Developer Options
    −d*letters*   **−dumpspecs   −dumpmachine   −dumpversion   −dumpfullversion**
    −fcallgraph−info[=su,da]   −fchecking   −fchecking=*n*   **−fdbg−cnt−list**
    −fdbg−cnt=*counter−value−list*   **−fdisable−ipa−***pass_name*   **−fdisable−rtl−***pass_name*
    −fdisable−rtl−*pass−name***=***range−list*   **−fdisable−tree−***pass_name*
    −fdisable−tree−*pass−name***=***range−list* **−fdump−debug −fdump−earlydebug −fdump−noaddr**
    −fdump−unnumbered −fdump−unnumbered−links −fdump−final−insns[=*file*] **−fdump−ipa−all**
    −fdump−ipa−cgraph   −fdump−ipa−inline   −fdump−lang−all   −fdump−lang−*switch*
    −fdump−lang−*switch−options*   **−fdump−lang−***switch−options***=***filename*   **−fdump−passes**
    −fdump−rtl−*pass*   **−fdump−rtl−***pass***=***filename*   **−fdump−statistics   −fdump−tree−all**
    −fdump−tree−*switch*   **−fdump−tree−***switch−options*   **−fdump−tree−***switch−options***=***filename*
    −fcompare−debug[=*opts*]   **−fcompare−debug−second**   **−fenable−***kind−pass*
    −fenable−*kind−pass***=***range−list*   **−fira−verbose=***n*   **−flto−report**   **−flto−report−wpa**
    −fmem−report−wpa   −fmem−report   −fpre−ipa−mem−report   −fpost−ipa−mem−report
    −fopt−info   −fopt−info−*options*[**=***file*] **−fmultiflags −fprofile−report −frandom−seed=***string*
    −fsched−verbose=*n*   **−fsel−sched−verbose**   **−fsel−sched−dump−cfg**
    −fsel−sched−pipelining−verbose −fstats −fstack−usage −ftime−report −ftime−report−details
    −fvar−tracking−assignments−toggle   **−gtoggle**   **−print−file−name=***library*

−print−libgcc−file−name −print−multi−directory −print−multi−lib −print−multi−os−directory
−print−prog−name=*program*   **−print−search−dirs**   **−Q**   **−print−sysroot**
−print−sysroot−headers−suffix −save−temps −save−temps=cwd −save−temps=obj −time[**=***file*]

Machine−Dependent Options

AArch64 Options (**AArch64 Options**) **−mabi=***name* **−mbig−endian −mlittle−endian**
−mgeneral−regs−only −mcmodel=tiny −mcmodel=small −mcmodel=large −mstrict−align
−mno−strict−align −momit−leaf−frame−pointer −mtls−dialect=desc −mtls−dialect=traditional
−mtls−size=*size*   **−mfix−cortex−a53−835769**   **−mfix−cortex−a53−843419**
−mlow−precision−recip−sqrt   **−mlow−precision−sqrt**   **−mlow−precision−div**
−mpc−relative−literal−loads   **−msign−return−address=***scope*
−mbranch−protection=*none*|*standard*|*pac−ret*[**+***leaf*   **+***b−key*]|*bti*|*gcs*   **−mharden−sls=***opts*
−march=*name* **−mcpu=***name* **−mtune=***name* **−moverride=***string* **−mverbose−cost−dump**
−mstack−protector−guard=*guard*   **−mstack−protector−guard−reg=***sysreg*
−mstack−protector−guard−offset=*offset*   **−mtrack−speculation**   **−moutline−atomics**
−mearly−ldp−fusion −mlate−ldp−fusion −Wexperimental−fmv−target

Adapteva Epiphany Options (**Adapteva Epiphany Options**)   **−mhalf−reg−file**
−mprefer−short−insn−regs −mbranch−cost=*num* **−mcmove −mnops=***num* **−msoft−cmpsf**
−msplit−lohi **−mpost−inc** **−mpost−modify** **−mstack−offset=***num* **−mround−nearest**
−mlong−calls **−mshort−calls** **−msmall16** **−mfp−mode=***mode* **−mvect−double**
−max−vect−align=*num* **−msplit−vecmove−early** **−m1reg−***reg*

AMD GCN Options (**AMD GCN Options**) **−march=***gpu* **−mtune=***gpu* **−mstack−size=***bytes*

ARC Options (**ARC Options**) **−mbarrel−shifter** **−mjli−always** **−mcpu=***cpu* **−mA6** **−mARC600**
−mA7 **−mARC700** **−mdpfp** **−mdpfp−compact** **−mdpfp−fast** **−mno−dpfp−lrsr** **−mea**
−mno−mpy **−mmul32x16** **−mmul64** **−matomic** **−mnorm** **−mspfp** **−mspfp−compact**
−mspfp−fast **−msimd** **−msoft−float** **−mswap** **−mcrc** **−mdsp−packa** **−mdvbf** **−mlock**
−mmac−d16 **−mmac−24** **−mrtsc** **−mswape** **−mtelephony** **−mxy** **−misize** **−mannotate−align**
−marclinux **−marclinux_prof** **−mlong−calls** **−mmedium−calls** **−msdata** **−mirq−ctrl−saved**
−mrgf−banked−regs **−mlpc−width=***width* **−G** *num* **−mvolatile−cache** **−mtp−regno=***regno*
−malign−call **−mauto−modify−reg** **−mbbit−peephole** **−mno−brcc** **−mcase−vector−pcrel**
−mcompact−casesi **−mno−cond−exec** **−mearly−cbranchsi** **−mexpand−adddi** **−mindexed−loads**
−mlra **−mlra−priority−none** **−mlra−priority−compact** **−mlra−priority−noncompact**
−mmillicode **−mmixed−code** **−mq−class** **−mRcq** **−mRcw** **−msize−level=***level* **−mtune=***cpu*
−mmultcost=*num* **−mcode−density−frame** **−munalign−prob−threshold=***probability*
−mmpy−option=*multo* **−mdiv−rem** **−mcode−density** **−mll64** **−mfpu=***fpu* **−mrf16**
−mbranch−index

ARM Options (**ARM Options**) **−mapcs−frame** **−mno−apcs−frame** **−mabi=***name*
−mapcs−stack−check **−mno−apcs−stack−check** **−mapcs−reentrant** **−mno−apcs−reentrant**
−mgeneral−regs−only −msched−prolog −mno−sched−prolog −mlittle−endian −mbig−endian
−mbe8 **−mbe32** **−mfloat−abi=***name* **−mfp16−format=***name* **−mthumb−interwork**
−mno−thumb−interwork **−mcpu=***name* **−march=***name* **−mfpu=***name* **−mtune=***name*
−mprint−tune−info **−mstructure−size−boundary=***n* **−mabort−on−noreturn** **−mlong−calls**
−mno−long−calls **−msingle−pic−base** **−mno−single−pic−base** **−mpic−register=***reg*
−mnop−fun−dllimport **−mpoke−function−name** **−mthumb** **−marm** **−mflip−thumb**
−mtpcs−frame **−mtpcs−leaf−frame** **−mcaller−super−interworking**
−mcallee−super−interworking **−mtp=***name* **−mtls−dialect=***dialect* **−mword−relocations**
−mfix−cortex−m3−ldrd **−mfix−cortex−a57−aes−1742098** **−mfix−cortex−a72−aes−1655431**
−munaligned−access **−mneon−for−64bits** **−mslow−flash−data** **−masm−syntax−unified**
−mrestrict−it −mverbose−cost−dump −mpure−code −mcmse −mfix−cmse−cve−2021−35465
−mstack−protector−guard=*guard*   **−mstack−protector−guard−offset=***offset*   **−mfdpic**
−mbranch−protection=*none*|*standard*|*pac−ret*[**+***leaf*] [**+***bti*]|*bti*[**+***pac−ret*[**+***leaf*]]

AVR Options (**AVR Options**) **−mmcu=***mcu* **−mabsdata** **−maccumulate−args** **−mcvt**
−mbranch−cost=*cost* **−mfuse−add=***level* **−mfuse−move=***level* **−mcall−prologues**

−**mgas−isr−prologues** −**mint8** −**mflmap** −**mdouble=***bits* −**mlong−double=***bits* −**mno−call−main** −**mn_flash=***size* −**mfract−convert−truncate** −**mno−interrupts** −**mmain−is−OS_task** −**mrelax** −**mrmw** −**mstrict−X** −**mtiny−stack** −**mrodata−in−ram** −**msplit−bit−shift** −**msplit−ldst** −**mshort−calls** −**mskip−bug** −**muse−nonzero−bits** −**nodevicelib** −**nodevicespecs** −**Waddr−space−convert** −**Wmisspelled−isr**

Blackfin Options (**Blackfin Options**) −**mcpu=***cpu*[−*sirevision*] −**msim** −**momit−leaf−frame−pointer** −**mno−omit−leaf−frame−pointer** −**mspecld−anomaly** −**mno−specld−anomaly** −**mcsync−anomaly** −**mno−csync−anomaly** −**mlow−64k** −**mno−low64k** −**mstack−check−l1** −**mid−shared−library** −**mno−id−shared−library** −**mshared−library−id=***n* −**mleaf−id−shared−library** −**mno−leaf−id−shared−library** −**msep−data** −**mno−sep−data** −**mlong−calls** −**mno−long−calls** −**mfast−fp** −**minline−plt** −**mmulticore** −**mcorea** −**mcoreb** −**msdram** −**micplb**

C6X Options (**C6X Options**) −**mbig−endian** −**mlittle−endian** −**march=***cpu* −**msim** −**msdata=***sdata−type*

CRIS Options (**CRIS Options**) −**mcpu=***cpu* −**march=***cpu* −**mtune=***cpu* −**mmax−stack−frame=***n* −**metrax4** −**metrax100** −**mpdebug** −**mcc−init** −**mno−side−effects** −**mstack−align** −**mdata−align** −**mconst−align** −**m32−bit** −**m16−bit** −**m8−bit** −**mno−prologue−epilogue** −**melf** −**maout** −**sim** −**sim2** −**mmul−bug−workaround** −**mno−mul−bug−workaround**

C−SKY Options (**C−SKY Options**) −**march=***arch* −**mcpu=***cpu* −**mbig−endian** −**EB** −**mlittle−endian** −**EL** −**mhard−float** −**msoft−float** −**mfpu=***fpu* −**mdouble−float** −**mfdivdu** −**mfloat−abi=***name* −**melrw** −**mistack** −**mmp** −**mcp** −**mcache** −**msecurity** −**mtrust** −**mdsp** −**medsp** −**mvdsp** −**mdiv** −**msmart** −**mhigh−registers** −**manchor** −**mpushpop** −**mmultiple−stld** −**mconstpool** −**mstack−size** −**mccrt** −**mbranch−cost=***n* −**mcse−cc** −**msched−prolog** −**msim**

Cygwin and MinGW Options (**Cygwin and MinGW Options**) −**mconsole** −**mcrtdll=***library* −**mdll** −**mnop−fun−dllimport** −**mthread** −**municode** −**mwin32** −**mwindows** −**fno−set−stack−executable**

Darwin Options (**Darwin Options**) −**all_load** −**allowable_client** −**arch** −**arch_errors_fatal** −**arch_only** −**bind_at_load** −**bundle** −**bundle_loader** −**client_name** −**compatibility_version** −**current_version** −**dead_strip** −**dependency−file** −**dylib_file** −**dylinker_install_name** −**dynamic** −**dynamiclib** −**exported_symbols_list** −**filelist** −**flat_namespace** −**force_cpusubtype_ALL** −**force_flat_namespace** −**headerpad_max_install_names** −**iframework** −**image_base** −**init** −**install_name** −**keep_private_externs** −**multi_module** −**multiply_defined** −**multiply_defined_unused** −**noall_load** −**no_dead_strip_inits_and_terms** −**nodefaultrpaths** −**nofixprebinding** −**nomultidefs** −**noprebind** −**noseglinkedit** −**pagezero_size** −**prebind** −**prebind_all_twolevel_modules** −**private_bundle** −**read_only_relocs** −**sectalign** −**sectobjectsymbols** −**whyload** −**seg1addr** −**sectcreate** −**sectobjectsymbols** −**sectorder** −**segaddr** −**segs_read_only_addr** −**segs_read_write_addr** −**seg_addr_table** −**seg_addr_table_filename** −**seglinkedit** −**segprot** −**segs_read_only_addr** −**segs_read_write_addr** −**single_module** −**static** −**sub_library** −**sub_umbrella** −**twolevel_namespace** −**umbrella** −**undefined** −**unexported_symbols_list** −**weak_reference_mismatches** −**whatsloaded** −**F** −**gused** −**gfull** −**mmacosx−version−min=***version* −**mkernel** −**mone−byte−bool**

DEC Alpha Options (**DEC Alpha Options**) −**mno−fp−regs** −**msoft−float** −**mieee** −**mieee−with−inexact** −**mieee−conformant** −**mfp−trap−mode=***mode* −**mfp−rounding−mode=***mode* −**mtrap−precision=***mode* −**mbuild−constants** −**mcpu=***cpu−type* −**mtune=***cpu−type* −**mbwx** −**mmax** −**mfix** −**mcix** −**msafe−bwa** −**msafe−partial** −**mfloat−vax** −**mfloat−ieee** −**mexplicit−relocs** −**msmall−data** −**mlarge−data** −**msmall−text** −**mlarge−text** −**mmemory−latency=***time*

eBPF Options (**eBPF Options**) −**mbig−endian** −**mlittle−endian** −**mframe−limit=***bytes* −**mxbpf** −**mco−re** −**mno−co−re** −**mjmpext** −**mjmp32** −**malu32** −**mv3−atomics** −**mbswap** −**msdiv** −**msmov** −**mcpu=***version* −**masm=***dialect* −**minline−memops−threshold=***bytes*

FR30 Options (**FR30 Options**) **−msmall−model −mno−lsim**

FT32 Options (**FT32 Options**) **−msim −mlra −mnodiv −mft32b −mcompress −mnopm**

FRV Options (**FRV Options**) **−mgpr−32 −mgpr−64 −mfpr−32 −mfpr−64 −mhard−float −msoft−float −malloc−cc −mfixed−cc −mdword −mno−dword −mdouble −mno−double −mmedia −mno−media −mmuladd −mno−muladd −mfdpic −minline−plt −mgprel−ro −multilib−library−pic −mlinked−fp −mlong−calls −malign−labels −mlibrary−pic −macc−4 −macc−8 −mpack −mno−pack −mno−eflags −mcond−move −mno−cond−move −moptimize−membar −mno−optimize−membar −mscc −mno−scc −mcond−exec −mno−cond−exec −mvliw−branch −mno−vliw−branch −mmulti−cond−exec −mno−multi−cond−exec −mnested−cond−exec −mno−nested−cond−exec −mtomcat−stats −mTLS −mtls −mcpu=***cpu*

GNU/Linux Options (**GNU/Linux Options**) **−mglibc −muclibc −mmusl −mbionic −mandroid −tno−android−cc −tno−android−ld**

H8/300 Options (**H8/300 Options**) **−mrelax −mh −ms −mn −mexr −mno−exr −mint32 −malign−300**

HPPA Options (**HPPA Options**) **−march=***architecture−type* **−matomic−libcalls −mbig−switch −mcaller−copies −mdisable−fpregs −mdisable−indexing −mordered −mfast−indirect−calls −mgas −mgnu−ld −mhp−ld −mfixed−range=***register−range* **−mcoherent−ldcw −mjump−in−delay −mlinker−opt −mlong−calls −mlong−load−store −mno−atomic−libcalls −mno−disable−fpregs −mno−disable−indexing −mno−fast−indirect−calls −mno−gas −mno−jump−in−delay −mno−long−load−store −mno−portable−runtime −mno−soft−float −mno−space−regs −msoft−float −mpa−risc−1−0 −mpa−risc−1−1 −mpa−risc−2−0 −mportable−runtime −mschedule=***cpu−type* **−mspace−regs −msoft−mult −msio −mwsio −munix=***unix−std* **−nolibdld −static −threads**

IA−64 Options (**IA−64 Options**) **−mbig−endian −mlittle−endian −mgnu−as −mgnu−ld −mno−pic −mvolatile−asm−stop −mregister−names −msdata −mno−sdata −mconstant−gp −mauto−pic −mfused−madd −minline−float−divide−min−latency −minline−float−divide−max−throughput −mno−inline−float−divide −minline−int−divide−min−latency −minline−int−divide−max−throughput −mno−inline−int−divide −minline−sqrt−min−latency −minline−sqrt−max−throughput −mno−inline−sqrt −mdwarf2−asm −mearly−stop−bits −mfixed−range=***register−range* **−mtls−size=***tls−size* **−mtune=***cpu−type* **−milp32 −mlp64 −msched−br−data−spec −msched−ar−data−spec −msched−control−spec −msched−br−in−data−spec −msched−ar−in−data−spec −msched−in−control−spec −msched−spec−ldc −msched−spec−control−ldc −msched−prefer−non−data−spec−insns −msched−prefer−non−control−spec−insns −msched−stop−bits−after−every−cycle −msched−count−spec−in−critical−path −msel−sched−dont−check−control−spec −msched−fp−mem−deps−zero−cost −msched−max−memory−insns−hard−limit −msched−max−memory−insns=***max−insns*

LM32 Options (**LM32 Options**) **−mbarrel−shift−enabled −mdivide−enabled −mmultiply−enabled −msign−extend−enabled −muser−enabled**

LoongArch Options (**LoongArch Options**) **−march=***arch−type* **−mtune=***tune−type* **−mabi=***base−abi−type* **−mfpu=***fpu−type* **−msimd=***simd−type* **−msoft−float −msingle−float −mdouble−float −mlsx −mno−lsx −mlasx −mno−lasx −mbranch−cost=***n* **−maddr−reg−reg−cost=***n* **−mcheck−zero−division −mno−check−zero−division −mcond−move−int −mno−cond−move−int −mcond−move−float −mno−cond−move−float −mmemcpy −mno−memcpy −mstrict−align −mno−strict−align −G** *num* **−mmax−inline−memcpy−size=***n* **−mexplicit−relocs=***style* **−mexplicit−relocs −mno−explicit−relocs −mdirect−extern−access −mno−direct−extern−access −mcmodel=***code−model* **−mrelax −mpass−mrelax−to−as −mrecip −mrecip=***opt* **−mfrecipe −mno−frecipe −mdiv32 −mno−div32 −mlam−bh −mno−lam−bh −mlamcas −mno−lamcas**

−mld−seq−sa   **−mno−ld−seq−sa**   **−mtls−dialect=***opt*   **−mannotate−tablejump**
−mno−annotate−tablejump

M32C Options (**M32C Options**) **−mcpu=***cpu* **−msim** **−memregs=***number*

M32R/D Options (**M32R/D Options**) **−m32r2** **−m32rx** **−m32r** **−mdebug** **−malign−loops**
−mno−align−loops   **−missue−rate=***number*   **−mbranch−cost=***number*
−mmodel=*code−size−model−type* **−msdata=***sdata−type* **−mno−flush−func** **−mflush−func=***name*
−mno−flush−trap **−mflush−trap=***number* **−G** *num*

M680x0 Options (**M680x0 Options**) **−march=***arch* **−mcpu=***cpu* **−mtune=***tune* **−m68000** **−m68020**
−m68020−40 **−m68020−60** **−m68030** **−m68040** **−m68060** **−mcpu32** **−m5200** **−m5206e** **−m528x**
−m5307 **−m5407** **−mcfv4e** **−mbitfield** **−mno−bitfield** **−mc68000** **−mc68020** **−mnobitfield**
−mrtd **−mno−rtd** **−mdiv** **−mno−div** **−mshort** **−mno−short** **−mhard−float** **−m68881**
−msoft−float **−mpcrel** **−malign−int** **−mstrict−align** **−msep−data** **−mno−sep−data**
−mshared−library−id=n **−mid−shared−library** **−mno−id−shared−library** **−mxgot** **−mno−xgot**
−mlong−jump−table−offsets

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

MicroBlaze Options (**MicroBlaze Options**) **−msoft−float** **−mhard−float** **−msmall−divides**
−mcpu=*cpu*   **−mmemcpy**   **−mxl−soft−mul**   **−mxl−soft−div**   **−mxl−barrel−shift**
−mxl−pattern−compare **−mxl−stack−check** **−mxl−gp−opt** **−mno−clearbss** **−mxl−multiply−high**
−mxl−float−convert **−mxl−float−sqrt** **−mbig−endian** **−mlittle−endian** **−mxl−reorder**
−mxl−mode−*app−model* **−mpic−data−is−text−relative**

MIPS Options (**MIPS Options**) **−EL** **−EB** **−march=***arch* **−mtune=***arch* **−mips1** **−mips2** **−mips3**
−mips4 **−mips32** **−mips32r2** **−mips32r3** **−mips32r5** **−mips32r6** **−mips64** **−mips64r2**
−mips64r3 **−mips64r5** **−mips64r6** **−mips16** **−mno−mips16** **−mflip−mips16**
−minterlink−compressed   **−mno−interlink−compressed**   **−minterlink−mips16**
−mno−interlink−mips16 **−mabi=***abi* **−mabicalls** **−mno−abicalls** **−mshared** **−mno−shared** **−mplt**
−mno−plt **−mxgot** **−mno−xgot** **−mgp32** **−mgp64** **−mfp32** **−mfpxx** **−mfp64** **−mhard−float**
−msoft−float **−mno−float** **−msingle−float** **−mdouble−float** **−modd−spreg** **−mno−odd−spreg**
−mabs=*mode* **−mnan=***encoding* **−mdsp** **−mno−dsp** **−mdspr2** **−mno−dspr2** **−mmcu**
−mmno−mcu **−meva** **−mno−eva** **−mvirt** **−mno−virt** **−mxpa** **−mno−xpa** **−mcrc** **−mno−crc**
−mginv **−mno−ginv** **−mmicromips** **−mno−micromips** **−mmsa** **−mno−msa** **−mloongson−mmi**
−mno−loongson−mmi **−mloongson−ext** **−mno−loongson−ext** **−mloongson−ext2**
−mno−loongson−ext2 **−mfpu=***fpu−type* **−msmartmips** **−mno−smartmips** **−mpaired−single**
−mno−paired−single **−mdmx** **−mno−mdmx** **−mips3d** **−mno−mips3d** **−mmt** **−mno−mt** **−mllsc**
−mno−llsc **−mlong64** **−mlong32** **−msym32** **−mno−sym32** **−G***num* **−mlocal−sdata**
−mno−local−sdata **−mextern−sdata** **−mno−extern−sdata** **−mgpopt** **−mno−gopt**
−membedded−data **−mno−embedded−data** **−muninit−const−in−rodata**
−mno−uninit−const−in−rodata **−mcode−readable=***setting* **−msplit−addresses**
−mno−split−addresses **−mexplicit−relocs** **−mno−explicit−relocs** **−mexplicit−relocs=***release*
−mcheck−zero−division **−mno−check−zero−division** **−mdivide−traps** **−mdivide−breaks**
−mload−store−pairs **−mno−load−store−pairs** **−mstrict−align** **−mno−strict−align**
−mno−unaligned−access **−munaligned−access** **−mmemcpy** **−mno−memcpy** **−mlong−calls**
−mno−long−calls **−mmad** **−mno−mad** **−mimadd** **−mno−imadd** **−mfused−madd**
−mno−fused−madd **−nocpp** **−mfix−24k** **−mno−fix−24k** **−mfix−r4000** **−mno−fix−r4000**
−mfix−r4400 **−mno−fix−r4400** **−mfix−r5900** **−mno−fix−r5900** **−mfix−r10000** **−mno−fix−r10000**
−mfix−rm7000 **−mno−fix−rm7000** **−mfix−vr4120** **−mno−fix−vr4120** **−mfix−vr4130**
−mno−fix−vr4130 **−mfix−sb1** **−mno−fix−sb1** **−mflush−func=***func* **−mno−flush−func**
−mbranch−cost=*num* **−mbranch−likely** **−mno−branch−likely** **−mcompact−branches=***policy*

−mfp−exceptions −mno−fp−exceptions −mvr4130−align −mno−vr4130−align −msynci
−mno−synci −mlxc1−sxc1 −mno−lxc1−sxc1 −mmadd4 −mno−madd4 −mrelax−pic−calls
−mno−relax−pic−calls −mmcount−ra−address −mframe−header−opt −mno−frame−header−opt

MMIX Options (**MMIX Options**) −mlibfuncs −mno−libfuncs −mepsilon −mno−epsilon
−mabi=gnu −mabi=mmixware −mzero−extend −mknuthdiv −mtoplevel−symbols −melf
−mbranch−predict −mno−branch−predict −mbase−addresses −mno−base−addresses
−msingle−exit −mno−single−exit

MN10300 Options (**MN10300 Options**) −mmult−bug −mno−mult−bug −mno−am33 −mam33
−mam33−2 −mam34 −mtune=*cpu−type* −mreturn−pointer−on−d0 −mno−crt0 −mrelax −mliw
−msetlb

Moxie Options (**Moxie Options**) −meb −mel −mmul.x −mno−crt0

MSP430 Options (**MSP430 Options**) −msim −masm−hex −mmcu= −mcpu= −mlarge −msmall
−mrelax −mwarn−mcu −mcode−region= −mdata−region= −msilicon−errata=
−msilicon−errata−warn= −mhwmult= −minrt −mtiny−printf −mmax−inline−shift=

NDS32 Options (**NDS32 Options**) −mbig−endian −mlittle−endian −mreduced−regs −mfull−regs
−mcmov −mno−cmov −mext−perf −mno−ext−perf −mext−perf2 −mno−ext−perf2
−mext−string −mno−ext−string −mv3push −mno−v3push −m16bit −mno−16bit
−misr−vector−size=*num* −mcache−block−size=*num* −march=*arch* −mcmodel=*code−model*
−mctor−dtor −mrelax

Nvidia PTX Options (**Nvidia PTX Options**) −m64 −mmainkernel −moptimize

OpenRISC Options (**OpenRISC Options**) −mboard=*name* −mnewlib −mhard−mul −mhard−div
−msoft−mul −msoft−div −msoft−float −mhard−float −mdouble−float −munordered−float
−mcmov −mror −mrori −msext −msfimm −mshftimm −mcmodel=*code−model*

PDP−11 Options (**PDP−11 Options**) −mfpu −msoft−float −mac0 −mno−ac0 −m40 −m45 −m10
−mint32 −mno−int16 −mint16 −mno−int32 −msplit −munix−asm −mdec−asm −mgnu−asm
−mlra

PowerPC Options See RS/6000 and PowerPC Options.

PRU Options (**PRU Options**) −mmcu=*mcu* −minrt −mno−relax −mloop −mabi=*variant*

RISC−V Options (**RISC−V Options**) −mbranch−cost=*N−instruction* −mplt −mno−plt
−mabi=*ABI−string* −mfdiv −mno−fdiv −mfence−tso −mno−fence−tso −mdiv −mno−div
−misa−spec=*ISA−spec−string* −march=*ISA−string* −mtune=*processor−string*
−mpreferred−stack−boundary=*num* −msmall−data−limit=*N−bytes* −msave−restore
−mno−save−restore −mshorten−memrefs −mno−shorten−memrefs −mstrict−align
−mno−strict−align −mcmodel=medlow −mcmodel=medany −mcmodel=large −mexplicit−relocs
−mno−explicit−relocs −mrelax −mno−relax −mriscv−attribute −mno−riscv−attribute
−malign−data=*type* −mbig−endian −mlittle−endian −mstack−protector−guard=*guard*
−mstack−protector−guard−reg=*reg* −mstack−protector−guard−offset=*offset* −mcsr−check
−mno−csr−check −mmovcc −mno−movcc −minline−atomics −mno−inline−atomics
−minline−strlen −mno−inline−strlen −minline−strcmp −mno−inline−strcmp −minline−strncmp
−mno−inline−strncmp −mtls−dialect=desc −mtls−dialect=trad

RL78 Options (**RL78 Options**) −msim −mmul=none −mmul=g13 −mmul=g14 −mallregs
−mcpu=g10 −mcpu=g13 −mcpu=g14 −mg10 −mg13 −mg14 −m64bit−doubles
−m32bit−doubles −msave−mduc−in−interrupts

RS/6000 and PowerPC Options (**RS/6000 and PowerPC Options**) −mcpu=*cpu−type*
−mtune=*cpu−type* −mcmodel=*code−model* −mpowerpc64 −maltivec −mno−altivec
−mpowerpc−gpopt −mno−powerpc−gpopt −mpowerpc−gfxopt −mno−powerpc−gfxopt
−mmfcrf −mno−mfcrf −mpopcntb −mno−popcntb −mpopcntd −mno−popcntd −mfprnd
−mno−fprnd −mcmpb −mno−cmpb −mhard−dfp −mno−hard−dfp −mfull−toc

−mminimal−toc   −mno−fp−in−toc   −mno−sum−in−toc   −m64   −m32   −mxl−compat −mno−xl−compat   −mpe   −malign−power   −malign−natural   −msoft−float   −mhard−float −mmultiple   −mno−multiple   −mupdate   −mno−update   −mavoid−indexed−addresses −mno−avoid−indexed−addresses   −mfused−madd   −mno−fused−madd   −mbit−align −mno−bit−align   −mstrict−align   −mno−strict−align   −mrelocatable   −mno−relocatable −mrelocatable−lib −mno−relocatable−lib −mtoc −mno−toc −mlittle −mlittle−endian −mbig −mbig−endian   −mdynamic−no−pic   −mswdiv   −msingle−pic−base −mprioritize−restricted−insns=*priority*   −msched−costly−dep=*dependence_type* −minsert−sched−nops=*scheme* −mcall−aixdesc −mcall−eabi −mcall−freebsd −mcall−linux −mcall−netbsd   −mcall−openbsd   −mcall−sysv   −mcall−sysv−eabi   −mcall−sysv−noeabi −mtraceback=*traceback_type*   −maix−struct−return   −msvr4−struct−return   −mabi=*abi−type* −msecure−plt   −mbss−plt   −msplit−patch−nops   −mlongcall   −mno−longcall   −mpltseq −mno−pltseq   −mblock−move−inline−limit=*num*   −mblock−compare−inline−limit=*num* −mblock−compare−inline−loop−limit=*num*   −mno−block−ops−unaligned−vsx −mstring−compare−inline−limit=*num* −misel −mno−isel −mvrsave −mno−vrsave −mmulhw −mno−mulhw −mdlmzb −mno−dlmzb −mprototype −mno−prototype −msim −mmvme −mads −myellowknife −memb −msdata −msdata=*opt* −mreadonly−in−sdata −mvxworks −**G** *num* −mrecip −mrecip=*opt* −mno−recip −mrecip−precision −mno−recip−precision −mveclibabi=*type* −mfriz   −mno−friz   −mpointers−to−nested−functions   −mno−pointers−to−nested−functions −msave−toc−indirect   −mno−save−toc−indirect   −mpower8−fusion   −mno−mpower8−fusion −mcrypto   −mno−crypto   −mhtm   −mno−htm   −mquad−memory   −mno−quad−memory −mquad−memory−atomic   −mno−quad−memory−atomic   −mcompat−align−parm −mno−compat−align−parm   −mfloat128   −mno−float128   −mfloat128−hardware −mno−float128−hardware   −mgnu−attribute   −mno−gnu−attribute −mstack−protector−guard=*guard*   −mstack−protector−guard−reg=*reg* −mstack−protector−guard−offset=*offset* −mprefixed −mno−prefixed −mpcrel −mno−pcrel −mmma −mno−mmma −mrop−protect −mno−rop−protect −mprivileged −mno−privileged

RX Options (**RX Options**) −m64bit−doubles −m32bit−doubles −fpu −nofpu −mcpu= −mbig−endian−data −mlittle−endian−data −msmall−data −msim −mno−sim −mas100−syntax −mno−as100−syntax   −mrelax   −mmax−constant−size=   −mint−register=   −mpid −mallow−string−insns −mno−allow−string−insns −mjsr −mno−warn−multiple−fast−interrupts −msave−acc−in−interrupts

S/390 and zSeries Options (**S/390 and zSeries Options**) −mtune=*cpu−type* −**march**=*cpu−type* −mhard−float   −msoft−float   −mhard−dfp   −mno−hard−dfp   −mlong−double−64 −mlong−double−128 −mbackchain −mno−backchain −mpacked−stack −mno−packed−stack −msmall−exec −mno−small−exec −mmvcle −mno−mvcle −m64 −m31 −mdebug −mno−debug −mesa −mzarch −mhtm −mvx −mzvector −mtpf−trace −mno−tpf−trace −mtpf−trace−skip −mno−tpf−trace−skip   −mfused−madd   −mno−fused−madd   −mwarn−framesize −mwarn−dynamicstack −mstack−size −mstack−guard −mhotpatch=*halfwords*,*halfwords*

SH Options (**SH Options**) −m1 −m2 −m2e −m2a−nofpu −m2a−single−only −m2a−single −m2a −m3 −m3e −m4−nofpu −m4−single−only −m4−single −m4 −m4a−nofpu −m4a−single−only −m4a−single −m4a −m4al −mb −ml −mdalign −mrelax −mbigtable −mfmovd −mrenesas −mno−renesas −mnomacsave −mieee −mno−ieee −mbitops −misize −minline−ic_invalidate −mpadstruct   −mprefergot   −musermode   −multcost=*number*   −mdiv=*strategy* −mdivsi3_libfunc=*name*   −mfixed−range=*register−range*   −maccumulate−outgoing−args −matomic−model=*atomic−model*   −mbranch−cost=*num*   −mzdcbranch   −mno−zdcbranch −mcbranch−force−delay−slot −mfused−madd −mno−fused−madd −mfsca −mno−fsca −mfsrra −mno−fsrra −mpretend−cmove −mtas

Solaris 2 Options (**Solaris 2 Options**) −mclear−hwcap −mno−clear−hwcap −mimpure−text −mno−impure−text −pthreads

SPARC Options (**SPARC Options**) −mcpu=*cpu−type* −**mtune**=*cpu−type* −**mcmodel**=*code−model* −**mmemory−model**=*mem−model* −**m32** −**m64** −**mapp−regs** −**mno−app−regs** −**mfaster−structs**

−mno−faster−structs −mflat −mno−flat −mfpu −mno−fpu −mhard−float −msoft−float −mhard−quad−float −msoft−quad−float −mstack−bias −mno−stack−bias −mstd−struct−return −mno−std−struct−return −munaligned−doubles −mno−unaligned−doubles −muser−mode −mno−user−mode −mv8plus −mno−v8plus −mvis −mno−vis −mvis2 −mno−vis2 −mvis3 −mno−vis3 −mvis3b −mno−vis3b −mvis4 −mno−vis4 −mvis4b −mno−vis4b −mcbcond −mno−cbcond −mfmaf −mno−fmaf −mfsmuld −mno−fsmuld −mpopc −mno−popc −msubxc −mno−subxc −mfix−at697f −mfix−ut699 −mfix−ut700 −mfix−gr712rc

System V Options (**System V Options**) −**Qy** −**Qn** −**YP,***paths* −**Ym,***dir*

V850 Options (**V850 Options**) −**mlong−calls** −**mno−long−calls** −**mep** −**mno−ep** −**mprolog−function** −**mno−prolog−function** −**mspace** −**mtda=***n* −**msda=***n* −**mzda=***n* −**mapp−regs** −**mno−app−regs** −**mdisable−callt** −**mno−disable−callt** −**mv850e2v3** −**mv850e2** −**mv850e1** −**mv850es** −**mv850e** −**mv850** −**mv850e3v5** −**mloop** −**mrelax** −**mlong−jumps** −**msoft−float** −**mhard−float** −**mgcc−abi** −**mrh850−abi** −**mbig−switch**

VAX Options (**VAX Options**) −**munix** −**mgnu** −**md** −**md−float** −**mg** −**mg−float** −**mlra**

Visium Options (**Visium Options**) −**mdebug** −**msim** −**mfpu** −**mno−fpu** −**mhard−float** −**msoft−float** −**mcpu=***cpu−type* −**mtune=***cpu−type* −**msv−mode** −**muser−mode**

VMS Options (**VMS Options**) −**mvms−return−codes** −**mdebug−main=***prefix* −**mmalloc64** −**mpointer−size=***size*

VxWorks Options (**VxWorks Options**) −**mrtp** −**msmp** −**non−static** −**Bstatic** −**Bdynamic** −**Xbind−lazy** −**Xbind−now**

x86 Options (**x86 Options**) −**mtune=***cpu−type* −**march=***cpu−type* −**mtune−ctrl=***feature−list* −**mdump−tune−features** −**mno−default** −**mfpmath=***unit* −**masm=***dialect* −**mno−fancy−math−387** −**mno−fp−ret−in−387** −**m80387** −**mhard−float** −**msoft−float** −**mno−wide−multiply** −**mrtd** −**malign−double** −**mpreferred−stack−boundary=***num* −**mincoming−stack−boundary=***num* −**mcld** −**mcx16** −**msahf** −**mmovbe** −**mcrc32** −**mmwait** −**mrecip** −**mrecip=***opt* −**mvzeroupper** −**mprefer−avx128** −**mprefer−vector−width=***opt* −**mpartial−vector−fp−math** −**mmove−max=***bits* −**mstore−max=***bits* −**mnoreturn−no−callee−saved−registers** −**mmmx** −**msse** −**msse2** −**msse3** −**mssse3** −**msse4.1** −**msse4.2** −**msse4** −**mavx** −**mavx2** −**mavx512f** −**mavx512cd** −**mavx512vl** −**mavx512bw** −**mavx512dq** −**mavx512ifma** −**mavx512vbmi** −**msha** −**maes** −**mpclmul** −**mfsgsbase** −**mrdrnd** −**mf16c** −**mfma** −**mpconfig** −**mwbnoinvd** −**mptwrite** −**mclflushopt** −**mclwb** −**mxsavec** −**mxsaves** −**msse4a** −**m3dnow** −**m3dnowa** −**mpopcnt** −**mabm** −**mbmi** −**mtbm** −**mfma4** −**mxop** −**madx** −**mlzcnt** −**mbmi2** −**mfxsr** −**mxsave** −**mxsaveopt** −**mrtm** −**mhle** −**mlwp** −**mmwaitx** −**mclzero** −**mpku** −**mthreads** −**mgfni** −**mvaes** −**mwaitpkg** −**mshstk** −**mmanual−endbr** −**mcet−switch** −**mforce−indirect−call** −**mavx512vbmi2** −**mavx512bf16** −**menqcmd** −**mvpclmulqdq** −**mavx512bitalg** −**mmovdiri** −**mmovdir64b** −**mavx512vpopcntdq** −**mavx512vnni** −**mprfchw** −**mrdpid** −**mrdseed** −**msgx** −**mavx512vp2intersect** −**mserialize** −**mtsxldtrk** −**mamx−tile** −**mamx−int8** −**mamx−bf16** −**muintr** −**mhreset** −**mavxvnni** −**mamx−fp8** −**mavx512fp16** −**mavxifma** −**mavxvnniint8** −**mavxneconvert** −**mcmpccxadd** −**mamx−fp16** −**mprefetchi** −**mraoint** −**mamx−complex** −**mavxvnniint16** −**msm3** −**msha512** −**msm4** −**mapxf** −**musermsr** −**mavx10.1** −**mavx10.1−256** −**mavx10.1−512** −**mevex512** −**mavx10.2** −**mamx−avx512** −**mamx−tf32** −**mamx−transpose** −**mmovrs** −**mamx−movrs** −**mcldemote** −**mms−bitfields** −**mno−align−stringops** −**minline−all−stringops** −**minline−stringops−dynamically** −**mstringop−strategy=***alg* −**mkl** −**mwidekl** −**mmemcpy−strategy=***strategy* −**mmemset−strategy=***strategy* −**mpush−args** −**maccumulate−outgoing−args** −**m128bit−long−double** −**m96bit−long−double** −**mlong−double−64** −**mlong−double−80** −**mlong−double−128** −**mregparm=***num* −**msseregparm** −**mveclibabi=***type* −**mvect8−ret−in−mem** −**mpc32** −**mpc64** −**mpc80** −**mdaz−ftz** −**mstackrealign** −**momit−leaf−frame−pointer** −**mno−red−zone** −**mno−tls−direct−seg−refs** −**mcmodel=***code−model* −**mabi=***name* −**maddress−mode=***mode* −**m32** −**m64** −**mx32** −**m16** −**miamcu** −**mlarge−data−threshold=***num* −**msse2avx** −**mfentry** −**mrecord−mcount** −**mnop−mcount** −**m8bit−idiv** −**minstrument−return=***type* −**mfentry−name=***name* −**mfentry−section=***name*

−mavx256−split−unaligned−load **−mavx256−split−unaligned−store** **−malign−data=***type*
−mstack−protector−guard=*guard* **−mstack−protector−guard−reg=***reg*
−mstack−protector−guard−offset=*offset* **−mstack−protector−guard−symbol=***symbol*
−mgeneral−regs−only **−mcall−ms2sysv−xlogues** **−mrelax−cmpxchg−loop**
−mindirect−branch=*choice* **−mfunction−return=***choice* **−mindirect−branch−register**
−mharden−sls=*choice* **−mindirect−branch−cs−prefix** **−mneeded** **−mno−direct−extern−access**
−munroll−only−small−loops −mlam=*choice*

x86 Windows Options See Cygwin and MinGW Options.

Xstormy16 Options (**Xstormy16 Options**) **−msim**

Xtensa Options (**Xtensa Options**) **−mconst16 −mno−const16 −mfused−madd −mno−fused−madd**
−mforce−no−pic **−mserialize−volatile** **−mno−serialize−volatile** **−mtext−section−literals**
−mno−text−section−literals **−mauto−litpools** **−mno−auto−litpools** **−mtarget−align**
−mno−target−align −mlongcalls −mno−longcalls −mabi=*abi−type* **−mextra−l32r−costs=***cycles*
−mstrict−align −mno−strict−align

zSeries Options See S/390 and zSeries Options.

Options Controlling the Kind of Output

Compilation can involve up to four stages: preprocessing, compilation proper, assembly and linking, always in that order. GCC is capable of preprocessing and compiling several files either into several assembler input files, or into one assembler input file; then each assembler input file produces an object file, and linking combines all the object files (those newly compiled, and those specified as input) into an executable file.

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

*file***.c**
> C source code that must be preprocessed.

*file***.i**
> C source code that should not be preprocessed.

*file***.ii**
> C++ source code that should not be preprocessed.

*file***.m**
> Objective−C source code. Note that you must link with the *libobjc* library to make an Objective−C program work.

*file***.mi**
> Objective−C source code that should not be preprocessed.

*file***.mm**
*file***.M**
> Objective−C++ source code. Note that you must link with the *libobjc* library to make an Objective−C++ program work. Note that **.M** refers to a literal capital M.

*file***.mii**
> Objective−C++ source code that should not be preprocessed.

*file***.h**
> C, C++, Objective−C or Objective−C++ header file to be turned into a precompiled header (default), or C, C++ header file to be turned into an Ada spec (via the **−fdump−ada−spec** switch).

*file***.cc**
*file***.cp**
*file***.cxx**
*file***.cpp**

file.**CPP**
file.**c++**
file.**C**

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

file.**mm**
file.**M**

> Objective−C++ source code that must be preprocessed.

file.**mii**

> Objective−C++ source code that should not be preprocessed.

file.**hh**
file.**H**
file.**hp**
file.**hxx**
file.**hpp**
file.**HPP**
file.**h++**
file.**tcc**

> C++ header file to be turned into a precompiled header or Ada spec.

file.**f**
file.**for**
file.**ftn**
file.**fi**

> Fixed form Fortran source code that should not be preprocessed.

file.**F**
file.**FOR**
file.**fpp**
file.**FPP**
file.**FTN**

> Fixed form Fortran source code that must be preprocessed (with the traditional preprocessor).

file.**f90**
file.**f95**
file.**f03**
file.**f08**
file.**fii**

> Free form Fortran source code that should not be preprocessed.

file.**F90**
file.**F95**
file.**F03**
file.**F08**

> Free form Fortran source code that must be preprocessed (with the traditional preprocessor).

file.**cob**
file.**COB**
file.**cbl**
file.**CBL**

> COBOL source code.

file.**go**

> Go source code.

file.**d**
      D source code.

file.**di**
      D interface file.

file.**dd**
      D documentation code (Ddoc).

file.**ads**
      Ada source code file that contains a library unit declaration (a declaration of a package, subprogram, or generic, or a generic instantiation), or a library unit renaming declaration (a package, generic, or subprogram renaming declaration). Such files are also called *specs*.

file.**adb**
      Ada source code file containing a library unit body (a subprogram or package body). Such files are also called *bodies*.

file.**s**
      Assembler code.

file.**S**
file.**sx**
      Assembler code that must be preprocessed.

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

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

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

```
c  c-header  cpp-output
c++  c++-header  c++-system-header c++-user-header c++-cpp-output
objective-c  objective-c-header  objective-c-cpp-output
objective-c++ objective-c++-header objective-c++-cpp-output
assembler  assembler-with-cpp
ada
cobol
d
f77  f77-cpp-input f95  f95-cpp-input
go
```

      Note that **−x** does not imply a particular language standard. For example **−x f77** may also require **−std=legacy** for some older source codes.

−x none
      Turn off any specification of a language, so that subsequent files are handled according to their file name suffixes (as if **−x** has not been used at all).

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

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

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

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

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

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

Input files that don't require compilation are ignored.

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

Input files that don't require preprocessing are ignored.

−o *file*

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

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

Though **−o** names only the primary output, it also affects the naming of auxiliary and dump outputs. See the examples below. Unless overridden, both auxiliary outputs and dump outputs are placed in the same directory as the primary output. In auxiliary outputs, the suffix of the input file is replaced with that of the auxiliary output file type; in dump outputs, the suffix of the dump file is appended to the input file suffix. In compilation commands, the base name of both auxiliary and dump outputs is that of the primary output; in compile and link commands, the primary output name, minus the executable suffix, is combined with the input file name. If both share the same base name, disregarding the suffix, the result of the combination is that base name, otherwise, they are concatenated, separated by a dash.

            gcc −c foo.c ...

will use *foo.o* as the primary output, and place aux outputs and dumps next to it, e.g., aux file *foo.dwo* for **−gsplit−dwarf**, and dump file *foo.c.???r.final* for **−fdump−rtl−final**.

If a non−linker output file is explicitly specified, aux and dump files by default take the same base name:

            gcc −c foo.c −o dir/foobar.o ...

will name aux outputs *dir/foobar.** and dump outputs *dir/foobar.c.**.

A linker output will instead prefix aux and dump outputs:

            gcc foo.c bar.c −o dir/foobar ...

will generally name aux outputs *dir/foobar−foo.** and *dir/foobar−bar.**, and dump outputs *dir/foobar−foo.c.** and *dir/foobar−bar.c.**.

The one exception to the above is when the executable shares the base name with the single input:

            gcc foo.c −o dir/foo ...

in which case aux outputs are named *dir/foo.** and dump outputs named *dir/foo.c.**.

The location and the names of auxiliary and dump outputs can be adjusted by the options **−dumpbase**, **−dumpbase−ext**, **−dumpdir**, **−save−temps=cwd**, and **−save−temps=obj**.

−dumpbase *dumpbase*

This option sets the base name for auxiliary and dump output files. It does not affect the name of the primary output file. Intermediate outputs, when preserved, are not regarded as primary outputs, but as auxiliary outputs:

            gcc −save-temps −S foo.c

saves the (no longer) temporary preprocessed file in *foo.i*, and then compiles to the (implied) output file *foo.s*, whereas:

```
              gcc -save-temps -dumpbase save-foo -c foo.c
```

preprocesses to in *save−foo.i*, compiles to *save−foo.s* (now an intermediate, thus auxiliary output), and then assembles to the (implied) output file *foo.o*.

Absent this option, dump and aux files take their names from the input file, or from the (non−linker) output file, if one is explicitly specified: dump output files (e.g. those requested by **−fdump−*** options) with the input name suffix, and aux output files (those requested by other non−dump options, e.g. -save-temps, -gsplit-dwarf, -fcallgraph-info) without it.

Similar suffix differentiation of dump and aux outputs can be attained for explicitly−given **−dumpbase basename.suf** by also specifying **−dumpbase−ext .suf**.

If *dumpbase* is explicitly specified with any directory component, any *dumppfx* specification (e.g. **−dumpdir** or **−save−temps=***) is ignored, and instead of appending to it, *dumpbase* fully overrides it:

```
              gcc foo.c -c -o dir/foo.o -dumpbase alt/foo \
                -dumpdir pfx- -save-temps=cwd ...
```

creates auxiliary and dump outputs named *alt/foo.**, disregarding *dir/* in **−o**, the *./* prefix implied by **−save−temps=cwd**, and *pfx−* in **−dumpdir**.

When **−dumpbase** is specified in a command that compiles multiple inputs, or that compiles and then links, it may be combined with *dumppfx*, as specified under **−dumpdir**. Then, each input file is compiled using the combined *dumppfx*, and default values for *dumpbase* and *auxdropsuf* are computed for each input file:

```
              gcc foo.c bar.c -c -dumpbase main ...
```

creates *foo.o* and *bar.o* as primary outputs, and avoids overwriting the auxiliary and dump outputs by using the *dumpbase* as a prefix, creating auxiliary and dump outputs named *main−foo.** and *main−bar.**.

An empty string specified as *dumpbase* avoids the influence of the output basename in the naming of auxiliary and dump outputs during compilation, computing default values :

```
              gcc -c foo.c -o dir/foobar.o -dumpbase " ...
```

will name aux outputs *dir/foo.** and dump outputs *dir/foo.c.**. Note how their basenames are taken from the input name, but the directory still defaults to that of the output.

The empty−string dumpbase does not prevent the use of the output basename for outputs during linking:

```
              gcc foo.c bar.c -o dir/foobar -dumpbase " -flto ...
```

The compilation of the source files will name auxiliary outputs *dir/foo.** and *dir/bar.**, and dump outputs *dir/foo.c.** and *dir/bar.c.**. LTO recompilation during linking will use *dir/foobar.* as the prefix for dumps and auxiliary files.

−dumpbase−ext *auxdropsuf*
    When forming the name of an auxiliary (but not a dump) output file, drop trailing *auxdropsuf* from *dumpbase* before appending any suffixes. If not specified, this option defaults to the suffix of a default *dumpbase*, i.e., the suffix of the input file when **−dumpbase** is not present in the command line, or *dumpbase* is combined with *dumppfx*.

```
              gcc foo.c -c -o dir/foo.o -dumpbase x-foo.c -dumpbase-ext .c ...
```

    creates *dir/foo.o* as the main output, and generates auxiliary outputs in *dir/x−foo.**, taking the location of the primary output, and dropping the *.c* suffix from the *dumpbase*. Dump outputs retain the suffix: *dir/x−foo.c.**.

    This option is disregarded if it does not match the suffix of a specified *dumpbase*, except as an alternative to the executable suffix when appending the linker output base name to *dumppfx*, as

specified below:

```
gcc foo.c bar.c -o main.out -dumpbase-ext .out ...
```

creates *main.out* as the primary output, and avoids overwriting the auxiliary and dump outputs by using the executable name minus *auxdropsuf* as a prefix, creating auxiliary outputs named *main−foo.** and *main−bar.** and dump outputs named *main−foo.c.** and *main−bar.c.**.

−dumpdir *dumppfx*
    When forming the name of an auxiliary or dump output file, use *dumppfx* as a prefix:

```
gcc -dumpdir pfx- -c foo.c ...
```

creates *foo.o* as the primary output, and auxiliary outputs named *pfx−foo.**, combining the given *dumppfx* with the default *dumpbase* derived from the default primary output, derived in turn from the input name. Dump outputs also take the input name suffix: *pfx−foo.c.**.

If *dumppfx* is to be used as a directory name, it must end with a directory separator:

```
gcc -dumpdir dir/ -c foo.c -o obj/bar.o ...
```

creates *obj/bar.o* as the primary output, and auxiliary outputs named *dir/bar.**, combining the given *dumppfx* with the default *dumpbase* derived from the primary output name. Dump outputs also take the input name suffix: *dir/bar.c.**.

It defaults to the location of the output file, unless the output file is a special file like /dev/null. Options **−save−temps=cwd** and **−save−temps=obj** override this default, just like an explicit **−dumpdir** option. In case multiple such options are given, the last one prevails:

```
gcc -dumpdir pfx- -c foo.c -save-temps=obj ...
```

outputs *foo.o*, with auxiliary outputs named *foo.** because **−save−temps=*** overrides the *dumppfx* given by the earlier **−dumpdir** option. It does not matter that **=obj** is the default for **−save−temps**, nor that the output directory is implicitly the current directory. Dump outputs are named *foo.c.**.

When compiling from multiple input files, if **−dumpbase** is specified, *dumpbase*, minus a *auxdropsuf* suffix, and a dash are appended to (or override, if containing any directory components) an explicit or defaulted *dumppfx*, so that each of the multiple compilations gets differently−named aux and dump outputs.

```
gcc foo.c bar.c -c -dumpdir dir/pfx- -dumpbase main ...
```

outputs auxiliary dumps to *dir/pfx−main−foo.** and *dir/pfx−main−bar.**, appending *dumpbase−* to *dumppfx*. Dump outputs retain the input file suffix: *dir/pfx−main−foo.c.** and *dir/pfx−main−bar.c.**, respectively. Contrast with the single−input compilation:

```
gcc foo.c -c -dumpdir dir/pfx- -dumpbase main ...
```

that, applying **−dumpbase** to a single source, does not compute and append a separate *dumpbase* per input file. Its auxiliary and dump outputs go in *dir/pfx−main.**.

When compiling and then linking from multiple input files, a defaulted or explicitly specified *dumppfx* also undergoes the *dumpbase−* transformation above (e.g. the compilation of *foo.c* and *bar.c* above, but without **−c**). If neither **−dumpdir** nor **−dumpbase** are given, the linker output base name, minus *auxdropsuf*, if specified, or the executable suffix otherwise, plus a dash is appended to the default *dumppfx* instead. Note, however, that unlike earlier cases of linking:

```
gcc foo.c bar.c -dumpdir dir/pfx- -o main ...
```

does not append the output name *main* to *dumppfx*, because **−dumpdir** is explicitly specified. The goal is that the explicitly−specified *dumppfx* may contain the specified output name as part of the prefix, if desired; only an explicitly−specified **−dumpbase** would be combined with it, in order to avoid simply discarding a meaningful option.

When compiling and then linking from a single input file, the linker output base name will only be

appended to the default *dumppfx* as above if it does not share the base name with the single input file name. This has been covered in single−input linking cases above, but not with an explicit **−dumpdir** that inhibits the combination, even if overridden by **−save−temps=***:

```
gcc foo.c −dumpdir alt/pfx− −o dir/main.exe −save-temps=cwd ...
```

Auxiliary outputs are named *foo.**, and dump outputs *foo.c.**, in the current working directory as ultimately requested by **−save−temps=cwd**.

Summing it all up for an intuitive though slightly imprecise data flow: the primary output name is broken into a directory part and a basename part; *dumppfx* is set to the former, unless overridden by **−dumpdir** or **−save−temps=***, and *dumpbase* is set to the latter, unless overriden by **−dumpbase**. If there are multiple inputs or linking, this *dumpbase* may be combined with *dumppfx* and taken from each input file. Auxiliary output names for each input are formed by combining *dumppfx*, *dumpbase* minus suffix, and the auxiliary output suffix; dump output names are only different in that the suffix from *dumpbase* is retained.

When it comes to auxiliary and dump outputs created during LTO recompilation, a combination of *dumppfx* and *dumpbase*, as given or as derived from the linker output name but not from inputs, even in cases in which this combination would not otherwise be used as such, is passed down with a trailing period replacing the compiler−added dash, if any, as a **−dumpdir** option to **lto−wrapper**; being involved in linking, this program does not normally get any **−dumpbase** and **−dumpbase−ext**, and it ignores them.

When running sub−compilers, **lto−wrapper** appends LTO stage names to the received *dumppfx*, ensures it contains a directory component so that it overrides any **−dumpdir**, and passes that as **−dumpbase** to sub−compilers.

−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.

−###

Like **−v** except the commands are not executed and arguments are quoted unless they contain only alphanumeric characters or `./−_`. This is useful for shell scripts to capture the driver−generated command lines.

−−help

Print (on the standard output) a description of the command−line options understood by **gcc**. If the **−v** option is also specified then **−−help** is also passed on to the various processes invoked by **gcc**, so that they can display the command−line options they accept. If the **−Wextra** option has also been specified (prior to the **−−help** option), then command−line options that have no documentation associated with them are also displayed.

−−target−help

Print (on the standard output) a description of target−specific command−line options for each tool. For some targets extra target−specific information may also be printed.

−−help={*class*|[^]*qualifier*}[,...]

Print (on the standard output) a description of the command−line options understood by the compiler that fit into all specified classes and qualifiers. These are the supported classes:

optimizers

Display all of the optimization options supported by the compiler.

warnings

Display all of the options controlling warning messages produced by the compiler.

target

Display target−specific options. Unlike the **−−target−help** option however, target−specific options of the linker and assembler are not displayed. This is because those tools do not currently support the extended **−−help=** syntax.

params
> Display the values recognized by the **−−param** option.

language
> Display the options supported for *language*, where *language* is the name of one of the languages supported in this version of GCC. If an option is supported by all languages, one needs to select **common** class.

common
> Display the options that are common to all languages.

These are the supported qualifiers:

undocumented
> Display only those options that are undocumented.

joined
> Display options taking an argument that appears after an equal sign in the same continuous piece of text, such as: **−−help=target**.

separate
> Display options taking an argument that appears as a separate word following the original option, such as: **−o output−file**.

Thus for example to display all the undocumented target−specific switches supported by the compiler, use:

        −−help=target,undocumented

The sense of a qualifier can be inverted by prefixing it with the ˆ character, so for example to display all binary warning options (i.e., ones that are either on or off and that do not take an argument) that have a description, use:

        −−help=warnings,ˆjoined,ˆundocumented

The argument to **−−help=** should not consist solely of inverted qualifiers.

Combining several classes is possible, although this usually restricts the output so much that there is nothing to display. One case where it does work, however, is when one of the classes is *target*. For example, to display all the target−specific optimization options, use:

        −−help=target,optimizers

The **−−help=** option can be repeated on the command line. Each successive use displays its requested class of options, skipping those that have already been displayed. If **−−help** is also specified anywhere on the command line then this takes precedence over any **−−help=** option.

If the **−Q** option appears on the command line before the **−−help=** option, then the descriptive text displayed by **−−help=** is changed. Instead of describing the displayed options, an indication is given as to whether the option is enabled, disabled or set to a specific value (assuming that the compiler knows this at the point where the **−−help=** option is used).

Here is a truncated example from the ARM port of **gcc**:

```
% gcc −Q −mabi=2 −−help=target −c
The following options are target specific:
−mabi=                          2
−mabort−on−noreturn             [disabled]
−mapcs                          [disabled]
```

The output is sensitive to the effects of previous command−line options, so for example it is possible to find out which optimizations are enabled at **−O2** by using:

```
−Q −O2 −−help=optimizers
```

Alternatively you can discover which binary optimizations are enabled by **−O3** by using:

```
gcc -c -Q -O3 --help=optimizers > /tmp/O3-opts
gcc -c -Q -O2 --help=optimizers > /tmp/O2-opts
diff /tmp/O2-opts /tmp/O3-opts | grep enabled
```

−−version

Display the version number and copyrights of the invoked GCC.

−pass−exit−codes

Normally the **gcc** program exits with the code of 1 if any phase of the compiler returns a non−success return code. If you specify **−pass−exit−codes**, the **gcc** program instead returns with the numerically highest error produced by any phase returning an error indication. The C, C++, and Fortran front ends return 4 if an internal compiler error is encountered.

−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.

−specs=*file*

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

−wrapper

Invoke all subcommands under a wrapper program. The name of the wrapper program and its parameters are passed as a comma separated list.

```
gcc -c t.c -wrapper gdb,--args
```

This invokes all subprograms of **gcc** under **gdb −−args**, thus the invocation of **cc1** is **gdb −−args cc1 ...**.

−ffile−prefix−map=*old***=***new*

When compiling files residing in directory *old*, record any references to them in the result of the compilation as if the files resided in directory *new* instead. Specifying this option is equivalent to specifying all the individual **−f*−prefix−map** options. This can be used to make reproducible builds that are location independent. Directories referenced by directives are not affected by these options. See also **−fmacro−prefix−map**, **−fdebug−prefix−map**, **−fprofile−prefix−map** and **−fcanon−prefix−map**.

−fcanon−prefix−map

For the **−f*−prefix−map** options normally comparison of *old* prefix against the filename that would be normally referenced in the result of the compilation is done using textual comparison of the prefixes, or ignoring character case for case insensitive filesystems and considering slashes and backslashes as equal on DOS based filesystems. The **−fcanon−prefix−map** causes such comparisons to be done on canonicalized paths of *old* and the referenced filename.

−fplugin=*name***.so**

Load the plugin code in file *name*.so, assumed to be a shared object to be dlopen'd by the compiler. The base name of the shared object file is used to identify the plugin for the purposes of argument parsing (See **−fplugin−arg−***name−key***=***value* below). Each plugin should define the callback functions specified in the Plugins API.

−fplugin−arg−*name−key***=***value*

Define an argument called *key* with a value of *value* for the plugin called *name*.

−fdump−ada−spec[**−slim**]

For C and C++ source and include files, generate corresponding Ada specs.

−fada−spec−parent=*unit*

    In conjunction with **−fdump−ada−spec**[**−slim**] above, generate Ada specs as child units of parent *unit*.

−fdump−go−spec=*file*

    For input files in any language, generate corresponding Go declarations in *file*. This generates Go `const`, `type`, `var`, and `func` declarations which may be a useful way to start writing a Go interface to code written in some other language.

@*file*

    Read command−line options from *file*. The options read are inserted in place of the original @*file* option. If *file* does not exist, or cannot be read, then the option will be treated literally, and not removed.

    Options in *file* are separated by whitespace. A whitespace character may be included in an option by surrounding the entire option in either single or double quotes. Any character (including a backslash) may be included by prefixing the character to be included with a backslash. The *file* may itself contain additional @*file* options; any such options will be processed recursively.

Compiling C++ Programs

C++ source files conventionally use one of the suffixes **.C**, **.cc**, **.cpp**, **.CPP**, **.c++**, **.cp**, or **.cxx**; C++ header files often use **.hh**, **.hpp**, **.H**, or (for shared template code) **.tcc**; preprocessed C++ files use the suffix **.ii**; and C++20 module interface units sometimes use **.ixx**, **.cppm**, **.cxxm**, **.c++m**, or **.ccm**.

GCC recognizes files with these names and compiles them as C++ programs even if you call the compiler the same way as for compiling C programs (usually with the name **gcc**).

However, the use of **gcc** does not add the C++ library. **g++** is a program that calls GCC and automatically specifies linking against the C++ library. It treats **.c**, **.h** and **.i** files as C++ source files instead of C source files unless **−x** is used. This program is also useful when precompiling a C header file with a **.h** extension for use in C++ compilations. On many systems, **g++** is also installed with the name **c++**.

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

Options Controlling C Dialect

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

−ansi

    In C mode, this is equivalent to **−std=c90**. In C++ mode, it is equivalent to **−std=c++98**.

−std=

    Determine the language standard. This option is currently only supported when compiling C or C++.

    The compiler can accept several base standards, such as **c90** or **c++98**, and GNU dialects of those standards, such as **gnu90** or **gnu++98**. When a base standard is specified, the compiler accepts all programs following that standard plus those using GNU extensions that do not contradict it. For example, **−std=c90** turns off certain features of GCC that are incompatible with ISO C90, such as the `asm` and `typeof` keywords, but not other GNU extensions that do not have a meaning in ISO C90, such as omitting the middle term of a `?:` expression. On the other hand, when a GNU dialect of a standard is specified, all features supported by the compiler are enabled, even when those features change the meaning of the base standard. As a result, some strict−conforming programs may be rejected. The particular standard is used by **−Wpedantic** to identify which features are GNU extensions given that version of the standard. For example **−std=gnu90 −Wpedantic** warns about C++ style **//** comments, while **−std=gnu99 −Wpedantic** does not.

    A value for this option must be provided; possible values are

c90
c89
iso9899:1990

Support all ISO C90 programs (certain GNU extensions that conflict with ISO C90 are disabled). Same as **−ansi** for C code.

iso9899:199409

ISO C90 as modified in amendment 1.

c99
c9x
iso9899:1999
iso9899:199x

ISO C99. This standard is substantially completely supported, modulo bugs and floating−point issues (mainly but not entirely relating to optional C99 features from Annexes F and G). See **<https://gcc.gnu.org/c99status.html>** for more information. The names **c9x** and **iso9899:199x** are deprecated.

c11
c1x
iso9899:2011

ISO C11, the 2011 revision of the ISO C standard. This standard is substantially completely supported, modulo bugs, floating−point issues (mainly but not entirely relating to optional C11 features from Annexes F and G) and the optional Annexes K (Bounds−checking interfaces) and L (Analyzability). The name **c1x** is deprecated.

c17
c18
iso9899:2017
iso9899:2018

ISO C17, the 2017 revision of the ISO C standard (published in 2018). This standard is same as C11 except for corrections of defects (all of which are also applied with **−std=c11**) and a new value of `__STDC_VERSION__`, and so is supported to the same extent as C11.

c23
c2x
iso9899:2024

ISO C23, the 2023 revision of the ISO C standard (published in 2024). The name **c2x** is deprecated.

c2y The next version of the ISO C standard, still under development. The support for this version is experimental and incomplete.

gnu90
gnu89

GNU dialect of ISO C90 (including some C99 features).

gnu99
gnu9x

GNU dialect of ISO C99. The name **gnu9x** is deprecated.

gnu11
gnu1x

GNU dialect of ISO C11. The name **gnu1x** is deprecated.

gnu17
gnu18

GNU dialect of ISO C17.

gnu23
gnu2x
  GNU dialect of ISO C23. This is the default for C code. The name **gnu2x** is deprecated.

gnu2y
  The next version of the ISO C standard, still under development, plus GNU extensions. The support for this version is experimental and incomplete. The name **gnu2x** is deprecated.

c++98
c++03
  The 1998 ISO C++ standard plus the 2003 technical corrigendum and some additional defect reports. Same as **−ansi** for C++ code.

gnu++98
gnu++03
  GNU dialect of **−std=c++98**.

c++11
c++0x
  The 2011 ISO C++ standard plus amendments. The name **c++0x** is deprecated.

gnu++11
gnu++0x
  GNU dialect of **−std=c++11**. The name **gnu++0x** is deprecated.

c++14
c++1y
  The 2014 ISO C++ standard plus amendments. The name **c++1y** is deprecated.

gnu++14
gnu++1y
  GNU dialect of **−std=c++14**. The name **gnu++1y** is deprecated.

c++17
c++1z
  The 2017 ISO C++ standard plus amendments. The name **c++1z** is deprecated.

gnu++17
gnu++1z
  GNU dialect of **−std=c++17**. This is the default for C++ code. The name **gnu++1z** is deprecated.

c++20
c++2a
  The 2020 ISO C++ standard plus amendments. Support is experimental, and could change in incompatible ways in future releases. The name **c++2a** is deprecated.

gnu++20
gnu++2a
  GNU dialect of **−std=c++20**. Support is experimental, and could change in incompatible ways in future releases. The name **gnu++2a** is deprecated.

c++23
c++2b
  The 2023 ISO C++ standard plus amendments (published in 2024). Support is experimental, and could change in incompatible ways in future releases. The name **c++2b** is deprecated.

gnu++23
gnu++2b
  GNU dialect of **−std=c++23**. Support is experimental, and could change in incompatible ways in future releases. The name **gnu++2b** is deprecated.

c++2c

c++26

> The next revision of the ISO C++ standard, planned for 2026. Support is highly experimental, and will almost certainly change in incompatible ways in future releases.

gnu++2c

gnu++26

> GNU dialect of **−std=c++2c**. Support is highly experimental, and will almost certainly change in incompatible ways in future releases.

−aux−info *filename*

> Output to the given filename prototyped declarations for all functions declared and/or defined in a translation unit, including those in header files. This option is silently ignored in any language other than C.
>
> Besides declarations, the file indicates, in comments, the origin of each declaration (source file and line), whether the declaration was implicit, prototyped or unprototyped (**I**, **N** for new or **O** for old, respectively, in the first character after the line number and the colon), and whether it came from a declaration or a definition (**C** or **F**, respectively, in the following character). In the case of function definitions, a K&R−style list of arguments followed by their declarations is also provided, inside comments, after the declaration.

−fno−asm

> Do not recognize `asm`, `inline` or `typeof` as a keyword, so that code can use these words as identifiers. You can use the keywords `__asm__`, `__inline__` and `__typeof__` instead. In C, **−ansi** implies **−fno−asm**.
>
> In C++, `inline` is a standard keyword and is not affected by this switch. You may want to use the **−fno−gnu−keywords** flag instead, which disables `typeof` but not `asm` and `inline`. In C99 mode (**−std=c99** or **−std=gnu99**), this switch only affects the `asm` and `typeof` keywords, since `inline` is a standard keyword in ISO C99. In C23 mode (**−std=c23** or **−std=gnu23**), this switch only affects the `asm` keyword, since `typeof` is a standard keyword in ISO C23.

−fno−builtin

−fno−builtin−*function*

> Don't recognize built−in functions that do not begin with **__builtin_** as prefix.
>
> GCC normally generates special code to handle certain built−in functions more efficiently; for instance, calls to `alloca` may become single instructions which adjust the stack directly, and calls to `memcpy` may become inline copy loops. The resulting code is often both smaller and faster, but since the function calls no longer appear as such, you cannot set a breakpoint on those calls, nor can you change the behavior of the functions by linking with a different library. In addition, when a function is recognized as a built−in function, GCC may use information about that function to warn about problems with calls to that function, or to generate more efficient code, even if the resulting code still contains calls to that function. For example, warnings are given with **−Wformat** for bad calls to `printf` when `printf` is built in and `strlen` is known not to modify global memory.
>
> With the **−fno−builtin−***function* option only the built−in function *function* is disabled. *function* must not begin with **__builtin_**. If a function is named that is not built−in in this version of GCC, this option is ignored. There is no corresponding **−fbuiltin−***function* option; if you wish to enable built−in functions selectively when using **−fno−builtin** or **−ffreestanding**, you may define macros such as:

```
#define abs(n)          __builtin_abs ((n))
#define strcpy(d, s)    __builtin_strcpy ((d), (s))
```

−fcond−mismatch

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

−ffreestanding

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

−fgimple

Enable parsing of function definitions marked with `__GIMPLE`. This is an experimental feature that allows unit testing of GIMPLE passes.

−fgnu−tm

When the option **−fgnu−tm** is specified, the compiler generates code for the Linux variant of Intel's current Transactional Memory ABI specification document (Revision 1.1, May 6 2009). This is an experimental feature whose interface may change in future versions of GCC, as the official specification changes. Please note that not all architectures are supported for this feature.

For more information on GCC's support for transactional memory,

Note that the transactional memory feature is not supported with non−call exceptions (**−fnon−call−exceptions**).

−fgnu89−inline

The option **−fgnu89−inline** tells GCC to use the traditional GNU semantics for `inline` functions when in C99 mode.

Using this option is roughly equivalent to adding the `gnu_inline` function attribute to all inline functions.

The option **−fno−gnu89−inline** explicitly tells GCC to use the C99 semantics for `inline` when in C99 or gnu99 mode (i.e., it specifies the default behavior). This option is not supported in **−std=c90** or **−std=gnu90** mode.

The preprocessor macros `__GNUC_GNU_INLINE__` and `__GNUC_STDC_INLINE__` may be used to check which semantics are in effect for `inline` functions.

−fhosted

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

−flax−vector−conversions

Allow implicit conversions between vectors with differing numbers of elements and/or incompatible element types. This option should not be used for new code.

−fms−extensions

Accept some non−standard constructs used in Microsoft header files.

In C++ code, this allows member names in structures to be similar to previous types declarations.

```
typedef int UOW;
struct ABC {
  UOW UOW;
};
```

Some cases of unnamed fields in structures and unions are only accepted with this option.

Note that this option is off for all targets except for x86 targets using ms−abi.

−fpermitted−flt−eval−methods=*style*

ISO/IEC TS 18661−3 defines new permissible values for `FLT_EVAL_METHOD` that indicate that operations and constants with a semantic type that is an interchange or extended format should be evaluated to the precision and range of that type. These new values are a superset of those permitted under C99/C11, which does not specify the meaning of other positive values of `FLT_EVAL_METHOD`.

As such, code conforming to C11 may not have been written expecting the possibility of the new values.

−fpermitted−flt−eval−methods specifies whether the compiler should allow only the values of `FLT_EVAL_METHOD` specified in C99/C11, or the extended set of values specified in ISO/IEC TS 18661−3.

style is either `c11` or `ts-18661-3` as appropriate.

The default when in a standards compliant mode (**−std=c11** or similar) is **−fpermitted−flt−eval−methods=c11**. The default when in a GNU dialect (**−std=gnu11** or similar) is **−fpermitted−flt−eval−methods=ts−18661−3**.

The **−fdeps−*** options are used to extract structured dependency information for a source. This involves determining what resources provided by other source files will be required to compile the source as well as what resources are provided by the source. This information can be used to add required dependencies between compilation rules of dependent sources based on their contents rather than requiring such information be reflected within the build tools as well.

−fdeps−file=_file_
> Where to write structured dependency information.

−fdeps−format=_format_
> The format to use for structured dependency information. **p1689r5** is the only supported format right now. Note that when this argument is specified, the output of **−MF** is stripped of some information (namely C++ modules) so that it does not use extended makefile syntax not understood by most tools.

−fdeps−target=_file_
> Analogous to **−MT** but for structured dependency information. This indicates the target which will ultimately need any required resources and provide any resources extracted from the source that may be required by other sources.

−fplan9−extensions
> Accept some non−standard constructs used in Plan 9 code.

> This enables **−fms−extensions**, permits passing pointers to structures with anonymous fields to functions that expect pointers to elements of the type of the field, and permits referring to anonymous fields declared using a typedef.   This is only supported for C, not C++.

−fsigned−bitfields
−funsigned−bitfields
−fno−signed−bitfields
−fno−unsigned−bitfields
> These options control whether a bit−field is signed or unsigned, when the declaration does not use either `signed` or `unsigned`. By default, such a bit−field is signed, because this is consistent: the basic integer types such as `int` are signed types.

−fsigned−char
> Let the type `char` be signed, like `signed char`.

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

−funsigned−char
> Let the type `char` be unsigned, like `unsigned char`.

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

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

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

−fstrict−flex−arrays (C and C++ only)
−fstrict−flex−arrays=_level_ (C and C++ only)

Control when to treat the trailing array of a structure as a flexible array member for the purpose of accessing the elements of such an array. The value of _level_ controls the level of strictness.

−fstrict−flex−arrays is equivalent to **−fstrict−flex−arrays=3**, which is the strictest; a trailing array is treated as a flexible array member only when it is declared as a flexible array member per C99 standard onwards.

The negative form **−fno−strict−flex−arrays** is equivalent to **−fstrict−flex−arrays=0**, which is the least strict. In this case all trailing arrays of structures are treated as flexible array members.

There are two more levels in between 0 and 3, which are provided to support older code that uses the GCC zero−length array extension (**[0]**) or one−element array as flexible array members (**[1]**). When _level_ is 1, the trailing array is treated as a flexible array member when it is declared as either **[]**, **[0]**, or **[1]**. When _level_ is 2, the trailing array is treated as a flexible array member when it is declared as either **[]**, or **[0]**.

You can control this behavior for a specific trailing array field of a structure by using the variable attribute `strict_flex_array` attribute.

The **−fstrict_flex_arrays** option interacts with the **−Wstrict−flex−arrays** option.

−fsso−struct=_endianness_

Set the default scalar storage order of structures and unions to the specified endianness. The accepted values are **big−endian**, **little−endian** and **native** for the native endianness of the target (the default). This option is not supported for C++.

Warning: the **−fsso−struct** switch causes GCC to generate code that is not binary compatible with code generated without it if the specified endianness is not the native endianness of the target.

Options Controlling C++ Dialect

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

```
g++ −g −fstrict-enums −O −c firstClass.C
```

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

Some options for compiling C programs, such as **−std**, are also relevant for C++ programs.

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

−fabi−version=_n_

Use version *n* of the C++ ABI. The default is version 0.

Version 0 refers to the version conforming most closely to the C++ ABI specification. Therefore, the ABI obtained using version 0 will change in different versions of G++ as ABI bugs are fixed.

Version 1 is the version of the C++ ABI that first appeared in G++ 3.2.

Version 2 is the version of the C++ ABI that first appeared in G++ 3.4, and was the default through G++ 4.9.

Version 3 corrects an error in mangling a constant address as a template argument.

Version 4, which first appeared in G++ 4.5, implements a standard mangling for vector types.

Version 5, which first appeared in G++ 4.6, corrects the mangling of attribute const/volatile on function pointer types, decltype of a plain decl, and use of a function parameter in the declaration of

another parameter.

Version 6, which first appeared in G++ 4.7, corrects the promotion behavior of C++11 scoped enums and the mangling of template argument packs, const/static_cast, prefix ++ and −−, and a class scope function used as a template argument.

Version 7, which first appeared in G++ 4.8, that treats nullptr_t as a builtin type and corrects the mangling of lambdas in default argument scope.

Version 8, which first appeared in G++ 4.9, corrects the substitution behavior of function types with function−cv−qualifiers.

Version 9, which first appeared in G++ 5.2, corrects the alignment of `nullptr_t`.

Version 10, which first appeared in G++ 6.1, adds mangling of attributes that affect type identity, such as ia32 calling convention attributes (e.g. **stdcall**).

Version 11, which first appeared in G++ 7, corrects the mangling of sizeof... expressions and operator names. For multiple entities with the same name within a function, that are declared in different scopes, the mangling now changes starting with the twelfth occurrence. It also implies **−fnew−inheriting−ctors**.

Version 12, which first appeared in G++ 8, corrects the calling conventions for empty classes on the x86_64 target and for classes with only deleted copy/move constructors. It accidentally changes the calling convention for classes with a deleted copy constructor and a trivial move constructor.

Version 13, which first appeared in G++ 8.2, fixes the accidental change in version 12.

Version 14, which first appeared in G++ 10, corrects the mangling of the nullptr expression.

Version 15, which first appeared in G++ 10.3, corrects G++ 10 ABI tag regression.

Version 16, which first appeared in G++ 11, changes the mangling of `__alignof__` to be distinct from that of `alignof`, and dependent operator names.

Version 17, which first appeared in G++ 12, fixes layout of classes that inherit from aggregate classes with default member initializers in C++14 and up.

Version 18, which first appeared in G++ 13, fixes manglings of lambdas that have additional context.

Version 19, which first appeared in G++ 14, fixes manglings of structured bindings to include ABI tags.

Version 20, which first appeared in G++ 15, fixes manglings of lambdas in static data member initializers.

See also **−Wabi**.

−fabi−compat−version=_n_

On targets that support strong aliases, G++ works around mangling changes by creating an alias with the correct mangled name when defining a symbol with an incorrect mangled name. This switch specifies which ABI version to use for the alias.

With **−fabi−version=0** (the default), this defaults to 13 (GCC 8.2 compatibility). If another ABI version is explicitly selected, this defaults to 0. For compatibility with GCC versions 3.2 through 4.9, use **−fabi−compat−version=2**.

If this option is not provided but **−Wabi=**_n_ is, that version is used for compatibility aliases. If this option is provided along with **−Wabi** (without the version), the version from this option is used for the warning.

−fno−access−control

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

–faligned–new

Enable support for C++17 new of types that require more alignment than `void* ::operator new(std::size_t)` provides. A numeric argument such as `–faligned-new=32` can be used to specify how much alignment (in bytes) is provided by that function, but few users will need to override the default of `alignof(std::max_align_t)`.

This flag is enabled by default for **–std=c++17**.

–fno–assume–sane–operators–new

The C++ standard allows replacing the global `new`, `new[]`, `delete` and `delete[]` operators, though a lot of C++ programs don't replace them and just use the implementation provided version. Furthermore, the C++ standard allows omitting those calls if they are made from new or delete expressions (and by extension the same is assumed if `__builtin_operator_new` or `__builtin_operator_delete` functions are used). This option allows control over some optimizations around calls to those operators. With `-fassume-sane-operators-new-delete` option GCC may assume that calls to the replaceable global operators from new or delete expressions or from `__builtin_operator_new` or `__builtin_operator_delete` calls don't read or modify any global variables or variables whose address could escape to the operators (global state; except for `errno` for the `new` and `new[]` operators). This allows most optimizations across those calls and is something that the implementation provided operators satisfy unless `malloc` implementation details are observable in the code or unless `malloc` hooks are used, but might not be satisfied if a program replaces those operators. This behavior is enabled by default. With `-fno-assume-sane-operators-new-delete` option GCC must assume all these calls (whether from new or delete expressions or called directly) may read and write global state unless proven otherwise (e.g. when GCC compiles their implementation). Use this option if those operators are or may be replaced and code needs to expect such behavior.

–fchar8_t
–fno–char8_t

Enable support for `char8_t` as adopted for C++20. This includes the addition of a new `char8_t` fundamental type, changes to the types of UTF–8 string and character literals, new signatures for user–defined literals, associated standard library updates, and new `__cpp_char8_t` and `__cpp_lib_char8_t` feature test macros.

This option enables functions to be overloaded for ordinary and UTF–8 strings:

```
int f(const char *);     // #1
int f(const char8_t *); // #2
int v1 = f("text");      // Calls #1
int v2 = f(u8"text");    // Calls #2
```

and introduces new signatures for user–defined literals:

```
int operator""_udl1(char8_t);
int v3 = u8'x'_udl1;
int operator""_udl2(const char8_t*, std::size_t);
int v4 = u8"text"_udl2;
template<typename T, T...> int operator""_udl3();
int v5 = u8"text"_udl3;
```

The change to the types of UTF–8 string and character literals introduces incompatibilities with ISO C++11 and later standards. For example, the following code is well–formed under ISO C++11, but is ill–formed when **–fchar8_t** is specified.

```
                    const char *cp = u8"xx";// error: invalid conversion from
                                      //           `const char8_t*' to `const char*'
                    int f(const char*);
                    auto v = f(u8"xx");      // error: invalid conversion from
                                      //           `const char8_t*' to `const char*'
                    std::string s{u8"xx"};  // error: no matching function for call to
                                      //           `std::basic_string<char>::basic_string(
                    using namespace std::literals;
                    s = u8"xx"s;            // error: conversion from
                                      //           `basic_string<char8_t>' to non-scalar
                                      //           type `basic_string<char>' requested
```

−fcheck−new

Check that the pointer returned by operator new is non−null before attempting to modify the storage allocated. This check is normally unnecessary because the C++ standard specifies that operator new only returns 0 if it is declared throw(), in which case the compiler always checks the return value even without this option. In all other cases, when operator new has a non−empty exception specification, memory exhaustion is signalled by throwing std::bad_alloc. See also **new (nothrow)**.

−fconcepts

Enable support for the C++ Concepts feature for constraining template arguments. With **−std=c++20** and above, Concepts are part of the language standard, so **−fconcepts** defaults to on.

Some constructs that were allowed by the earlier C++ Extensions for Concepts Technical Specification, ISO 19217 (2015), but didn't make it into the standard, could additionally be enabled by **−fconcepts−ts**. The option **−fconcepts−ts** was deprecated in GCC 14 and removed in GCC 15; users are expected to convert their code to C++20 concepts.

−fconstexpr−depth=_n_

Set the maximum nested evaluation depth for C++11 constexpr functions to _n_. A limit is needed to detect endless recursion during constant expression evaluation. The minimum specified by the standard is 512.

−fconstexpr−cache−depth=_n_

Set the maximum level of nested evaluation depth for C++11 constexpr functions that will be cached to _n_. This is a heuristic that trades off compilation speed (when the cache avoids repeated calculations) against memory consumption (when the cache grows very large from highly recursive evaluations). The default is 8. Very few users are likely to want to adjust it, but if your code does heavy constexpr calculations you might want to experiment to find which value works best for you.

−fconstexpr−fp−except

Annex F of the C standard specifies that IEC559 floating point exceptions encountered at compile time should not stop compilation. C++ compilers have historically not followed this guidance, instead treating floating point division by zero as non−constant even though it has a well defined value. This flag tells the compiler to give Annex F priority over other rules saying that a particular operation is undefined.

```
                    constexpr float inf = 1./0.; // OK with −fconstexpr−fp−except
```

−fconstexpr−loop−limit=_n_

Set the maximum number of iterations for a loop in C++14 constexpr functions to _n_. A limit is needed to detect infinite loops during constant expression evaluation. The default is 262144 (1<<18).

−fconstexpr−ops−limit=_n_

Set the maximum number of operations during a single constexpr evaluation. Even when number of iterations of a single loop is limited with the above limit, if there are several nested loops and each of them has many iterations but still smaller than the above limit, or if in a body of some loop or even outside of a loop too many expressions need to be evaluated, the resulting constexpr evaluation might take too long. The default is 33554432 (1<<25).

−fcontracts

Enable experimental support for the C++ Contracts feature, as briefly added to and then removed from the C++20 working paper (N4820). The implementation also includes proposed enhancements from papers P1290, P1332, and P1429. This functionality is intended mostly for those interested in experimentation towards refining the feature to get it into shape for a future C++ standard.

On violation of a checked contract, the violation handler is called. Users can replace the violation handler by defining

```
void
handle_contract_violation (const std::experimental::contract_violation&);
```

There are different sets of additional flags that can be used together to specify which contracts will be checked and how, for N4820 contracts, P1332 contracts, or P1429 contracts; these sets cannot be used together.

−fcontract−mode=[on|off]

Control whether any contracts have any semantics at all. Defaults to on.

−fcontract−assumption−mode=[on|off]

[N4820] Control whether contracts with level **axiom** should have the assume semantic. Defaults to on.

−fcontract−build−level=[off|default|audit]

[N4820] Specify which level of contracts to generate checks for. Defaults to **default**.

−fcontract−continuation−mode=[on|off]

[N4820] Control whether to allow the program to continue executing after a contract violation. That is, do checked contracts have the **maybe** semantic described below rather than the **never** semantic. Defaults to off.

−fcontract−role=<name>:<default>,<audit>,<axiom>

[P1332] Specify the concrete semantics for each contract level of a particular contract role.

−fcontract−semantic=[default|audit|axiom]:<semantic>

[P1429] Specify the concrete semantic for a particular contract level.

−fcontract−strict−declarations=[on|off]

Control whether to reject adding contracts to a function after its first declaration. Defaults to off.

The possible concrete semantics for that can be specified with **−fcontract−role** or **−fcontract−semantic** are:

ignore
    This contract has no effect.

assume
    This contract is treated like C++23 `[[assume]]`.

check_never_continue
never
abort
    This contract is checked. If it fails, the violation handler is called. If the handler returns, `std::terminate` is called.

check_maybe_continue
maybe
    This contract is checked. If it fails, the violation handler is called. If the handler returns, execution continues normally.

−fcoroutines

Enable support for the C++ coroutines extension (experimental).

−fdiagnostics−all−candidates

Permit the C++ front end to note all candidates during overload resolution failure, including when a deleted function is selected.

−fno−elide−constructors

The C++ standard allows an implementation to omit creating a temporary that is only used to initialize another object of the same type. Specifying this option disables that optimization, and forces G++ to call the copy constructor in all cases. This option also causes G++ to call trivial member functions which otherwise would be expanded inline.

In C++17, the compiler is required to omit these temporaries, but this option still affects trivial member functions.

−fno−enforce−eh−specs

Don't generate code to check for violation of exception specifications at run time. This option violates the C++ standard, but may be useful for reducing code size in production builds, much like defining `NDEBUG`. This does not give user code permission to throw exceptions in violation of the exception specifications; the compiler still optimizes based on the specifications, so throwing an unexpected exception results in undefined behavior at run time.

−fextern−tls−init
−fno−extern−tls−init

The C++11 and OpenMP standards allow `thread_local` and `threadprivate` variables to have dynamic (runtime) initialization. To support this, any use of such a variable goes through a wrapper function that performs any necessary initialization. When the use and definition of the variable are in the same translation unit, this overhead can be optimized away, but when the use is in a different translation unit there is significant overhead even if the variable doesn't actually need dynamic initialization. If the programmer can be sure that no use of the variable in a non−defining TU needs to trigger dynamic initialization (either because the variable is statically initialized, or a use of the variable in the defining TU will be executed before any uses in another TU), they can avoid this overhead with the **−fno−extern−tls−init** option.

On targets that support symbol aliases, the default is **−fextern−tls−init**. On targets that do not support symbol aliases, the default is **−fno−extern−tls−init**.

−ffold−simple−inlines
−fno−fold−simple−inlines

Permit the C++ frontend to fold calls to `std::move`, `std::forward`, `std::addressof` and `std::as_const`. In contrast to inlining, this means no debug information will be generated for such calls. Since these functions are rarely interesting to debug, this flag is enabled by default unless **−fno−inline** is active.

−fno−gnu−keywords

Do not recognize `typeof` as a keyword, so that code can use this word as an identifier. You can use the keyword `__typeof__` instead. This option is implied by the strict ISO C++ dialects: **−ansi**, **−std=c++98**, **−std=c++11**, etc.

−fno−immediate−escalation

Do not enable immediate function escalation whereby certain functions can be promoted to consteval, as specified in P2564R3. For example:

```
consteval int id(int i) { return i; }

constexpr int f(auto t)
{
  return t + id(t); // id causes f<int> to be promoted to consteval
}

void g(int i)
{
```

```
                            f (3);
                         }
```

compiles in C++20: `f` is an immediate−escalating function (due to the `auto` it is a function template and is declared `constexpr`) and `id(t)` is an immediate−escalating expression, so `f` is promoted to `consteval`. Consequently, the call to `id(t)` is in an immediate context, so doesn't have to produce a constant (that is the mechanism allowing consteval function composition). However, with **−fno−immediate−escalation**, `f` is not promoted to `consteval`, and since the call to consteval function `id(t)` is not a constant expression, the compiler rejects the code.

This option is turned on by default; it is only effective in C++20 mode or later.

−fimplicit−constexpr
Make inline functions implicitly constexpr, if they satisfy the requirements for a constexpr function. This option can be used in C++14 mode or later. This can result in initialization changing from dynamic to static and other optimizations.

−fno−implicit−templates
Never emit code for non−inline templates that are instantiated implicitly (i.e. by use); only emit code for explicit instantiations. If you use this option, you must take care to structure your code to include all the necessary explicit instantiations to avoid getting undefined symbols at link time.

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

−fno−implement−inlines
To save space, do not emit out−of−line copies of inline functions controlled by `#pragma implementation`. This causes linker errors if these functions are not inlined everywhere they are called.

−fmodules
−fno−modules
Enable support for C++20 modules. The **−fno−modules** is usually not needed, as that is the default. Even though this is a C++20 feature, it is not currently implicitly enabled by selecting that standard version.

−fmodule−header
−fmodule−header=user
−fmodule−header=system
Compile a header file to create an importable header unit.

−fmodule−implicit−inline
Member functions defined in their class definitions are not implicitly inline for modular code. This is different to traditional C++ behavior, for good reasons. However, it may result in a difficulty during code porting. This option makes such function definitions implicitly inline. It does however generate an ABI incompatibility, so you must use it everywhere or nowhere. (Such definitions outside of a named module remain implicitly inline, regardless.)

−fno−module−lazy
Disable lazy module importing and module mapper creation.

−fmodule−mapper=[*hostname*]**:***port*[**?***ident*]
−fmodule−mapper=|*program*[**?***ident*] *args...*
−fmodule−mapper==*socket*[**?***ident*]
−fmodule−mapper=<>[*inout*][**?***ident*]
−fmodule−mapper=<*in***>***out*[**?***ident*]
−fmodule−mapper=*file*[**?***ident*]
An oracle to query for module name to filename mappings. If unspecified the **CXX_MODULE_MAPPER** environment variable is used, and if that is unset, an in−process default is provided.

−fmodule−only

Only emit the Compiled Module Interface, inhibiting any object file.

−fms−extensions

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

−fnew−inheriting−ctors

Enable the P0136 adjustment to the semantics of C++11 constructor inheritance. This is part of C++17 but also considered to be a Defect Report against C++11 and C++14. This flag is enabled by default unless **−fabi−version=10** or lower is specified.

−fnew−ttp−matching

Enable the P0522 resolution to Core issue 150, template template parameters and default arguments: this allows a template with default template arguments as an argument for a template template parameter with fewer template parameters. This flag is enabled by default for **−std=c++17**.

−fno−nonansi−builtins

Disable built−in declarations of functions that are not mandated by ANSI/ISO C. These include `ffs`, `alloca`, `_exit`, `index`, `bzero`, `conjf`, and other related functions.

−fnothrow−opt

Treat a `throw()` exception specification as if it were a `noexcept` specification to reduce or eliminate the text size overhead relative to a function with no exception specification. If the function has local variables of types with non−trivial destructors, the exception specification actually makes the function smaller because the EH cleanups for those variables can be optimized away. The semantic effect is that an exception thrown out of a function with such an exception specification results in a call to `terminate` rather than `unexpected`.

−fno−operator−names

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

−fno−optional−diags

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

−fno−pretty−templates

When an error message refers to a specialization of a function template, the compiler normally prints the signature of the template followed by the template arguments and any typedefs or typenames in the signature (e.g. `void f(T) [with T = int]` rather than `void f(int)`) so that it's clear which template is involved. When an error message refers to a specialization of a class template, the compiler omits any template arguments that match the default template arguments for that template. If either of these behaviors make it harder to understand the error message rather than easier, you can use **−fno−pretty−templates** to disable them.

−frange−for−ext−temps

Enable lifetime extension of C++ range based for temporaries. With **−std=c++23** and above this is part of the language standard, so lifetime of the temporaries is extended until the end of the loop by default. This option allows enabling that behavior also in earlier versions of the standard.

−fno−rtti

Disable generation of information about every class with virtual functions for use by the C++ run−time type identification features (`dynamic_cast` and `typeid`). If you don't use those parts of the language, you can save some space by using this flag. Note that exception handling uses the same information, but G++ generates it as needed. The `dynamic_cast` operator can still be used for casts that do not require run−time type information, i.e. casts to `void *` or to unambiguous base classes.

Mixing code compiled with **−frtti** with that compiled with **−fno−rtti** may not work. For example, programs may fail to link if a class compiled with **−fno−rtti** is used as a base for a class compiled with **−frtti**.

−fsized−deallocation

Enable the built−in global declarations

```
void operator delete (void *, std::size_t) noexcept;
void operator delete[] (void *, std::size_t) noexcept;
```

as introduced in C++14. This is useful for user−defined replacement deallocation functions that, for example, use the size of the object to make deallocation faster. Enabled by default under **−std=c++14** and above. The flag **−Wsized−deallocation** warns about places that might want to add a definition.

−fstrict−enums

Allow the compiler to optimize using the assumption that a value of enumerated type can only be one of the values of the enumeration (as defined in the C++ standard; basically, a value that can be represented in the minimum number of bits needed to represent all the enumerators). This assumption may not be valid if the program uses a cast to convert an arbitrary integer value to the enumerated type. This option has no effect for an enumeration type with a fixed underlying type.

−fstrong−eval−order
−fstrong−eval−order=_kind_

Evaluate member access, array subscripting, and shift expressions in left−to−right order, and evaluate assignment in right−to−left order, as adopted for C++17. **−fstrong−eval−order** is equivalent to **−fstrong−eval−order=all**, and is enabled by default with **−std=c++17** or later.

−fstrong−eval−order=some enables just the ordering of member access and shift expressions, and is the default for C++ dialects prior to C++17.

−fstrong−eval−order=none is equivalent to **−fno−strong−eval−order**.

−ftemplate−backtrace−limit=_n_

Set the maximum number of template instantiation notes for a single warning or error to _n_. The default value is 10.

−ftemplate−depth=_n_

Set the maximum instantiation depth for template classes to _n_. A limit on the template instantiation depth is needed to detect endless recursions during template class instantiation. ANSI/ISO C++ conforming programs must not rely on a maximum depth greater than 17 (changed to 1024 in C++11). The default value is 900, as the compiler can run out of stack space before hitting 1024 in some situations.

−fno−threadsafe−statics

Do not emit the extra code to use the routines specified in the C++ ABI for thread−safe initialization of local statics. You can use this option to reduce code size slightly in code that doesn't need to be thread−safe.

−fuse−cxa−atexit

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

−fno−use−cxa−get−exception−ptr

Don't use the `__cxa_get_exception_ptr` runtime routine. This causes `std::uncaught_exception` to be incorrect, but is necessary if the runtime routine is not available.

−fvisibility−inlines−hidden

This switch declares that the user does not attempt to compare pointers to inline functions or methods where the addresses of the two functions are taken in different shared objects.

The effect of this is that GCC may, effectively, mark inline methods with `__attribute__ ((visibility ("hidden")))` so that they do not appear in the export table of a DSO and do not require a PLT indirection when used within the DSO. Enabling this option can have a dramatic effect on load and link times of a DSO as it massively reduces the size of the dynamic export table

when the library makes heavy use of templates.

The behavior of this switch is not quite the same as marking the methods as hidden directly, because it does not affect static variables local to the function or cause the compiler to deduce that the function is defined in only one shared object.

You may mark a method as having a visibility explicitly to negate the effect of the switch for that method. For example, if you do want to compare pointers to a particular inline method, you might mark it as having default visibility. Marking the enclosing class with explicit visibility has no effect.

Explicitly instantiated inline methods are unaffected by this option as their linkage might otherwise cross a shared library boundary.

−fvisibility−ms−compat

This flag attempts to use visibility settings to make GCC's C++ linkage model compatible with that of Microsoft Visual Studio.

The flag makes these changes to GCC's linkage model:

1.   It sets the default visibility to `hidden`, like **−fvisibility=hidden**.

2.   Types, but not their members, are not hidden by default.

3.   The One Definition Rule is relaxed for types without explicit visibility specifications that are defined in more than one shared object: those declarations are permitted if they are permitted when this option is not used.

In new code it is better to use **−fvisibility=hidden** and export those classes that are intended to be externally visible. Unfortunately it is possible for code to rely, perhaps accidentally, on the Visual Studio behavior.

Among the consequences of these changes are that static data members of the same type with the same name but defined in different shared objects are different, so changing one does not change the other; and that pointers to function members defined in different shared objects may not compare equal. When this flag is given, it is a violation of the ODR to define types with the same name differently.

−fno−weak

Do not use weak symbol support, even if it is provided by the linker. By default, G++ uses weak symbols if they are available. This option exists only for testing, and should not be used by end−users; it results in inferior code and has no benefits. This option may be removed in a future release of G++.

−fext−numeric−literals (C++ and Objective−C++ only)

Accept imaginary, fixed−point, or machine−defined literal number suffixes as GNU extensions. When this option is turned off these suffixes are treated as C++11 user−defined literal numeric suffixes. This is on by default for all pre−C++11 dialects and all GNU dialects: **−std=c++98**, **−std=gnu++98**, **−std=gnu++11**, **−std=gnu++14**. This option is off by default for ISO C++11 onwards (**−std=c++11**, ...).

−nostdinc++

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

−flang−info−include−translate
−flang−info−include−translate−not
−flang−info−include−translate=*header*

Inform of include translation events. The first will note accepted include translations, the second will note declined include translations. The *header* form will inform of include translations relating to that specific header. If *header* is of the form `"user"` or `<system>` it will be resolved to a specific user or system header using the include path.

−flang−info−module−cmi

−flang−info−module−cmi=_module_
> Inform of Compiled Module Interface pathnames. The first will note all read CMI pathnames. The _module_ form will not reading a specific module's CMI. _module_ may be a named module or a header−unit (the latter indicated by either being a pathname containing directory separators or enclosed in <> or `""`).

−stdlib=_libstdc++,libc++_
> When G++ is configured to support this option, it allows specification of alternate C++ runtime libraries. Two options are available: _libstdc++_ (the default, native C++ runtime for G++) and _libc++_ which is the C++ runtime installed on some operating systems (e.g. Darwin versions from Darwin11 onwards). The option switches G++ to use the headers from the specified library and to emit `-lstdc++` or `-lc++` respectively, when a C++ runtime is required for linking.

In addition, these warning options have meanings only for C++ programs:

−Wabi−tag (C++ and Objective−C++ only)
> Warn when a type with an ABI tag is used in a context that does not have that ABI tag. See **C++ Attributes** for more information about ABI tags.

−Wcomma−subscript (C++ and Objective−C++ only)
> Warn about uses of a comma expression within a subscripting expression. This usage was deprecated in C++20 and is going to be removed in C++23. However, a comma expression wrapped in ( ) is not deprecated. Example:

```
void f(int *a, int b, int c) {
    a[b,c];      // deprecated in C++20, invalid in C++23
    a[(b,c)];    // OK
}
```

> In C++23 it is valid to have comma separated expressions in a subscript when an overloaded subscript operator is found and supports the right number and types of arguments. G++ will accept the formerly valid syntax for code that is not valid in C++23 but used to be valid but deprecated in C++20 with a pedantic warning that can be disabled with **−Wno−comma−subscript**.

> Enabled by default with **−std=c++20** unless **−Wno−deprecated**, and after **−std=c++23** regardless of **−Wno−deprecated**. Before **−std=c++20**, enabled with explicit **−Wdeprecated**.

> This warning is upgraded to an error by **−pedantic−errors** in C++23 mode or later.

−Wctad−maybe−unsupported (C++ and Objective−C++ only)
> Warn when performing class template argument deduction (CTAD) on a type with no explicitly written deduction guides. This warning will point out cases where CTAD succeeded only because the compiler synthesized the implicit deduction guides, which might not be what the programmer intended. Certain style guides allow CTAD only on types that specifically "opt−in"; i.e., on types that are designed to support CTAD. This warning can be suppressed with the following pattern:

```
struct allow_ctad_t; // any name works
template <typename T> struct S {
  S(T) { }
};
// Guide with incomplete parameter type will never be considered.
S(allow_ctad_t) -> S<void>;
```

−Wctor−dtor−privacy (C++ and Objective−C++ only)
> Warn when a class seems unusable because all the constructors or destructors in that class are private, and it has neither friends nor public static member functions. Also warn if there are no non−private methods, and there's at least one private member function that isn't a constructor or destructor.

−Wdangling−reference (C++ and Objective−C++ only)
> Warn when a reference is bound to a temporary whose lifetime has ended. For example:

```
int n = 1;
const int& r = std::max(n - 1, n + 1); // r is dangling
```

In the example above, two temporaries are created, one for each argument, and a reference to one of the temporaries is returned. However, both temporaries are destroyed at the end of the full expression, so the reference r is dangling. This warning also detects dangling references in member initializer lists:

```
const int& f(const int& i) { return i; }
struct S {
  const int &r; // r is dangling
  S() : r(f(10)) { }
};
```

Member functions are checked as well, but only their object argument:

```
struct S {
    const S& self () { return *this; }
};
const S& s = S().self(); // s is dangling
```

Certain functions are safe in this respect, for example std::use_facet: they take and return a reference, but they don't return one of its arguments, which can fool the warning. Such functions can be excluded from the warning by wrapping them in a #pragma:

```
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wdangling-reference"
const T& foo (const T&) { ... }
#pragma GCC diagnostic pop
```

The #pragma can also surround the class; in that case, the warning will be disabled for all the member functions.

−Wdangling−reference also warns about code like

```
auto p = std::minmax(1, 2);
```

where std::minmax returns std::pair<const int&, const int&>, and both references dangle after the end of the full expression that contains the call to std::minmax.

The warning does not warn for std::span−like classes. We consider classes of the form:

```
template<typename T>
struct Span {
  T* data_;
  std::size len_;
};
```

as std::span−like; that is, the class is a non−union class that has a pointer data member and a trivial destructor.

The warning can be disabled by using the gnu::no_dangling attribute.

This warning is enabled by **−Wextra**.

−Wdelete−non−virtual−dtor (C++ and Objective−C++ only)
Warn when delete is used to destroy an instance of a class that has virtual functions and non−virtual destructor. It is unsafe to delete an instance of a derived class through a pointer to a base class if the base class does not have a virtual destructor. This warning is enabled by **−Wall**.

−Wdeprecated−copy (C++ and Objective−C++ only)
Warn that the implicit declaration of a copy constructor or copy assignment operator is deprecated if the class has a user−provided copy constructor or copy assignment operator, in C++11 and up. This

warning is enabled by **−Wextra**. With **−Wdeprecated−copy−dtor**, also deprecate if the class has a user−provided destructor.

−Wno−deprecated−enum−enum−conversion (C++ and Objective−C++ only)
> Disable the warning about the case when the usual arithmetic conversions are applied on operands where one is of enumeration type and the other is of a different enumeration type. This conversion was deprecated in C++20. For example:
>
> ```
> enum E1 { e };
> enum E2 { f };
> int k = f − e;
> ```
>
> **−Wdeprecated−enum−enum−conversion** is enabled by default with **−std=c++20**. In pre−C++20 dialects, this warning can be enabled by **−Wenum−conversion** or **−Wdeprecated**.

−Wno−deprecated−enum−float−conversion (C++ and Objective−C++ only)
> Disable the warning about the case when the usual arithmetic conversions are applied on operands where one is of enumeration type and the other is of a floating−point type. This conversion was deprecated in C++20. For example:
>
> ```
> enum E1 { e };
> enum E2 { f };
> bool b = e <= 3.7;
> ```
>
> **−Wdeprecated−enum−float−conversion** is enabled by default with **−std=c++20**. In pre−C++20 dialects, this warning can be enabled by **−Wenum−conversion** or **−Wdeprecated**.

−Wdeprecated−literal−operator (C++ and Objective−C++ only)
> Warn that the declaration of a user−defined literal operator with a space before the suffix is deprecated. This warning is enabled by default in C++23, or with explicit **−Wdeprecated**.
>
> ```
> string operator "" _i18n(const char*, std::size_t); // deprecated
> string operator ""_i18n(const char*, std::size_t); // preferred
> ```

−Wdeprecated−variadic−comma−omission (C++ and Objective−C++ only)
> Warn that omitting a comma before the varargs `...` at the end of a function parameter list is deprecated. This warning is enabled by default in C++26, or with explicit **−Wdeprecated**.
>
> ```
> void f1(int...); // deprecated
> void f1(int, ...); // preferred
> template <typename ...T>
> void f2(T...); // ok
> template <typename ...T>
> void f3(T......); // deprecated
> ```

−Wno−elaborated−enum−base
> For C++11 and above, warn if an (invalid) additional enum−base is used in an elaborated−type−specifier. That is, if an enum with given underlying type and no enumerator list is used in a declaration other than just a standalone declaration of the enum. Enabled by default. This warning is upgraded to an error with −pedantic−errors.

−Wno−init−list−lifetime (C++ and Objective−C++ only)
> Do not warn about uses of `std::initializer_list` that are likely to result in dangling pointers. Since the underlying array for an `initializer_list` is handled like a normal C++ temporary object, it is easy to inadvertently keep a pointer to the array past the end of the array's lifetime. For example:
>
> *   If a function returns a temporary `initializer_list`, or a local `initializer_list` variable, the array's lifetime ends at the end of the return statement, so the value returned has a dangling pointer.

* If a new−expression creates an `initializer_list`, the array only lives until the end of the enclosing full−expression, so the `initializer_list` in the heap has a dangling pointer.

* When an `initializer_list` variable is assigned from a brace−enclosed initializer list, the temporary array created for the right side of the assignment only lives until the end of the full−expression, so at the next statement the `initializer_list` variable has a dangling pointer.

```
// li's initial underlying array lives as long as li
std::initializer_list<int> li = { 1,2,3 };
// assignment changes li to point to a temporary array
li = { 4, 5 };
// now the temporary is gone and li has a dangling pointer
int i = li.begin()[0] // undefined behavior
```

* When a list constructor stores the `begin` pointer from the `initializer_list` argument, this doesn't extend the lifetime of the array, so if a class variable is constructed from a temporary `initializer_list`, the pointer is left dangling by the end of the variable declaration statement.

−Winvalid−constexpr

Warn when a function never produces a constant expression. In C++20 and earlier, for every `constexpr` function and function template, there must be at least one set of function arguments in at least one instantiation such that an invocation of the function or constructor could be an evaluated subexpression of a core constant expression. C++23 removed this restriction, so it's possible to have a function or a function template marked `constexpr` for which no invocation satisfies the requirements of a core constant expression.

This warning is enabled as a pedantic warning by default in C++20 and earlier. In C++23, **−Winvalid−constexpr** can be turned on, in which case it will be an ordinary warning. For example:

```
void f (int& i);
constexpr void
g (int& i)
{
  // Warns by default in C++20, in C++23 only with -Winvalid-constexpr.
  f(i);
}
```

−Winvalid−imported−macros

Verify all imported macro definitions are valid at the end of compilation. This is not enabled by default, as it requires additional processing to determine. It may be useful when preparing sets of header−units to ensure consistent macros.

−Wno−literal−suffix (C++ and Objective−C++ only)

Do not warn when a string or character literal is followed by a ud−suffix which does not begin with an underscore. As a conforming extension, GCC treats such suffixes as separate preprocessing tokens in order to maintain backwards compatibility with code that uses formatting macros from `<inttypes.h>`. For example:

```
#define __STDC_FORMAT_MACROS
#include <inttypes.h>
#include <stdio.h>

int main() {
  int64_t i64 = 123;
  printf("My int64: %" PRId64"\n", i64);
}
```

In this case, `PRId64` is treated as a separate preprocessing token.

This option also controls warnings when a user−defined literal operator is declared with a literal suffix identifier that doesn't begin with an underscore. Literal suffix identifiers that don't begin with an underscore are reserved for future standardization.

These warnings are enabled by default.

−Wno−narrowing (C++ and Objective−C++ only)

For C++11 and later standards, narrowing conversions are diagnosed by default, as required by the standard. A narrowing conversion from a constant produces an error, and a narrowing conversion from a non−constant produces a warning, but **−Wno−narrowing** suppresses the diagnostic. Note that this does not affect the meaning of well−formed code; narrowing conversions are still considered ill−formed in SFINAE contexts.

With **−Wnarrowing** in C++98, warn when a narrowing conversion prohibited by C++11 occurs within { }, e.g.

```
int i = { 2.2 }; // error: narrowing from double to int
```

This flag is included in **−Wall** and **−Wc++11−compat**.

−Wnoexcept (C++ and Objective−C++ only)

Warn when a noexcept−expression evaluates to false because of a call to a function that does not have a non−throwing exception specification (i.e. `throw()` or `noexcept`) but is known by the compiler to never throw an exception.

−Wnoexcept−type (C++ and Objective−C++ only)

Warn if the C++17 feature making `noexcept` part of a function type changes the mangled name of a symbol relative to C++14. Enabled by **−Wabi** and **−Wc++17−compat**.

As an example:

```
template <class T> void f(T t) { t(); };
void g() noexcept;
void h() { f(g); }
```

In C++14, `f` calls `f<void(*)()>`, but in C++17 it calls `f<void(*)()noexcept>`.

−Wclass−memaccess (C++ and Objective−C++ only)

Warn when the destination of a call to a raw memory function such as `memset` or `memcpy` is an object of class type, and when writing into such an object might bypass the class non−trivial or deleted constructor or copy assignment, violate const−correctness or encapsulation, or corrupt virtual table pointers. Modifying the representation of such objects may violate invariants maintained by member functions of the class. For example, the call to `memset` below is undefined because it modifies a non−trivial class object and is, therefore, diagnosed. The safe way to either initialize or clear the storage of objects of such types is by using the appropriate constructor or assignment operator, if one is available.

```
std::string str = "abc";
memset (&str, 0, sizeof str);
```

The **−Wclass−memaccess** option is enabled by **−Wall**. Explicitly casting the pointer to the class object to `void *` or to a type that can be safely accessed by the raw memory function suppresses the warning.

−Wnon−virtual−dtor (C++ and Objective−C++ only)

Warn when a class has virtual functions and an accessible non−virtual destructor itself or in an accessible polymorphic base class, in which case it is possible but unsafe to delete an instance of a derived class through a pointer to the class itself or base class. This warning is automatically enabled if **−Weffc++** is specified. The **−Wdelete−non−virtual−dtor** option (enabled by **−Wall**) should be preferred because it warns about the unsafe cases without false positives.

−Wregister (C++ and Objective−C++ only)

Warn on uses of the `register` storage class specifier, except when it is part of the GNU **Explicit Register Variables** extension. The use of the `register` keyword as storage class specifier has been deprecated in C++11 and removed in C++17. Enabled by default with **−std=c++17**.

−Wreorder (C++ and Objective−C++ only)

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

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

The compiler rearranges the member initializers for `i` and `j` to match the declaration order of the members, emitting a warning to that effect. This warning is enabled by **−Wall**.

−Wno−pessimizing−move (C++ and Objective−C++ only)

This warning warns when a call to `std::move` prevents copy elision. A typical scenario when copy elision can occur is when returning in a function with a class return type, when the expression being returned is the name of a non−volatile automatic object, and is not a function parameter, and has the same type as the function return type.

```
struct T {
...
};
T fn()
{
  T t;
  ...
  return std::move (t);
}
```

But in this example, the `std::move` call prevents copy elision.

This warning is enabled by **−Wall**.

−Wno−redundant−move (C++ and Objective−C++ only)

This warning warns about redundant calls to `std::move`; that is, when a move operation would have been performed even without the `std::move` call. This happens because the compiler is forced to treat the object as if it were an rvalue in certain situations such as returning a local variable, where copy elision isn't applicable. Consider:

```
struct T {
...
};
T fn(T t)
{
  ...
  return std::move (t);
}
```

Here, the `std::move` call is redundant. Because G++ implements Core Issue 1579, another example is:

```
struct T { // convertible to U
...
};
struct U {
...
};
U fn()
{
  T t;
  ...
  return std::move (t);
}
```

In this example, copy elision isn't applicable because the type of the expression being returned and the function return type differ, yet G++ treats the return value as if it were designated by an rvalue.

This warning is enabled by **−Wextra**.

−Wrange−loop−construct (C++ and Objective−C++ only)

This warning warns when a C++ range−based for−loop is creating an unnecessary copy. This can happen when the range declaration is not a reference, but probably should be. For example:

```
struct S { char arr[128]; };
void fn () {
  S arr[5];
  for (const auto x : arr) { ... }
}
```

It does not warn when the type being copied is a trivially−copyable type whose size is less than 64 bytes.

This warning also warns when a loop variable in a range−based for−loop is initialized with a value of a different type resulting in a copy. For example:

```
void fn() {
  int arr[10];
  for (const double &x : arr) { ... }
}
```

In the example above, in every iteration of the loop a temporary value of type double is created and destroyed, to which the reference const double & is bound.

This warning is enabled by **−Wall**.

−Wredundant−tags (C++ and Objective−C++ only)

Warn about redundant class−key and enum−key in references to class types and enumerated types in contexts where the key can be eliminated without causing an ambiguity. For example:

```
struct foo;
struct foo *p;   // warn that keyword struct can be eliminated
```

On the other hand, in this example there is no warning:

```
struct foo;
void foo ();   // "hides" struct foo
void bar (struct foo&);  // no warning, keyword struct is necessary
```

−Wno−subobject−linkage (C++ and Objective−C++ only)

Do not warn if a class type has a base or a field whose type uses the anonymous namespace or depends on a type with no linkage. If a type A depends on a type B with no or internal linkage, defining it in multiple translation units would be an ODR violation because the meaning of B is different in each translation unit. If A only appears in a single translation unit, the best way to silence the warning is to

give it internal linkage by putting it in an anonymous namespace as well. The compiler doesn't give this warning for types defined in the main .C file, as those are unlikely to have multiple definitions. **−Wsubobject−linkage** is enabled by default.

−Weffc++ (C++ and Objective−C++ only)
>   Warn about violations of the following style guidelines from Scott Meyers' *Effective C++* series of books:

*   Define a copy constructor and an assignment operator for classes with dynamically−allocated memory.

*   Prefer initialization to assignment in constructors.

*   Have `operator=` return a reference to `*this`.

*   Don't try to return a reference when you must return an object.

*   Distinguish between prefix and postfix forms of increment and decrement operators.

*   Never overload `&&`, `||`, or `,`.

>   This option also enables **−Wnon−virtual−dtor**, which is also one of the effective C++ recommendations. However, the check is extended to warn about the lack of virtual destructor in accessible non−polymorphic bases classes too.

>   When selecting this option, be aware that the standard library headers do not obey all of these guidelines; use **grep −v** to filter out those warnings.

−Wno−exceptions (C++ and Objective−C++ only)
>   Disable the warning about the case when an exception handler is shadowed by another handler, which can point out a wrong ordering of exception handlers.

−Wstrict−null−sentinel (C++ and Objective−C++ only)
>   Warn about the use of an uncasted NULL as sentinel. When compiling only with GCC this is a valid sentinel, as NULL is defined to `__null`. Although it is a null pointer constant rather than a null pointer, it is guaranteed to be of the same size as a pointer. But this use is not portable across different compilers.

−Wno−non−template−friend (C++ and Objective−C++ only)
>   Disable warnings when non−template friend functions are declared within a template. In very old versions of GCC that predate implementation of the ISO standard, declarations such as **friend int foo(int)**, where the name of the friend is an unqualified−id, could be interpreted as a particular specialization of a template function; the warning exists to diagnose compatibility problems, and is enabled by default.

−Wold−style−cast (C++ and Objective−C++ only)
>   Warn if an old−style (C−style) cast to a non−void type is used within a C++ program. The new−style casts (`dynamic_cast`, `static_cast`, `reinterpret_cast`, and `const_cast`) are less vulnerable to unintended effects and much easier to search for.

−Woverloaded−virtual (C++ and Objective−C++ only)
−Woverloaded−virtual=n
>   Warn when a function declaration hides virtual functions from a base class. For example, in:

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

struct B: public A {
  void f(int); // does not override
};
```

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

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

fails to compile.

In cases where the different signatures are not an accident, the simplest solution is to add a using−declaration to the derived class to un−hide the base function, e.g. add using A::f; to B.

The optional level suffix controls the behavior when all the declarations in the derived class override virtual functions in the base class, even if not all of the base functions are overridden:

```
struct C {
  virtual void f();
  virtual void f(int);
};

struct D: public C {
  void f(int); // does override
}
```

This pattern is less likely to be a mistake; if D is only used virtually, the user might have decided that the base class semantics for some of the overloads are fine.

At level 1, this case does not warn; at level 2, it does. **−Woverloaded−virtual** by itself selects level 2. Level 1 is included in **−Wall**.

−Wno−pmf−conversions (C++ and Objective−C++ only)
 Disable the diagnostic for converting a bound pointer to member function to a plain pointer.

−Wsign−promo (C++ and Objective−C++ only)
 Warn when overload resolution chooses a promotion from unsigned or enumerated type to a signed type, over a conversion to an unsigned type of the same size. Previous versions of G++ tried to preserve unsignedness, but the standard mandates the current behavior.

−Wtemplates (C++ and Objective−C++ only)
 Warn when a primary template declaration is encountered. Some coding rules disallow templates, and this may be used to enforce that rule. The warning is inactive inside a system header file, such as the STL, so one can still use the STL. One may also instantiate or specialize templates.

−Wmismatched−new−delete (C++ and Objective−C++ only)
 Warn for mismatches between calls to operator new or operator delete and the corresponding call to the allocation or deallocation function. This includes invocations of C++ operator delete with pointers returned from either mismatched forms of operator new, or from other functions that allocate objects for which the operator delete isn't a suitable deallocator, as well as calls to other deallocation functions with pointers returned from operator new for which the deallocation function isn't suitable.

 For example, the delete expression in the function below is diagnosed because it doesn't match the array form of the new expression the pointer argument was returned from. Similarly, the call to free is also diagnosed.

```
void f ()
{
  int *a = new int[n];
  delete a;   // warning: mismatch in array forms of expressions

  char *p = new char[n];
  free (p);   // warning: mismatch between new and free
}
```

 The related option **−Wmismatched−dealloc** diagnoses mismatches involving allocation and

deallocation functions other than `operator new` and `operator delete`.

−Wmismatched−new−delete is included in **−Wall**.

−Wmismatched−tags (C++ and Objective−C++ only)
Warn for declarations of structs, classes, and class templates and their specializations with a class−key that does not match either the definition or the first declaration if no definition is provided.

For example, the declaration of `struct Object` in the argument list of `draw` triggers the warning. To avoid it, either remove the redundant class−key `struct` or replace it with `class` to match its definition.

```
class Object {
public:
  virtual ˜Object () = 0;
};
void draw (struct Object*);
```

It is not wrong to declare a class with the class−key `struct` as the example above shows. The **−Wmismatched−tags** option is intended to help achieve a consistent style of class declarations. In code that is intended to be portable to Windows−based compilers the warning helps prevent unresolved references due to the difference in the mangling of symbols declared with different class−keys. The option can be used either on its own or in conjunction with **−Wredundant−tags**.

−Wmultiple−inheritance (C++ and Objective−C++ only)
Warn when a class is defined with multiple direct base classes. Some coding rules disallow multiple inheritance, and this may be used to enforce that rule. The warning is inactive inside a system header file, such as the STL, so one can still use the STL. One may also define classes that indirectly use multiple inheritance.

−Wvirtual−inheritance
Warn when a class is defined with a virtual direct base class. Some coding rules disallow multiple inheritance, and this may be used to enforce that rule. The warning is inactive inside a system header file, such as the STL, so one can still use the STL. One may also define classes that indirectly use virtual inheritance.

−Wno−virtual−move−assign
Suppress warnings about inheriting from a virtual base with a non−trivial C++11 move assignment operator. This is dangerous because if the virtual base is reachable along more than one path, it is moved multiple times, which can mean both objects end up in the moved−from state. If the move assignment operator is written to avoid moving from a moved−from object, this warning can be disabled.

−Wnamespaces
Warn when a namespace definition is opened. Some coding rules disallow namespaces, and this may be used to enforce that rule. The warning is inactive inside a system header file, such as the STL, so one can still use the STL. One may also use using directives and qualified names.

−Wno−template−body (C++ and Objective−C++ only)
Disable diagnosing errors when parsing a template, and instead issue an error only upon instantiation of the template. This flag can also be used to downgrade such errors into warnings with **Wno−error=** or **−fpermissive**.

−Wno−template−id−cdtor (C++ and Objective−C++ only)
Disable the warning about the use of simple−template−id as the declarator−id of a constructor or destructor, which became invalid in C++20 via DR 2237. For example:

```
template<typename T> struct S {
  S<T>(); // should be S();
  ˜S<T>();  // should be ˜S();
};
```

−Wtemplate−id−cdtor is enabled by default with **−std=c++20**; it is also enabled by **−Wc++20−compat**.

−Wtemplate−names−tu−local

    Warn when a template body hides an exposure of a translation−unit−local entity. In most cases, referring to a translation−unit−local entity (such as an internal linkage declaration) within an entity that is emitted into a module's CMI is an error. However, within the initializer of a variable, or in the body of a non−inline function, this is not an exposure and no error is emitted.

    This can cause variable or function templates to accidentally become unusable if they reference such an entity, because other translation units that import the template will never be able to instantiate it. This warning attempts to detect cases where this might occur. The presence of an explicit instantiation silences the warning.

    This flag is enabled by **−Wextra**.

−Wno−terminate (C++ and Objective−C++ only)

    Disable the warning about a throw−expression that will immediately result in a call to `terminate`.

−Wno−vexing−parse (C++ and Objective−C++ only)

    Warn about the most vexing parse syntactic ambiguity. This warns about the cases when a declaration looks like a variable definition, but the C++ language requires it to be interpreted as a function declaration. For instance:

```
void f(double a) {
  int i();        // extern int i (void);
  int n(int(a));  // extern int n (int);
}
```

Another example:

```
struct S { S(int); };
void f(double a) {
  S x(int(a));   // extern struct S x (int);
  S y(int());    // extern struct S y (int (*) (void));
  S z();         // extern struct S z (void);
}
```

    The warning will suggest options how to deal with such an ambiguity; e.g., it can suggest removing the parentheses or using braces instead.

    This warning is enabled by default.

−Wno−class−conversion (C++ and Objective−C++ only)

    Do not warn when a conversion function converts an object to the same type, to a base class of that type, or to void; such a conversion function will never be called.

−Wvolatile (C++ and Objective−C++ only)

    Warn about deprecated uses of the `volatile` qualifier. This includes postfix and prefix ++ and −− expressions of `volatile`−qualified types, using simple assignments where the left operand is a `volatile`−qualified non−class type for their value, compound assignments where the left operand is a `volatile`−qualified non−class type, `volatile`−qualified function return type, `volatile`−qualified parameter type, and structured bindings of a `volatile`−qualified type. This usage was deprecated in C++20.

    Enabled by default with **−std=c++20**. Before **−std=c++20**, enabled with explicit **−Wdeprecated**.

−Waligned−new

    Warn about a new−expression of a type that requires greater alignment than the `alignof(std::max_align_t)` but uses an allocation function without an explicit alignment parameter. This option is enabled by **−Wall**.

    Normally this only warns about global allocation functions, but **−Waligned−new=all** also warns about class member allocation functions.

−Wno−placement−new

−Wplacement−new=*n*

    Warn about placement new expressions with undefined behavior, such as constructing an object in a buffer that is smaller than the type of the object. For example, the placement new expression below is diagnosed because it attempts to construct an array of 64 integers in a buffer only 64 bytes large.

```
char buf [64];
new (buf) int[64];
```

This warning is enabled by default.

    −Wplacement−new=1

        This is the default warning level of **−Wplacement−new**. At this level the warning is not issued for some strictly undefined constructs that GCC allows as extensions for compatibility with legacy code. For example, the following new expression is not diagnosed at this level even though it has undefined behavior according to the C++ standard because it writes past the end of the one−element array.

```
struct S { int n, a[1]; };
S *s = (S *)malloc (sizeof *s + 31 * sizeof s->a[0]);
new (s->a)int [32]();
```

    −Wplacement−new=2

        At this level, in addition to diagnosing all the same constructs as at level 1, a diagnostic is also issued for placement new expressions that construct an object in the last member of structure whose type is an array of a single element and whose size is less than the size of the object being constructed. While the previous example would be diagnosed, the following construct makes use of the flexible member array extension to avoid the warning at level 2.

```
struct S { int n, a[]; };
S *s = (S *)malloc (sizeof *s + 32 * sizeof s->a[0]);
new (s->a)int [32]();
```

−Wcatch−value

−Wcatch−value=*n* (C++ and Objective−C++ only)

    Warn about catch handlers that do not catch via reference. With **−Wcatch−value=1** (or **−Wcatch−value** for short) warn about polymorphic class types that are caught by value. With **−Wcatch−value=2** warn about all class types that are caught by value. With **−Wcatch−value=3** warn about all types that are not caught by reference. **−Wcatch−value** is enabled by **−Wall**.

−Wconditionally−supported (C++ and Objective−C++ only)

    Warn for conditionally−supported (C++11 [intro.defs]) constructs.

−Wno−defaulted−function−deleted (C++ and Objective−C++ only)

    Warn when an explicitly defaulted function is deleted by the compiler. That can occur when the function's declared type does not match the type of the function that would have been implicitly declared. This warning is enabled by default.

−Wno−delete−incomplete (C++ and Objective−C++ only)

    Do not warn when deleting a pointer to incomplete type, which may cause undefined behavior at runtime. This warning is enabled by default.

−Wextra−semi (C++, Objective−C++ only)
> Warn about redundant semicolons. There are various contexts in which an extra semicolon can occur. One is a semicolon after in−class function definitions, which is valid in all C++ dialects (and is never a pedwarn):

```
struct S {
  void foo () {};
};
```

> Another is an extra semicolon at namespace scope, which has been allowed since C++11 (therefore is a pedwarn in C++98):

```
struct S {
};
;
```

> And yet another is an extra semicolon in class definitions, which has been allowed since C++11 (therefore is a pedwarn in C++98):

```
struct S {
  int a;
  ;
};
```

−Wno−global−module (C++ and Objective−C++ only)
> Disable the diagnostic for when the global module fragment of a module unit does not consist only of preprocessor directives.

−Wno−inaccessible−base (C++, Objective−C++ only)
> This option controls warnings when a base class is inaccessible in a class derived from it due to ambiguity. The warning is enabled by default. Note that the warning for ambiguous virtual bases is enabled by the **−Wextra** option.

```
struct A { int a; };

struct B : A { };

struct C : B, A { };
```

−Wno−inherited−variadic−ctor
> Suppress warnings about use of C++11 inheriting constructors when the base class inherited from has a C variadic constructor; the warning is on by default because the ellipsis is not inherited.

−Wno−invalid−offsetof (C++ and Objective−C++ only)
> Suppress warnings from applying the `offsetof` macro to a non−POD type. According to the 2014 ISO C++ standard, applying `offsetof` to a non−standard−layout type is undefined. In existing C++ implementations, however, `offsetof` typically gives meaningful results. This flag is for users who are aware that they are writing nonportable code and who have deliberately chosen to ignore the warning about it.

> The restrictions on `offsetof` may be relaxed in a future version of the C++ standard.

−Wsized−deallocation (C++ and Objective−C++ only)
> Warn about a definition of an unsized deallocation function

```
void operator delete (void *) noexcept;
void operator delete[] (void *) noexcept;
```

> without a definition of the corresponding sized deallocation function

```
void operator delete (void *, std::size_t) noexcept;
void operator delete[] (void *, std::size_t) noexcept;
```

or vice versa. Enabled by **−Wextra** along with **−fsized−deallocation**.

−Wsuggest−final−types
>Warn about types with virtual methods where code quality would be improved if the type were declared with the C++11 `final` specifier, or, if possible, declared in an anonymous namespace. This allows GCC to more aggressively devirtualize the polymorphic calls. This warning is more effective with link−time optimization, where the information about the class hierarchy graph is more complete.

−Wsuggest−final−methods
>Warn about virtual methods where code quality would be improved if the method were declared with the C++11 `final` specifier, or, if possible, its type were declared in an anonymous namespace or with the `final` specifier. This warning is more effective with link−time optimization, where the information about the class hierarchy graph is more complete. It is recommended to first consider suggestions of **−Wsuggest−final−types** and then rebuild with new annotations.

−Wsuggest−override
>Warn about overriding virtual functions that are not marked with the `override` keyword.

−Wno−conversion−null (C++ and Objective−C++ only)
>Do not warn for conversions between `NULL` and non−pointer types. **−Wconversion−null** is enabled by default.

Options Controlling Objective−C and Objective−C++ Dialects
>(NOTE: This manual does not describe the Objective−C and Objective−C++ languages themselves.

>This section describes the command−line options that are only meaningful for Objective−C and Objective−C++ programs. You can also use most of the language−independent GNU compiler options. For example, you might compile a file *some_class.m* like this:

>   gcc −g −fgnu−runtime −O −c some_class.m

>In this example, **−fgnu−runtime** is an option meant only for Objective−C and Objective−C++ programs; you can use the other options with any language supported by GCC.

>Note that since Objective−C is an extension of the C language, Objective−C compilations may also use options specific to the C front−end (e.g., **−Wtraditional**). Similarly, Objective−C++ compilations may use C++−specific options (e.g., **−Wabi**).

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

−fconstant−string−class=<i>class−name</i>
>Use *class−name* as the name of the class to instantiate for each literal string specified with the syntax `@"..."`. The default class name is `NXConstantString` if the GNU runtime is being used, and `NSConstantString` if the NeXT runtime is being used (see below). On Darwin / macOS platforms, the **−fconstant−cfstrings** option, if also present, overrides the **−fconstant−string−class** setting and cause `@"..."` literals to be laid out as constant CoreFoundation strings. Note that **−fconstant−cfstrings** is an alias for the target−specific **−mconstant−cfstrings** equivalent.

−fgnu−runtime
>Generate object code compatible with the standard GNU Objective−C runtime. This is the default for most types of systems.

−fnext−runtime
>Generate output compatible with the NeXT runtime. This is the default for NeXT−based systems, including Darwin / macOS. The macro `__NEXT_RUNTIME__` is predefined if (and only if) this option is used.

−fno−nil−receivers
>Assume that all Objective−C message dispatches (`[receiver message:arg]`) in this translation unit ensure that the receiver is not `nil`. This allows for more efficient entry points in the runtime to be used. This option is only available in conjunction with the NeXT runtime and ABI version 0 or 1.

−fobjc−abi−version=*n*

Use version *n* of the Objective−C ABI for the selected runtime. This option is currently supported only for the NeXT runtime. In that case, Version 0 is the traditional (32−bit) ABI without support for properties and other Objective−C 2.0 additions. Version 1 is the traditional (32−bit) ABI with support for properties and other Objective−C 2.0 additions. Version 2 is the modern (64−bit) ABI. If nothing is specified, the default is Version 0 on 32−bit target machines, and Version 2 on 64−bit target machines.

−fobjc−call−cxx−cdtors

For each Objective−C class, check if any of its instance variables is a C++ object with a non−trivial default constructor. If so, synthesize a special − (id) .cxx_construct instance method which runs non−trivial default constructors on any such instance variables, in order, and then return self. Similarly, check if any instance variable is a C++ object with a non−trivial destructor, and if so, synthesize a special − (void) .cxx_destruct method which runs all such default destructors, in reverse order.

The − (id) .cxx_construct and − (void) .cxx_destruct methods thusly generated only operate on instance variables declared in the current Objective−C class, and not those inherited from superclasses. It is the responsibility of the Objective−C runtime to invoke all such methods in an object's inheritance hierarchy. The − (id) .cxx_construct methods are invoked by the runtime immediately after a new object instance is allocated; the − (void) .cxx_destruct methods are invoked immediately before the runtime deallocates an object instance.

As of this writing, only the NeXT runtime on Mac OS X 10.4 and later has support for invoking the − (id) .cxx_construct and − (void) .cxx_destruct methods.

−fobjc−direct−dispatch

Allow fast jumps to the message dispatcher. On Darwin this is accomplished via the comm page.

−fobjc−exceptions

Enable syntactic support for structured exception handling in Objective−C, similar to what is offered by C++. This option is required to use the Objective−C keywords @try, @throw, @catch, @finally and @synchronized. This option is available with both the GNU runtime and the NeXT runtime (but not available in conjunction with the NeXT runtime on Mac OS X 10.2 and earlier).

−fobjc−gc

Enable garbage collection (GC) in Objective−C and Objective−C++ programs. This option is only available with the NeXT runtime; the GNU runtime has a different garbage collection implementation that does not require special compiler flags.

−fobjc−nilcheck

For the NeXT runtime with version 2 of the ABI, check for a nil receiver in method invocations before doing the actual method call. This is the default and can be disabled using **−fno−objc−nilcheck**. Class methods and super calls are never checked for nil in this way no matter what this flag is set to. Currently this flag does nothing when the GNU runtime, or an older version of the NeXT runtime ABI, is used.

−fobjc−std=objc1

Conform to the language syntax of Objective−C 1.0, the language recognized by GCC 4.0. This only affects the Objective−C additions to the C/C++ language; it does not affect conformance to C/C++ standards, which is controlled by the separate C/C++ dialect option flags. When this option is used with the Objective−C or Objective−C++ compiler, any Objective−C syntax that is not recognized by GCC 4.0 is rejected. This is useful if you need to make sure that your Objective−C code can be compiled with older versions of GCC.

−freplace−objc−classes

Emit a special marker instructing **ld**(**1**) not to statically link in the resulting object file, and allow **dyld**(**1**) to load it in at run time instead. This is used in conjunction with the Fix−and−Continue debugging mode, where the object file in question may be recompiled and dynamically reloaded in the

course of program execution, without the need to restart the program itself. Currently, Fix−and−Continue functionality is only available in conjunction with the NeXT runtime on Mac OS X 10.3 and later.

−fzero−link

When compiling for the NeXT runtime, the compiler ordinarily replaces calls to `objc_getClass("...")` (when the name of the class is known at compile time) with static class references that get initialized at load time, which improves run−time performance. Specifying the **−fzero−link** flag suppresses this behavior and causes calls to `objc_getClass("...")` to be retained. This is useful in Zero−Link debugging mode, since it allows for individual class implementations to be modified during program execution. The GNU runtime currently always retains calls to `objc_get_class("...")` regardless of command−line options.

−fno−local−ivars

By default instance variables in Objective−C can be accessed as if they were local variables from within the methods of the class they're declared in. This can lead to shadowing between instance variables and other variables declared either locally inside a class method or globally with the same name. Specifying the **−fno−local−ivars** flag disables this behavior thus avoiding variable shadowing issues.

−fivar−visibility=[public|protected|private|package]

Set the default instance variable visibility to the specified option so that instance variables declared outside the scope of any access modifier directives default to the specified visibility.

−gen−decls

Dump interface declarations for all classes seen in the source file to a file named *sourcename.decl*.

−Wassign−intercept (Objective−C and Objective−C++ only)

Warn whenever an Objective−C assignment is being intercepted by the garbage collector.

−Wno−property−assign−default (Objective−C and Objective−C++ only)

Do not warn if a property for an Objective−C object has no assign semantics specified.

−Wno−protocol (Objective−C and Objective−C++ only)

If a class is declared to implement a protocol, a warning is issued for every method in the protocol that is not implemented by the class. The default behavior is to issue a warning for every method not explicitly implemented in the class, even if a method implementation is inherited from the superclass. If you use the **−Wno−protocol** option, then methods inherited from the superclass are considered to be implemented, and no warning is issued for them.

−Wobjc−root−class (Objective−C and Objective−C++ only)

Warn if a class interface lacks a superclass. Most classes will inherit from `NSObject` (or `Object`) for example. When declaring classes intended to be root classes, the warning can be suppressed by marking their interfaces with `__attribute__((objc_root_class))`.

−Wselector (Objective−C and Objective−C++ only)

Warn if multiple methods of different types for the same selector are found during compilation. The check is performed on the list of methods in the final stage of compilation. Additionally, a check is performed for each selector appearing in a `@selector(...)` expression, and a corresponding method for that selector has been found during compilation. Because these checks scan the method table only at the end of compilation, these warnings are not produced if the final stage of compilation is not reached, for example because an error is found during compilation, or because the **−fsyntax−only** option is being used.

−Wstrict−selector−match (Objective−C and Objective−C++ only)

Warn if multiple methods with differing argument and/or return types are found for a given selector when attempting to send a message using this selector to a receiver of type `id` or `Class`. When this flag is off (which is the default behavior), the compiler omits such warnings if any differences found are confined to types that share the same size and alignment.

−Wundeclared−selector (Objective−C and Objective−C++ only)

Warn if a `@selector(...)` expression referring to an undeclared selector is found. A selector is considered undeclared if no method with that name has been declared before the `@selector(...)` expression, either explicitly in an `@interface` or `@protocol` declaration, or implicitly in an `@implementation` section. This option always performs its checks as soon as a `@selector(...)` expression is found, while **−Wselector** only performs its checks in the final stage of compilation. This also enforces the coding style convention that methods and selectors must be declared before being used.

−print−objc−runtime−info

Generate C header describing the largest structure that is passed by value, if any.

Options Controlling OpenMP and OpenACC

GCC supports OpenMP extensions to the C, C++, and Fortran languages with the **−fopenmp** option. Similarly, OpenACC extensions are supported in all three languages with **−fopenacc**.

−foffload=disable
−foffload=default
−foffload=_target−list_

Specify for which OpenMP and OpenACC offload targets code should be generated. The default behavior, equivalent to **−foffload=default**, is to generate code for all supported offload targets. The **−foffload=disable** form generates code only for the host fallback, while **−foffload=**_target−list_ generates code only for the specified comma−separated list of offload targets.

Offload targets are specified in GCC's internal target−triplet format. You can run the compiler with **−v** to show the list of configured offload targets under `OFFLOAD_TARGET_NAMES`.

−foffload−options=_options_
−foffload−options=_target−triplet−list_**=**_options_

With **−foffload−options=**_options_, GCC passes the specified _options_ to the compilers for all enabled offloading targets. You can specify options that apply only to a specific target or targets by using the **−foffload−options=**_target−list_**=**_options_ form. The _target−list_ is a comma−separated list in the same format as for the **−foffload=** option.

Typical command lines are

```
-foffload-options='-fno-math-errno -ffinite-math-only' \
        -foffload-options=nvptx-none=-latomic
-foffload-options=amdgcn-amdhsa=-march=gfx906
```

−fopenacc

Enable handling of OpenACC directives **#pragma acc** in C/C++ and **!$acc** in free−form Fortran and **!$acc**, **c$acc** and ***$acc** in fixed−form Fortran. This option implies **−pthread**, and thus is only supported on targets that have support for **−pthread**.

−fopenacc−dim=_geom_

Specify default compute dimensions for parallel offload regions that do not explicitly specify them. The _geom_ value is a triple of **:**−separated sizes, in order _gang_, _worker_, and _vector_. A size can be omitted, to use a target−specific default value.

−fopenmp

Enable handling of OpenMP directives **#pragma omp**, **[[omp::directive(...)]]**, **[[omp::decl(...)]]**, and **[[omp::sequence(...)]]** in C/C++. In Fortran, it enables **!$omp** and the conditional compilation sentinel **!$**. In fixed source form Fortran, the sentinels can also start with **c** or *****.

This option implies **−pthread**, and thus is only supported on targets that have support for **−pthread**. **−fopenmp** implies **−fopenmp−simd**.

−fopenmp−simd

Enable handling of OpenMP's `simd`, `declare simd`, `declare reduction`, `assume`, `ordered`, `scan` and `loop` directive, and of combined or composite directives with `simd` as

constituent   with   #pragma   omp,   [[omp::directive(...)]],
[[omp::sequence(...)]] and [[omp::decl(...)]] in C/C++ and !$omp in Fortran. It
additionally enables the conditional compilation sentinel **!$** in Fortran. In fixed source form Fortran,
the sentinels can also start with **c** or *****. Other OpenMP directives are ignored. Unless **−fopenmp** is
additionally specified, the loop region binds to the current task region, independent of the specified
bind clause.

−fopenmp−target−simd−clone
−fopenmp−target−simd−clone=*device−type*
> In addition to generating SIMD clones for functions marked with the declare simd directive,
> GCC also generates clones for functions marked with the OpenMP declare target directive that
> are suitable for vectorization when this option is in effect. The *device−type* may be one of none,
> host, nohost, and any, which correspond to keywords for the device_type clause of the
> declare target directive; clones are generated for the intersection of devices specified.
> **−fopenmp−target−simd−clone** is equivalent to **−fopenmp−target−simd−clone=any** and
> **−fno−openmp−target−simd−clone** is equivalent to **−fopenmp−target−simd−clone=none**.

> At **−O2** and higher (but not **−Os** or **−Og**) this optimization defaults to
> **−fopenmp−target−simd−clone=nohost**; otherwise it is disabled by default.

Options to Control Diagnostic Messages Formatting
Traditionally, diagnostic messages have been formatted irrespective of the output device's aspect (e.g. its
width, ...). You can use the options described below to control the formatting algorithm for diagnostic
messages, e.g. how many characters per line, how often source location information should be reported.
Note that some language front ends may not honor these options.

−fmessage−length=*n*
> Try to format error messages so that they fit on lines of about *n* characters. If *n* is zero, then no
> line−wrapping is done; each error message appears on a single line. This is the default for all front
> ends.

> Note − this option also affects the display of the **#error** and **#warning** pre−processor directives, and
> the **deprecated** function/type/variable attribute. It does not however affect the **pragma GCC**
> **warning** and **pragma GCC error** pragmas.

−fdiagnostics−plain−output
> This option requests that diagnostic output look as plain as possible, which may be useful when
> running **dejagnu** or other utilities that need to parse diagnostics output and prefer that it remain more
> stable over time. **−fdiagnostics−plain−output** is currently equivalent to the following options:
> **−fno−diagnostics−show−caret −fno−diagnostics−show−line−numbers −fdiagnostics−color=never**
> **−fdiagnostics−urls=never   −fdiagnostics−path−format=separate−events**
> **−fdiagnostics−text−art−charset=none −fno−diagnostics−show−event−links** In the future, if GCC
> changes the default appearance of its diagnostics, the corresponding option to disable the new behavior
> will be added to this list.

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

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

−fdiagnostics−color[=*WHEN***]**
−fno−diagnostics−color
> Use color in diagnostics. *WHEN* is **never**, **always**, or **auto**. The default depends on how the compiler
> has been configured, it can be any of the above *WHEN* options or also **never** if **GCC_COLORS**

environment variable isn't present in the environment, and **auto** otherwise. **auto** makes GCC use color only when the standard error is a terminal, and when not executing in an emacs shell. The forms **−fdiagnostics−color** and **−fno−diagnostics−color** are aliases for **−fdiagnostics−color=always** and **−fdiagnostics−color=never**, respectively.

The colors are defined by the environment variable **GCC_COLORS**. Its value is a colon−separated list of capabilities and Select Graphic Rendition (SGR) substrings. SGR commands are interpreted by the terminal or terminal emulator. (See the section in the documentation of your text terminal for permitted values and their meanings as character attributes.) These substring values are integers in decimal representation and can be concatenated with semicolons. Common values to concatenate include **1** for bold, **4** for underline, **5** for blink, **7** for inverse, **39** for default foreground color, **30** to **37** for foreground colors, **90** to **97** for 16−color mode foreground colors, **38;5;0** to **38;5;255** for 88−color and 256−color modes foreground colors, **49** for default background color, **40** to **47** for background colors, **100** to **107** for 16−color mode background colors, and **48;5;0** to **48;5;255** for 88−color and 256−color modes background colors.

The default **GCC_COLORS** is

```
error=01;31:warning=01;35:note=01;36:range1=32:range2=34:locus=01:\
quote=01:path=01;36:fixit-insert=32:fixit-delete=31:\
diff-filename=01:diff-hunk=32:diff-delete=31:diff-insert=32:\
type-diff=01;32:fnname=01;32:targs=35:valid=01;31:invalid=01;32\
highlight-a=01;32:highlight-b=01;34
```

where **01;31** is bold red, **01;35** is bold magenta, **01;36** is bold cyan, **32** is green, **34** is blue, **01** is bold, and **31** is red. Setting **GCC_COLORS** to the empty string disables colors. Supported capabilities are as follows.

error=
    SGR substring for error: markers.

warning=
    SGR substring for warning: markers.

note=
    SGR substring for note: markers.

path=
    SGR   substring   for   colorizing   paths   of   control−flow   events   as   printed   via **−fdiagnostics−path−format=**, such as the identifiers of individual events and lines indicating interprocedural calls and returns.

range1=
    SGR substring for first additional range.

range2=
    SGR substring for second additional range.

locus=
    SGR substring for location information, **file:line** or **file:line:column** etc.

quote=
    SGR substring for information printed within quotes.

fnname=
    SGR substring for names of C++ functions.

targs=
    SGR substring for C++ function template parameter bindings.

fixit-insert=
    SGR substring for fix−it hints suggesting text to be inserted or replaced.

```
fixit-delete=
```
SGR substring for fix−it hints suggesting text to be deleted.

```
diff-filename=
```
SGR substring for filename headers within generated patches.

```
diff-hunk=
```
SGR substring for the starts of hunks within generated patches.

```
diff-delete=
```
SGR substring for deleted lines within generated patches.

```
diff-insert=
```
SGR substring for inserted lines within generated patches.

```
type-diff=
```
SGR substring for highlighting mismatching types within template arguments in the C++ frontend.

```
valid=
```
SGR substring for highlighting valid elements within text art diagrams.

```
invalid=
```
SGR substring for highlighting invalid elements within text art diagrams.

```
highlight-a=
highlight-b=
```
SGR substrings for contrasting two different things within diagnostics, such as a pair of mismatching types. See **−fdiagnostics−show−highlight−colors**.

−fdiagnostics−urls[=*WHEN***]**
> Use escape sequences to embed URLs in diagnostics. For example, when **−fdiagnostics−show−option** emits text showing the command−line option controlling a diagnostic, embed a URL for documentation of that option.
>
> *WHEN* is **never**, **always**, or **auto**. **auto** makes GCC use URL escape sequences only when the standard error is a terminal, and when not executing in an emacs shell or any graphical terminal which is known to be incompatible with this feature, see below.
>
> The default depends on how the compiler has been configured. It can be any of the above *WHEN* options.
>
> GCC can also be configured (via the **−−with−diagnostics−urls=auto−if−env** configure−time option) so that the default is affected by environment variables. Under such a configuration, GCC defaults to using **auto** if either **GCC_URLS** or **TERM_URLS** environment variables are present and non−empty in the environment of the compiler, or **never** if neither are.
>
> However, even with **−fdiagnostics−urls=always** the behavior is dependent on those environment variables: If **GCC_URLS** is set to empty or **no**, do not embed URLs in diagnostics. If set to **st**, URLs use ST escape sequences. If set to **bel**, the default, URLs use BEL escape sequences. Any other non−empty value enables the feature. If **GCC_URLS** is not set, use **TERM_URLS** as a fallback. Note: ST is an ANSI escape sequence, string terminator **ESC **, BEL is an ASCII character, CTRL−G that usually sounds like a beep.
>
> At this time GCC tries to detect also a few terminals that are known to not implement the URL feature, and have bugs or at least had bugs in some versions that are still in use, where the URL escapes are likely to misbehave, i.e. print garbage on the screen. That list is currently xfce4−terminal, certain known to be buggy gnome−terminal versions, the linux console, and mingw. This check can be skipped with the **−fdiagnostics−urls=always**.

−fno−diagnostics−show−option
> By default, each diagnostic emitted includes text indicating the command−line option that directly controls the diagnostic (if such an option is known to the diagnostic machinery). Specifying the

−fno−diagnostics−show−option flag suppresses that behavior.

−fno−diagnostics−show−caret
By default, each diagnostic emitted includes the original source line and a caret ˆ indicating the column. This option suppresses this information. The source line is truncated to *n* characters, if the **−fmessage−length=n** option is given. When the output is done to the terminal, the width is limited to the width given by the **COLUMNS** environment variable or, if not set, to the terminal width.

−fno−diagnostics−show−labels
By default, when printing source code (via **−fdiagnostics−show−caret**), diagnostics can label ranges of source code with pertinent information, such as the types of expressions:

```
printf ("foo %s bar", long_i + long_j);
             ~^         ~~~~~~~~~~~~~~~~
              |                |
            char *          long int
```

This option suppresses the printing of these labels (in the example above, the vertical bars and the "char *" and "long int" text).

−fno−diagnostics−show−event−links
By default, when printing execution paths (via **−fdiagnostics−path−format=inline−events**), GCC will print lines connecting related events, such as the line connecting events 1 and 2 in:

```
3 |    if (p)
  |       ^
  |       |
  |       (1) following `false' branch (when `p' is NULL)... ->-+
  |                                                             |
  |                                                             |
  |+-----------------------------------------------------------+
4 ||    return 0;
5 ||  return *p;
  ||          ~
  ||          |
  |+-------->(2) ...to here
  |            (3) dereference of NULL `p'
```

This option suppresses the printing of such connector lines.

−fno−diagnostics−show−cwe
Diagnostic messages can optionally have an associated
CWE (https://cwe.mitre.org/index.html) identifier. GCC itself only provides such metadata for some of the **−fanalyzer** diagnostics. GCC plugins may also provide diagnostics with such metadata. By default, if this information is present, it will be printed with the diagnostic. This option suppresses the printing of this metadata.

−fno−diagnostics−show−rules
Diagnostic messages can optionally have rules associated with them, such as from a coding standard, or a specification. GCC itself does not do this for any of its diagnostics, but plugins may do so. By default, if this information is present, it will be printed with the diagnostic. This option suppresses the printing of this metadata.

−fno−diagnostics−show−highlight−colors
GCC can use color for emphasis and contrast when printing diagnostic messages and quoting the user's source.

For example, in

```
                    demo.c: In function `test_bad_format_string_args':
                    ../../src/demo.c:25:18: warning: format `%i' expects argument of type `in
                       25 |    printf("hello %i", msg);
                          |                  ~^   ~~~
                          |                   |    |
                          |                   int const char *
                          |                  %s
```

*   the `%i` and `int` in the message and the `int` in the quoted source are colored using `highlight-a` (bold green by default), and

*   the `const char *` in the message and in the quoted source are both colored using `highlight-b` (bold blue by default).

The intent is to draw the reader's eyes to the relationships between the various aspects of the diagnostic message and the source, using color to group related elements and distinguish between mismatching ones.

This additional colorization is enabled by default if color printing is enabled (as per **−fdiagnostics−color=**), but it can be separately disabled via **−fno−diagnostics−show−highlight−colors**.

−fno−diagnostics−show−line−numbers
> By default, when printing source code (via **−fdiagnostics−show−caret**), a left margin is printed, showing line numbers. This option suppresses this left margin.

−fdiagnostics−minimum−margin−width=*width*
> This option controls the minimum width of the left margin printed by **−fdiagnostics−show−line−numbers**. It defaults to 6.

−fdiagnostics−parseable−fixits
> Emit fix−it hints in a machine−parseable format, suitable for consumption by IDEs. For each fix−it, a line will be printed after the relevant diagnostic, starting with the string "fix−it:". For example:

```
        fix-it:"test.c":{45:3-45:21}:"gtk_widget_show_all"
```

The location is expressed as a half−open range, expressed as a count of bytes, starting at byte 1 for the initial column. In the above example, bytes 3 through 20 of line 45 of "test.c" are to be replaced with the given string:

```
            00000000011111111112222222222
            12345678901234567890123456789
              gtk_widget_showall (dlg);
              ^^^^^^^^^^^^^^^^^^
              gtk_widget_show_all
```

The filename and replacement string escape backslash as "\\", tab as "\t", newline as "\n", double quotes as "\"", non−printable characters as octal (e.g. vertical tab as "\013").

An empty replacement string indicates that the given range is to be removed. An empty range (e.g. "45:3−45:3") indicates that the string is to be inserted at the given position.

−fdiagnostics−generate−patch
> Print fix−it hints to stderr in unified diff format, after any diagnostics are printed. For example:

```
            --- test.c
            +++ test.c
            @ -42,5 +42,5 @

             void show_cb(GtkDialog *dlg)
             {
            -  gtk_widget_showall(dlg);
```

```
                     +  gtk_widget_show_all(dlg);
                     }
```

The diff may or may not be colorized, following the same rules as for diagnostics (see
−fdiagnostics−color).

−fdiagnostics−show−template−tree

In the C++ frontend, when printing diagnostics showing mismatching template types, such as:

```
              could not convert 'std::map<int, std::vector<double> >()'
                from 'map<[...],vector<double>>' to 'map<[...],vector<float>>
```

the **−fdiagnostics−show−template−tree** flag enables printing a tree−like structure showing the
common and differing parts of the types, such as:

```
              map<
                [...],
                vector<
                  [double != float]>>
```

The parts that differ are highlighted with color ("double" and "float" in this case).

−fno−elide−type

By default when the C++ frontend prints diagnostics showing mismatching template types, common
parts of the types are printed as "[...]" to simplify the error message. For example:

```
              could not convert 'std::map<int, std::vector<double> >()'
                from 'map<[...],vector<double>>' to 'map<[...],vector<float>>
```

Specifying the **−fno−elide−type** flag suppresses that behavior. This flag also affects the output of the
−fdiagnostics−show−template−tree flag.

−fdiagnostics−path−format=_KIND_

Specify how to print paths of control−flow events for diagnostics that have such a path associated with
them.

KIND is **none**, **separate−events**, or **inline−events**, the default.

none means to not print diagnostic paths.

separate−events means to print a separate "note" diagnostic for each event within the diagnostic. For
example:

```
          test.c:29:5: error: passing NULL as argument 1 to 'PyList_Append' which r
          test.c:25:10: note: (1) when 'PyList_New' fails, returning NULL
          test.c:27:3: note: (2) when 'i < count'
          test.c:29:5: note: (3) when calling 'PyList_Append', passing NULL from (1
```
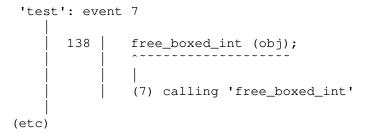
inline−events means to print the events "inline" within the source code. This view attempts to
consolidate the events into runs of sufficiently−close events, printing them as labelled ranges within
the source.

For example, the same events as above might be printed as:

```
                    'test': events 1-3
                     25 |    list = PyList_New(0);
                        |           ^~~~~~~~~~~~~
                        |                |
                        |                (1) when 'PyList_New' fails, returning NULL
                     26 |
                     27 |    for (i = 0; i < count; i++) {
                        |    ~~~
                        |    |
                        |    (2) when 'i < count'
                     28 |      item = PyLong_FromLong(random());
                     29 |      PyList_Append(list, item);
                        |      ~~~~~~~~~~~~~~~~~~~~~~~~~~
                        |      |
                        |      (3) when calling 'PyList_Append', passing NULL from (1) as ar
```

Interprocedural control flow is shown by grouping the events by stack frame, and using indentation to
show how stack frames are nested, pushed, and popped.

For example:

```
                    'test': events 1-2
                      |
                      |   133 |  {
                      |       |  ^
                      |       |  |
                      |       |  (1) entering 'test'
                      |   134 |    boxed_int *obj = make_boxed_int (i);
                      |       |                     ~~~~~~~~~~~~~~~~~~~
                      |       |                         |
                      |       |                         (2) calling 'make_boxed_int'
                      |
                    +--> 'make_boxed_int': events 3-4
                           |
                           |   120 |  {
                           |       |  ^
                           |       |  |
                           |       |  (3) entering 'make_boxed_int'
                           |   121 |    boxed_int *result = (boxed_int *)wrapped_malloc (si
                           |       |                                     ~~~~~~~~~~~~~~~~~~~
                           |       |                                         |
                           |       |                                         (4) calling 'wrapp
                           |
                         +--> 'wrapped_malloc': events 5-6
                                |
                                |     7 |  {
                                |       |  ^
                                |       |  |
                                |       |  (5) entering 'wrapped_malloc'
                                |     8 |    return malloc (size);
                                |       |           ~~~~~~~~~~~~~~
                                |       |               |
                                |       |               (6) calling 'malloc'
                                |
                    <------------+
                     |
```

```
                    'test': event 7
                       |
                       |  138 |    free_boxed_int (obj);
                       |      |    ^~~~~~~~~~~~~~~~~~~~
                       |      |    |
                       |      |    (7) calling 'free_boxed_int'
                       |
                  (etc)
```

−fdiagnostics−show−path−depths

This option provides additional information when printing control−flow paths associated with a diagnostic.

If this is option is provided then the stack depth will be printed for each run of events within **−fdiagnostics−path−format=inline−events**.   If   provided   with **−fdiagnostics−path−format=separate−events**, then the stack depth and function declaration will be appended when printing each event.

This is intended for use by GCC developers and plugin developers when debugging diagnostics that report interprocedural control flow.

−fno−show−column

Do not print column numbers in diagnostics. This may be necessary if diagnostics are being scanned by a program that does not understand the column numbers, such as **dejagnu**.

−fdiagnostics−column−unit=*UNIT*

Select the units for the column number. This affects traditional diagnostics (in the absence of **−fno−show−column**), as well as JSON format diagnostics if requested.

The default *UNIT*, **display**, considers the number of display columns occupied by each character. This may be larger than the number of bytes required to encode the character, in the case of tab characters, or it may be smaller, in the case of multibyte characters. For example, the character "GREEK SMALL LETTER PI (U+03C0)" occupies one display column, and its UTF−8 encoding requires two bytes; the character "SLIGHTLY SMILING FACE (U+1F642)" occupies two display columns, and its UTF−8 encoding requires four bytes.

Setting *UNIT* to **byte** changes the column number to the raw byte count in all cases, as was traditionally output by GCC prior to version 11.1.0.

−fdiagnostics−column−origin=*ORIGIN*

Select the origin for column numbers, i.e. the column number assigned to the first column. The default value of 1 corresponds to traditional GCC behavior and to the GNU style guide. Some utilities may perform better with an origin of 0; any non−negative value may be specified.

−fdiagnostics−escape−format=*FORMAT*

When GCC prints pertinent source lines for a diagnostic it normally attempts to print the source bytes directly. However, some diagnostics relate to encoding issues in the source file, such as malformed UTF−8, or issues with Unicode normalization. These diagnostics are flagged so that GCC will escape bytes that are not printable ASCII when printing their pertinent source lines.

This option controls how such bytes should be escaped.

The default *FORMAT*, **unicode** displays Unicode characters that are not printable ASCII in the form **<U+XXXX>**, and bytes that do not correspond to a Unicode character validly−encoded in UTF−8−encoded will be displayed as hexadecimal in the form **<XX>**.

For example, a source line containing the string **before** followed by the Unicode character U+03C0 ("GREEK SMALL LETTER PI", with UTF−8 encoding 0xCF 0x80) followed by the byte 0xBF (a stray UTF−8 trailing byte), followed by the string **after** will be printed for such a diagnostic as:

```
before<U+03C0><BF>after
```

Setting *FORMAT* to **bytes** will display all non−printable−ASCII bytes in the form **<XX>**, thus showing the underlying encoding of non−ASCII Unicode characters. For the example above, the following will be printed:

```
before<CF><80><BF>after
```

−fdiagnostics−text−art−charset=_CHARSET_

Some diagnostics can contain "text art" diagrams: visualizations created from text, intended to be viewed in a monospaced font.

This option selects which characters should be used for printing such diagrams, if any. *CHARSET* is **none**, **ascii**, **unicode**, or **emoji**.

The **none** value suppresses the printing of such diagrams. The **ascii** value will ensure that such diagrams are pure ASCII ("ASCII art"). The **unicode** value will allow for conservative use of unicode drawing characters (such as box−drawing characters). The **emoji** value further adds the possibility of emoji in the output (such as emitting U+26A0 WARNING SIGN followed by U+FE0F VARIATION SELECTOR−16 to select the emoji variant of the character).

The default is **emoji**, except when the environment variable **LANG** is set to **C**, in which case the default is **ascii**.

−fdiagnostics−format=_FORMAT_

Select a different format for printing diagnostics. *FORMAT* is **text**, **sarif−stderr**, **sarif−file**, **json**, **json−stderr**, or **json−file**.

Using this option replaces any additional "output sinks" added by **−fdiagnostics−add−output=**, or that set by **−fdiagnostics−set−output=**.

The default is **text**.

The **sarif−stderr** and **sarif−file** formats both emit diagnostics in SARIF Version 2.1.0 format, either to stderr, or to a file named *source.sarif*, respectively.

The various **json**, **json−stderr**, and **json−file** values are deprecated and refer to a legacy JSON−based output format. The **json** format is a synonym for **json−stderr**. The **json−stderr** and **json−file** formats are identical, apart from where the JSON is emitted to. With **json−stderr**, the JSON is emitted to stderr, whereas with **json−file** it is written to *source.gcc.json*.

−fdiagnostics−add−output=_DIAGNOSTICS−OUTPUT−SPEC_

Add an additional "output sink" for emitting diagnostics.

DIAGNOSTICS−OUTPUT−SPEC should specify a scheme, optionally followed by **:** and one or more *KEY=VALUE* pairs, in this form:

```
<SCHEME>
<SCHEME>:<KEY>=<VALUE>
<SCHEME>:<KEY>=<VALUE>,<KEY2>=<VALUE2>
```

etc.

Schemes, keys, or values with a name prefixed "experimental" may change or be removed without notice.

SCHEME can be

text

Emit diagnostics to stderr using GCC's classic text output format.

Supported keys are:

color=[yes|no]

Override colorization settings from **−fdiagnostics−color** for this text output.

experimental−nesting=[yes|no]
> Enable an experimental mode that emphasizes hierarchical relationships within diagnostics messages, displaying location information on separate lines.

experimental−nesting−show−locations=[yes|no]
> If `experimental-nesting=yes`, then by default locations are shown; set this key to `no` to disable printing such locations. This exists for use by GCC developers, for writing DejaGnu test cases.

experimental−nesting−show−levels=[yes|no]
> This is a debugging option for use with `experimental-nesting=yes`. Set this key to `yes` to print explicit nesting levels in the output. This exists for use by GCC developers.

sarif
> Emit diagnostics to a file in SARIF format.
>
> Supported keys are:
>
> **file=**_FILENAME_
>> Specify the filename to write the SARIF output to, potentially with a leading absolute or relative path. If not specified, it defaults to *source.sarif*.
>
> **version=[2.1|2.2−prerelease]**
>> Specify the version of SARIF to use for the output. If not specified, defaults to 2.1. `2.2-prerelease` uses an unofficial draft of the future SARIF 2.2 specification and should only be used for experimentation in this release.
>
> For example,
>
> ```
>  -fdiagnostics-add-output=sarif:version=2.1,file=foo.2.1.sarif
>  -fdiagnostics-add-output=sarif:version=2.2-prerelease,file=foo.2.2.sarif
> ```
>
> would add a pair of outputs, each writing to a different file, using versions 2.1 and 2.2 of the SARIF standard respectively.
>
> In EBNF:
>
> ```
>     <diagnostics-output-specifier> = <diagnostics-output-name>
>                                    │ <diagnostics-output-name>, ":", <key
>
>     <diagnostics-output-name> = "text" │ "sarif";
>
>     <key-value-pairs> = <key-value-pair>
>                       │ <key-value-pair> "," <key-value-pairs>;
>
>     <key-value-pair> = <key> "=" <value>;
>
>     <key> = ? string without a '=' ? ;
>     <value> = ? string without a ',' ? ;
> ```

−fdiagnostics−set−output=_DIAGNOSTICS−OUTPUT−SPEC_
> This works in a similar way to **−fdiagnostics−add−output=** except that instead of adding an additional "output sink" for diagnostics, it replaces all existing output sinks, such as from **−fdiagnostics−format=**, **−fdiagnostics−add−output=**, or a prior call to **−fdiagnostics−set−output=**.

−fno−diagnostics−json−formatting
> By default, when JSON is emitted for diagnostics (via **−fdiagnostics−format=sarif−stderr**, **−fdiagnostics−format=sarif−file**, **−fdiagnostics−format=json**, **−fdiagnostics−format=json−stderr**, **−fdiagnostics−format=json−file**), GCC will add newlines and indentation to visually emphasize the hierarchical structure of the JSON.
>
> Use **−fno−diagnostics−json−formatting** to suppress this whitespace. It must be passed before the

option it is to affect.

This is intended for compatibility with tools that do not expect the output to contain newlines, such as that emitted by older GCC releases.

Options to Request or Suppress Warnings

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

The following language−independent options do not enable specific warnings but control the kinds of diagnostics produced by GCC.

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

−fmax−errors=*n*
Limits the maximum number of error messages to *n*, at which point GCC bails out rather than attempting to continue processing the source code. If *n* is 0 (the default), there is no limit on the number of error messages produced. If **−Wfatal−errors** is also specified, then **−Wfatal−errors** takes precedence over this option.

−w Inhibit all warning messages.

−Werror
Turn all warnings into errors.

−Werror=
Turn the specified warning into an error. The specifier for a warning is appended; for example **−Werror=switch** turns the warnings controlled by **−Wswitch** into errors. This switch takes a negative form, to be used to negate **−Werror** for specific warnings; for example **−Wno−error=switch** makes **−Wswitch** warnings not be errors, even when **−Werror** is in effect.

The warning message for each controllable warning includes the option that controls the warning. That option can then be used with **−Werror=** and **−Wno−error=** as described above. (Printing of the option in the warning message can be disabled using the **−fno−diagnostics−show−option** flag.)

Note that specifying **−Werror=***foo* automatically implies **−W***foo*. However, **−Wno−error=***foo* does not imply anything.

−Wfatal−errors
This option causes the compiler to abort compilation on the first error occurred rather than trying to keep going and printing further error messages.

You can request many specific warnings with options beginning with **−W**, for example **−Wunused−variable** to request warnings on declarations of variables that are never used. Each of these specific warning options also has a negative form beginning with **−Wno−** to turn off warnings; for example, **−Wno−unused−variable**. This manual lists only one of the two forms, whichever is not the default. For further language−specific options also refer to **C++ Dialect Options** and **Objective−C and Objective−C++ Dialect Options**. Additional warnings can be produced by enabling the static analyzer;

Some options, such as **−Wall** and **−Wextra**, turn on other options, such as **−Wunused**, which may turn on further options, such as **−Wunused−variable**. The combined effect of positive and negative forms is that more specific options have priority over less specific ones, independently of their position in the command line. For options of the same specificity, the last one takes effect. Options enabled or disabled via pragmas take effect as if they appeared at the end of the command line.

When an unrecognized warning option is requested (e.g., **−Wunknown−warning**), GCC gives an error stating that the option is not recognized. However, if the **−Wno−** form is used, the behavior is slightly different: no diagnostic is produced for **−Wno−unknown−warning** unless other diagnostics are being produced. This allows the use of new **−Wno−** options with old compilers, but if something goes wrong, the compiler warns that an unrecognized option is present.

The effectiveness of some warnings depends on optimizations also being enabled. For example,

−Wsuggest−final−types is more effective with link−time optimization. Some other warnings may not be issued at all unless optimization is enabled. While optimization in general improves the efficacy of warnings about control and data−flow problems, in some cases it may also cause false positives.

−Wpedantic
−pedantic

> Issue all the warnings demanded by strict ISO C and ISO C++; diagnose all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. This follows the version of the ISO C or C++ standard specified by any **−std** option used.
>
> Valid ISO C and ISO C++ programs should compile properly with or without this option (though a rare few require **−ansi** or a **−std** option specifying the version of the standard). However, without this option, certain GNU extensions and traditional C and C++ features are supported as well. With this option, they are diagnosed (or rejected with **−pedantic−errors**).
>
> **−Wpedantic** does not cause warning messages for use of the alternate keywords whose names begin and end with `__`. This alternate format can also be used to disable warnings for non−ISO **__intN** types, i.e. `__intN__`. Pedantic warnings are also disabled in the expression that follows `__extension__`. However, only system header files should use these escape routes; application programs should avoid them.
>
> Some warnings about non−conforming programs are controlled by options other than **−Wpedantic**; in many cases they are implied by **−Wpedantic** but can be disabled separately by their specific option, e.g. **−Wpedantic −Wno−pointer−sign**.
>
> Where the standard specified with **−std** represents a GNU extended dialect of C, such as **gnu90** or **gnu99**, there is a corresponding *base standard*, the version of ISO C on which the GNU extended dialect is based. Warnings from **−Wpedantic** are given where they are required by the base standard. (It does not make sense for such warnings to be given only for features not in the specified GNU C dialect, since by definition the GNU dialects of C include all features the compiler supports with the given option, and there would be nothing to warn about.)

−pedantic−errors

> Give an error whenever the *base standard* (see **−Wpedantic**) requires a diagnostic, in some cases where there is undefined behavior at compile−time and in some other cases that do not prevent compilation of programs that are valid according to the standard. This is not equivalent to **−Werror=pedantic**: the latter option is unlikely to be useful, as it only makes errors of the diagnostics that are controlled by **−Wpedantic**, whereas this option also affects required diagnostics that are always enabled or controlled by options other than **−Wpedantic**.
>
> If you want the required diagnostics that are warnings by default to be errors instead, but don't also want to enable the **−Wpedantic** diagnostics, you can specify **−pedantic−errors −Wno−pedantic** (or **−pedantic−errors −Wno−error=pedantic** to enable them but only as warnings).
>
> Some required diagnostics are errors by default, but can be reduced to warnings using **−fpermissive** or their specific warning option, e.g. **−Wno−error=narrowing**.
>
> Some diagnostics for non−ISO practices are controlled by specific warning options other than **−Wpedantic**, but are also made errors by **−pedantic−errors**. For instance:
>
> **−Wattributes** (for standard attributes) **−Wchanges−meaning** (C++) **−Wcomma−subscript** (C++23 or later) **−Wdeclaration−after−statement** (C90 or earlier) **−Welaborated−enum−base** (C++11 or later) **−Wimplicit−int** (C99 or later) **−Wimplicit−function−declaration** (C99 or later) **−Wincompatible−pointer−types −Wint−conversion −Wlong−long** (C90 or earlier) **−Wmain −Wnarrowing** (C++11 or later) **−Wpointer−arith −Wpointer−sign −Wincompatible−pointer−types −Wregister** (C++17 or later) **−Wvla** (C90 or earlier) **−Wwrite−strings** (C++11 or later)

−fpermissive

    Downgrade some required diagnostics about nonconformant code from errors to warnings. Thus, using **−fpermissive** allows some nonconforming code to compile. Some C++ diagnostics are controlled only by this flag, but it also downgrades some C and C++ diagnostics that have their own flag:

    −Wdeclaration−missing−parameter−type (C and Objective−C only) **−Wimplicit−function−declaration** (C and Objective−C only) **−Wimplicit−int** (C and Objective−C only) **−Wincompatible−pointer−types** (C and Objective−C only) **−Wint−conversion** (C and Objective−C only) **−Wnarrowing** (C++ and Objective−C++ only) **−Wreturn−mismatch** (C and Objective−C only) **−Wtemplate−body** (C++ and Objective−C++ only)

    The **−fpermissive** option is the default for historic C language modes (**−std=c89**, **−std=gnu89**, **−std=c90**, **−std=gnu90**).

−Wall

    This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. This also enables some language−specific warnings described in **C++ Dialect Options** and **Objective−C and Objective−C++ Dialect Options**.

    −Wall turns on the following warning flags:

    −Waddress −Waligned−new (C++ and Objective−C++ only) **−Warray−bounds=1** (only with **−O2**) **−Warray−compare −Warray−parameter=2 −Wbool−compare −Wbool−operation −Wc++11−compat −Wc++14−compat −Wc++17compat −Wc++20compat −Wcatch−value** (C++ and Objective−C++ only) **−Wchar−subscripts −Wclass−memaccess** (C++ and Objective−C++ only) **−Wcomment −Wdangling−else −Wdangling−pointer=2 −Wdelete−non−virtual−dtor** (C++ and Objective−C++ only) **−Wduplicate−decl−specifier** (C and Objective−C only) **−Wenum−compare** (in C/ObjC; this is on by default in C++) **−Wenum−int−mismatch** (C and Objective−C only) **−Wformat=1 −Wformat−contains−nul −Wformat−diag −Wformat−extra−args −Wformat−overflow=1 −Wformat−truncation=1 −Wformat−zero−length −Wframe−address −Wimplicit** (C and Objective−C only) **−Wimplicit−function−declaration** (C and Objective−C only) **−Wimplicit−int** (C and Objective−C only) **−Winfinite−recursion −Winit−self** (C++ and Objective−C++ only) **−Wint−in−bool−context −Wlogical−not−parentheses −Wmain** (only for C/ObjC and unless **−ffreestanding**) **−Wmaybe−uninitialized −Wmemset−elt−size −Wmemset−transposed−args −Wmisleading−indentation** (only for C/C++) **−Wmismatched−dealloc −Wmismatched−new−delete** (C++ and Objective−C++ only) **−Wmissing−attributes −Wmissing−braces** (only for C/ObjC) **−Wmultistatement−macros −Wnarrowing** (C++ and Objective−C++ only) **−Wnonnull −Wnonnull−compare −Wopenmp−simd** (C and C++ only) **−Woverloaded−virtual=1** (C++ and Objective−C++ only) **−Wpacked−not−aligned −Wparentheses −Wpessimizing−move** (C++ and Objective−C++ only) **−Wpointer−sign** (only for C/ObjC) **−Wrange−loop−construct** (C++ and Objective−C++ only) **−Wreorder** (C++ and Objective−C++ only) **−Wrestrict −Wreturn−type −Wself−move** (C++ and Objective−C++ only) **−Wsequence−point −Wsign−compare** (C++ and Objective−C++ only) **−Wsizeof−array−div −Wsizeof−pointer−div −Wsizeof−pointer−memaccess −Wstrict−aliasing −Wstrict−overflow=1 −Wswitch −Wtautological−compare −Wtrigraphs −Wuninitialized −Wunknown−pragmas −Wunused −Wunused−but−set−variable −Wunused−const−variable=1** (only for C/ObjC) **−Wunused−function −Wunused−label −Wunused−local−typedefs −Wunused−value −Wunused−variable −Wuse−after−free=2 −Wvla−parameter −Wvolatile−register−var −Wzero−length−bounds**

    Note that some warning flags are not implied by **−Wall**. Some of them warn about constructions that users generally do not consider questionable, but which occasionally you might wish to check for; others warn about constructions that are necessary or hard to avoid in some cases, and there is no simple way to modify the code to suppress the warning. Some of them are enabled by **−Wextra** but many of them must be enabled individually.

−Wextra

This enables some extra warning flags that are not enabled by **−Wall**. (This option used to be called **−W**. The older name is still supported, but the newer name is more descriptive.)

−Wabsolute−value (only for C/ObjC) **−Walloc−size** **−Wcalloc−transposed−args** **−Wcast−function−type** **−Wclobbered** **−Wdangling−reference** (C++ only) **−Wdeprecated−copy** (C++ and Objective−C++ only) **−Wempty−body** **−Wenum−conversion** (only for C/ObjC) **−Wexpansion−to−defined** **−Wignored−qualifiers** (only for C/C++) **−Wimplicit−fallthrough=3** **−Wmaybe−uninitialized** **−Wmissing−field−initializers** **−Wmissing−parameter−name** (C/ObjC only) **−Wmissing−parameter−type** (C/ObjC only) **−Wold−style−declaration** (C/ObjC only) **−Woverride−init** (C/ObjC only) **−Wredundant−move** (C++ and Objective−C++ only) **−Wshift−negative−value** (in C++11 to C++17 and in C99 and newer) **−Wsign−compare** (C++ and Objective−C++ only) **−Wsized−deallocation** (C++ and Objective−C++ only) **−Wstring−compare** **−Wtype−limits** **−Wuninitialized** **−Wunterminated−string−initialization** **−Wunused−parameter** (only with **−Wunused** or **−Wall**) **−Wunused−but−set−parameter** (only with **−Wunused** or **−Wall**)

The option **−Wextra** also prints warning messages for the following cases:

*   A pointer is compared against integer zero with <, <=, >, or >=.

*   (C++ only) An enumerator and a non−enumerator both appear in a conditional expression.

*   (C++ only) Ambiguous virtual bases.

*   (C++ only) Subscripting an array that has been declared `register`.

*   (C++ only) Taking the address of a variable that has been declared `register`.

*   (C++ only) A base class is not initialized in the copy constructor of a derived class.

−Wabi (C, Objective−C, C++ and Objective−C++ only)

Warn about code affected by ABI changes. This includes code that may not be compatible with the vendor−neutral C++ ABI as well as the psABI for the particular target. The latter warnings are also controlled separately by **−Wpsabi**, which is implied by **−Wabi**.

Since G++ now defaults to updating the ABI with each major release, normally **−Wabi** warns only about C++ ABI compatibility problems if there is a check added later in a release series for an ABI issue discovered since the initial release. **−Wabi** warns about more things if an older ABI version is selected (with **−fabi−version=**n).

−Wabi can also be used with an explicit version number to warn about C++ ABI compatibility with a particular **−fabi−version** level, e.g. **−Wabi=2** to warn about changes relative to **−fabi−version=2**.

If an explicit version number is provided and **−fabi−compat−version** is not specified, the version number from this option is used for compatibility aliases. If no explicit version number is provided with this option, but **−fabi−compat−version** is specified, that version number is used for C++ ABI warnings.

Although an effort has been made to warn about all such cases, there are probably some cases that are not warned about, even though G++ is generating incompatible code. There may also be cases where warnings are emitted even though the code that is generated is compatible.

You should rewrite your code to avoid these warnings if you are concerned about the fact that code generated by G++ may not be binary compatible with code generated by other compilers.

Known incompatibilities in **−fabi−version=2** (which was the default from GCC 3.4 to 4.9) include:

*   A template with a non−type template parameter of reference type was mangled incorrectly:

        ```
        extern int N;
        template <int &> struct S {};
        void n (S<N>) {2}
        ```

This was fixed in **−fabi−version=3**.

* SIMD vector types declared using `__attribute ((vector_size))` were mangled in a non−standard way that does not allow for overloading of functions taking vectors of different sizes.

  The mangling was changed in **−fabi−version=4**.

* `__attribute ((const))` and `noreturn` were mangled as type qualifiers, and `decltype` of a plain declaration was folded away.

  These mangling issues were fixed in **−fabi−version=5**.

* Scoped enumerators passed as arguments to a variadic function are promoted like unscoped enumerators, causing `va_arg` to complain. On most targets this does not actually affect the parameter passing ABI, as there is no way to pass an argument smaller than `int`.

  Also, the ABI changed the mangling of template argument packs, `const_cast`, `static_cast`, prefix increment/decrement, and a class scope function used as a template argument.

  These issues were corrected in **−fabi−version=6**.

* Lambdas in default argument scope were mangled incorrectly, and the ABI changed the mangling of `nullptr_t`.

  These issues were corrected in **−fabi−version=7**.

* When mangling a function type with function−cv−qualifiers, the un−qualified function type was incorrectly treated as a substitution candidate.

  This was fixed in **−fabi−version=8**, the default for GCC 5.1.

* `decltype(nullptr)` incorrectly had an alignment of 1, leading to unaligned accesses. Note that this did not affect the ABI of a function with a `nullptr_t` parameter, as parameters have a minimum alignment.

  This was fixed in **−fabi−version=9**, the default for GCC 5.2.

* Target−specific attributes that affect the identity of a type, such as ia32 calling conventions on a function type (stdcall, regparm, etc.), did not affect the mangled name, leading to name collisions when function pointers were used as template arguments.

  This was fixed in **−fabi−version=10**, the default for GCC 6.1.

−Wpsabi (C, Objective−C, C++ and Objective−C++ only)
> **−Wpsabi** enables warnings about processor−specific ABI changes, such as changes in alignment requirements or how function arguments are passed. On several targets, including AArch64, ARM, x86, MIPS, RS6000/PowerPC, and S/390, these details have changed between different versions of GCC and/or different versions of the C or C++ language standards in ways that affect binary compatibility of compiled code. With **−Wpsabi**, GCC can detect potentially incompatible usages and warn you about them.
>
> **−Wpsabi** is enabled by default, and is also implied by **−Wabi**.

−Wno−changes−meaning (C++ and Objective−C++ only)
> C++ requires that unqualified uses of a name within a class have the same meaning in the complete scope of the class, so declaring the name after using it is ill−formed:

```
struct A;
struct B1 { A a; typedef A A; }; // warning, 'A' changes meaning
struct B2 { A a; struct A { }; }; // error, 'A' changes meaning
```

By default, the B1 case is only a warning because the two declarations have the same type, while the B2 case is an error. Both diagnostics can be disabled with **−Wno−changes−meaning**. Alternately, the error case can be reduced to a warning with **−Wno−error=changes−meaning** or **−fpermissive**.

Both diagnostics are also suppressed by **−fms−extensions**.

−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. This warning is enabled by **−Wall**.

−Wno−coverage−mismatch

Warn if feedback profiles do not match when using the **−fprofile−use** option. If a source file is changed between compiling with **−fprofile−generate** and with **−fprofile−use**, the files with the profile feedback can fail to match the source file and GCC cannot use the profile feedback information. By default, this warning is enabled and is treated as an error. **−Wno−coverage−mismatch** can be used to disable the warning or **−Wno−error=coverage−mismatch** can be used to disable the error. Disabling the error for this warning can result in poorly optimized code and is useful only in the case of very minor changes such as bug fixes to an existing code−base. Completely disabling the warning is not recommended.

−Wno−coverage−too−many−conditions

Warn if **−fcondition−coverage** is used and an expression have too many terms and GCC gives up coverage. Coverage is given up when there are more terms in the conditional than there are bits in a `gcov_type_unsigned`. This warning is enabled by default.

−Wno−coverage−too−many−paths

Warn if **−fpath−coverage** is used and a function has too many paths and GCC gives up coverage. Giving up is controlled by **−fpath−coverage−limit**. This warning is enabled by default.

−Wno−coverage−invalid−line−number

Warn in case a function ends earlier than it begins due to an invalid linenum macros. The warning is emitted only with **−−coverage** enabled.

By default, this warning is enabled and is treated as an error. **−Wno−coverage−invalid−line−number** can be used to disable the warning or **−Wno−error=coverage−invalid−line−number** can be used to disable the error.

−Wno−cpp (C, Objective−C, C++, Objective−C++ and Fortran only)

Suppress warning messages emitted by `#warning` directives.

−Wdouble−promotion (C, C++, Objective−C and Objective−C++ only)

Give a warning when a value of type `float` is implicitly promoted to `double`. CPUs with a 32−bit "single−precision" floating−point unit implement `float` in hardware, but emulate `double` in software. On such a machine, doing computations using `double` values is much more expensive because of the overhead required for software emulation.

It is easy to accidentally do computations with `double` because floating−point literals are implicitly of type `double`. For example, in:

```
float area(float radius)
{
    return 3.14159 * radius * radius;
}
```

the compiler performs the entire computation with `double` because the floating−point literal is a `double`.

−Wduplicate−decl−specifier (C and Objective−C only)

Warn if a declaration has duplicate `const`, `volatile`, `restrict` or `_Atomic` specifier. This warning is enabled by **−Wall**.

−Wformat
−Wformat=_n_

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 **−Wpedantic** is used with **−Wformat**, warnings are given about format features not in the selected standard version (but not for `strfmon` formats, since those are not in any version of the C standard).

−Wformat=1
−Wformat
> Option **−Wformat** is equivalent to **−Wformat=1**, and **−Wno−format** is equivalent to **−Wformat=0**. Since **−Wformat** also checks for null format arguments for several functions, **−Wformat** also implies **−Wnonnull**. Some aspects of this level of format checking can be disabled by the options: **−Wno−format−contains−nul**, **−Wno−format−extra−args**, and **−Wno−format−zero−length**. **−Wformat** is enabled by **−Wall**.

−Wformat=2
> Enable **−Wformat** plus additional format checks. Currently equivalent to **−Wformat −Wformat−nonliteral −Wformat−security −Wformat−y2k**.

−Wno−format−contains−nul
> If **−Wformat** is specified, do not warn about format strings that contain NUL bytes.

−Wno−format−extra−args
> If **−Wformat** is specified, do not warn about excess arguments to a `printf` or `scanf` format function. The C standard specifies that such arguments are ignored.
>
> Where the unused arguments lie between used arguments that are specified with **$** operand number specifications, normally warnings are still given, since the implementation could not know what type to pass to `va_arg` to skip the unused arguments. However, in the case of `scanf` formats, this option suppresses the warning if the unused arguments are all pointers, since the Single Unix Specification says that such unused arguments are allowed.

−Wformat−overflow
−Wformat−overflow=*level*
> Warn about calls to formatted input/output functions such as `sprintf` and `vsprintf` that might overflow the destination buffer. When the exact number of bytes written by a format directive cannot be determined at compile−time it is estimated based on heuristics that depend on the *level* argument and on optimization. While enabling optimization will in most cases improve the accuracy of the warning, it may also result in false positives.

−Wformat−overflow
−Wformat−overflow=1
> Level *1* of **−Wformat−overflow** enabled by **−Wformat** employs a conservative approach that warns only about calls that most likely overflow the buffer. At this level, numeric arguments to format directives with unknown values are assumed to have the value of one, and strings of unknown length to be empty. Numeric arguments that are known to be bounded to a subrange of their type, or string arguments whose output is bounded either by their directive's precision or by a finite set of string literals, are assumed to take on the value within the range that results in the most bytes on output. For example, the call to `sprintf` below is diagnosed because even with both *a* and *b* equal to zero, the terminating NUL character (`'\0'`) appended by the function to the destination buffer will be written past its end. Increasing the size of the buffer by a single byte is sufficient to avoid the warning, though it may not be sufficient to avoid the overflow.

```
void f (int a, int b)
{
  char buf [13];
  sprintf (buf, "a = %i, b = %i\n", a, b);
}
```

−Wformat−overflow=2

Level *2* warns also about calls that might overflow the destination buffer given an argument of sufficient length or magnitude. At level *2*, unknown numeric arguments are assumed to have the minimum representable value for signed types with a precision greater than 1, and the maximum representable value otherwise. Unknown string arguments whose length cannot be assumed to be bounded either by the directive's precision, or by a finite set of string literals they may evaluate to, or the character array they may point to, are assumed to be 1 character long.

At level *2*, the call in the example above is again diagnosed, but this time because with *a* equal to a 32−bit INT_MIN the first %i directive will write some of its digits beyond the end of the destination buffer. To make the call safe regardless of the values of the two variables, the size of the destination buffer must be increased to at least 34 bytes. GCC includes the minimum size of the buffer in an informational note following the warning.

An alternative to increasing the size of the destination buffer is to constrain the range of formatted values. The maximum length of string arguments can be bounded by specifying the precision in the format directive. When numeric arguments of format directives can be assumed to be bounded by less than the precision of their type, choosing an appropriate length modifier to the format specifier will reduce the required buffer size. For example, if *a* and *b* in the example above can be assumed to be within the precision of the short int type then using either the %hi format directive or casting the argument to short reduces the maximum required size of the buffer to 24 bytes.

```
void f (int a, int b)
{
  char buf [23];
  sprintf (buf, "a = %hi, b = %i\n", a, (short)b);
}
```

−Wno−format−zero−length

If **−Wformat** is specified, do not warn about zero−length formats. The C standard specifies that zero−length formats are allowed.

−Wformat−nonliteral

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

−Wformat−security

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

−Wformat−signedness

If **−Wformat** is specified, also warn if the format string requires an unsigned argument and the argument is signed and vice versa.

−Wformat−truncation
−Wformat−truncation=*level*

Warn about calls to formatted input/output functions such as snprintf and vsnprintf that might result in output truncation. When the exact number of bytes written by a format directive cannot be determined at compile−time it is estimated based on heuristics that depend on the *level* argument and

on optimization. While enabling optimization will in most cases improve the accuracy of the warning, it may also result in false positives. Except as noted otherwise, the option uses the same logic **−Wformat−overflow**.

−Wformat−truncation
−Wformat−truncation=1
>   Level *1* of **−Wformat−truncation** enabled by **−Wformat** employs a conservative approach that warns only about calls to bounded functions whose return value is unused and that will most likely result in output truncation.

−Wformat−truncation=2
>   Level *2* warns also about calls to bounded functions whose return value is used and that might result in truncation given an argument of sufficient length or magnitude.

−Wformat−y2k
>   If **−Wformat** is specified, also warn about `strftime` formats that may yield only a two−digit year.

−Wnonnull
>   Warn about passing a null pointer for arguments marked as requiring a non−null value by the `nonnull` function attribute.
>
>   **−Wnonnull** is included in **−Wall** and **−Wformat**. It can be disabled with the **−Wno−nonnull** option.

−Wnonnull−compare
>   Warn when comparing an argument marked with the `nonnull` function attribute against null inside the function.
>
>   **−Wnonnull−compare** is included in **−Wall**. It can be disabled with the **−Wno−nonnull−compare** option.

−Wnull−dereference
>   Warn if the compiler detects paths that trigger erroneous or undefined behavior due to dereferencing a null pointer. This option is only active when **−fdelete−null−pointer−checks** is active, which is enabled by optimizations in most targets. The precision of the warnings depends on the optimization options used.

−Wno−musttail−local−addr
>   Do not warn about passing a pointer (or in C++, a reference) to a local variable or label to argument of a `musttail` call. Those variables go out of scope before the tail call instruction.

−Wmaybe−musttail−local−addr
>   Warn when address of a local variable can escape to a `musttail` call, unless it goes out of scope already before the `musttail` call.

```
int foo (int *);

int
bar (int *x)
{
  if (x[0] == 1)
    {
      int a = 42;
      foo (&a);
      /* Without the C<musttail> attribute this call would not
         be tail called, because address of the C<a> variable escapes
         and the second foo call could dereference it.  With the attribut
         the local variables are assumed to go out of scope immediately
         before the tail call instruction and the compiler warns about
         this.  */
      [[gnu::musttail]] return foo (nullptr);
    }
```

```
                  else
                    {
                      {
                        int a = 42;
                        foo (&a);
                      }
                      /* The C<a> variable isn't already in scope, so even when it
                         escaped, even without C<musttail> attribute it would be
                         undefined behavior to dereference it and the compiler could
                         turn this into a tail call.  No warning is diagnosed here.  */
                      [[gnu::musttail]] return foo (nullptr);
                    }
                }
```

This warning is enabled by **−Wextra**.

−Wnrvo (C++ and Objective−C++ only)

Warn if the compiler does not elide the copy from a local variable to the return value of a function in a context where it is allowed by [class.copy.elision]. This elision is commonly known as the Named Return Value Optimization. For instance, in the example below the compiler cannot elide copies from both v1 and v2, so it elides neither.

```
        std::vector<int> f()
        {
          std::vector<int> v1, v2;
          // ...
          if (cond) return v1;
          else return v2; // warning: not eliding copy
        }
```

−Winfinite−recursion

Warn about infinitely recursive calls. The warning is effective at all optimization levels but requires optimization in order to detect infinite recursion in calls between two or more functions. **−Winfinite−recursion** is included in **−Wall**.

Compare with **−Wanalyzer−infinite−recursion** which provides a similar diagnostic, but is implemented in a different way (as part of **−fanalyzer**).

−Winit−self (C, C++, Objective−C and Objective−C++ only)

Warn about uninitialized variables that are initialized with themselves. Note this option can only be used with the **−Wuninitialized** option.

For example, GCC warns about i being uninitialized in the following snippet only when **−Winit−self** has been specified:

```
        int f()
        {
          int i = i;
          return i;
        }
```

This warning is enabled by **−Wall** in C++.

−Wno−implicit−int (C and Objective−C only)

This option controls warnings when a declaration does not specify a type. This warning is enabled by default, as an error, in C99 and later dialects of C, and also by **−Wall**. The error can be downgraded to a warning using **−fpermissive** (along with certain other errors), or for this error alone, with **−Wno−error=implicit−int**.

This warning is upgraded to an error by **−pedantic−errors**.

−Wno−implicit−function−declaration (C and Objective−C only)

This option controls warnings when a function is used before being declared. This warning is enabled by default, as an error, in C99 and later dialects of C, and also by **−Wall**. The error can be downgraded to a warning using **−fpermissive** (along with certain other errors), or for this error alone, with **−Wno−error=implicit−function−declaration**.

This warning is upgraded to an error by **−pedantic−errors**.

−Wimplicit (C and Objective−C only)

Same as **−Wimplicit−int** and **−Wimplicit−function−declaration**. This warning is enabled by **−Wall**.

−Whardened

Warn when **−fhardened** did not enable an option from its set (for which see **−fhardened**). For instance, using **−fhardened** and **−fstack−protector** at the same time on the command line causes **−Whardened** to warn because **−fstack−protector−strong** will not be enabled by **−fhardened**.

This warning is enabled by default and has effect only when **−fhardened** is enabled.

−Wimplicit−fallthrough

−Wimplicit−fallthrough is the same as **−Wimplicit−fallthrough=3** and **−Wno−implicit−fallthrough** is the same as **−Wimplicit−fallthrough=0**.

−Wimplicit−fallthrough=_n_

Warn when a switch case falls through. For example:

```
switch (cond)
  {
  case 1:
    a = 1;
    break;
  case 2:
    a = 2;
  case 3:
    a = 3;
    break;
  }
```

This warning does not warn when the last statement of a case cannot fall through, e.g. when there is a return statement or a call to function declared with the noreturn attribute. **−Wimplicit−fallthrough=** also takes into account control flow statements, such as ifs, and only warns when appropriate. E.g.

```
switch (cond)
  {
  case 1:
    if (i > 3) {
      bar (5);
      break;
    } else if (i < 1) {
      bar (0);
    } else
      return;
  default:
    ...
  }
```

Since there are occasions where a switch case fall through is desirable, GCC provides an attribute, `__attribute__ ((fallthrough))`, that is to be used along with a null statement to suppress this warning that would normally occur:

```
          switch (cond)
            {
            case 1:
              bar (0);
              __attribute__ ((fallthrough));
            default:
              ...
            }
```

C++17 provides a standard way to suppress the **−Wimplicit−fallthrough** warning using `[[fallthrough]];` instead of the GNU attribute. In C++11 or C++14 users can use `[[gnu::fallthrough]];`, which is a GNU extension. Instead of these attributes, it is also possible to add a fallthrough comment to silence the warning. The whole body of the C or C++ style comment should match the given regular expressions listed below. The option argument *n* specifies what kind of comments are accepted:

*<**−Wimplicit−fallthrough=0** disables the warning altogether.>
*<**−Wimplicit−fallthrough=1** matches `.*` regular>
    expression, any comment is used as fallthrough comment.

*<**−Wimplicit−fallthrough=2** case insensitively matches>
    `.*falls?[ \t−]*thr(ough|u).*` regular expression.

*<**−Wimplicit−fallthrough=3** case sensitively matches one of the>
    following regular expressions:

    `*<−fallthrough>`
    `*<@fallthrough@>`
    `*<lint −fallthrough[ \t]*>`
    `*<[ \t.!]*(ELSE,? |INTENTIONAL(LY)? )?FALL(S | |−)?THR(OUGH|U)[`
    `\t.!]*(−[^\n\r]*)?>`
    `*<[ \t.!]*(Else,? |Intentional(ly)? )?Fall((s |`
    `|−)[Tt]|t)hr(ough|u)[ \t.!]*(−[^\n\r]*)?>`
    `*<[ \t.!]*([Ee]lse,? |[Ii]ntentional(ly)? )?fall(s |`
    `|−)?thr(ough|u)[ \t.!]*(−[^\n\r]*)?>`

*<**−Wimplicit−fallthrough=4** case sensitively matches one of the>
    following regular expressions:

    `*<−fallthrough>`
    `*<@fallthrough@>`
    `*<lint −fallthrough[ \t]*>`
    `*<[ \t]*FALLTHR(OUGH|U)[ \t]*>`

*<**−Wimplicit−fallthrough=5** doesn't recognize any comments as>
    fallthrough comments, only attributes disable the warning.

The comment needs to be followed after optional whitespace and other comments by `case` or `default` keywords or by a user label that precedes some `case` or `default` label.

```
          switch (cond)
            {
            case 1:
              bar (0);
              /* FALLTHRU */
            default:
              ...
            }
```

The **−Wimplicit−fallthrough=3** warning is enabled by **−Wextra**.

−Wno−if−not−aligned (C, C++, Objective−C and Objective−C++ only)
>   Control if warnings triggered by the `warn_if_not_aligned` attribute should be issued. These
>   warnings are enabled by default.

−Wignored−qualifiers (C and C++ only)
>   Warn if the return type of a function has a type qualifier such as `const`. For ISO C such a type
>   qualifier has no effect, since the value returned by a function is not an lvalue. For C++, the warning is
>   only emitted for scalar types or `void`. ISO C prohibits qualified `void` return types on function
>   definitions, so such return types always receive a warning even without this option.
>
>   This warning is also enabled by **−Wextra**.

−Wno−ignored−attributes (C and C++ only)
>   This option controls warnings when an attribute is ignored. This is different from the **−Wattributes**
>   option in that it warns whenever the compiler decides to drop an attribute, not that the attribute is
>   either unknown, used in a wrong place, etc. This warning is enabled by default.

−Wmain
>   Warn if the type of `main` is suspicious. `main` should be a function with external linkage, returning
>   int, taking either zero arguments, two, or three arguments of appropriate types. This warning is
>   enabled by default in C++ and is enabled by either **−Wall** or **−Wpedantic**.
>
>   This warning is upgraded to an error by **−pedantic−errors**.

−Wmisleading−indentation (C and C++ only)
>   Warn when the indentation of the code does not reflect the block structure. Specifically, a warning is
>   issued for `if`, `else`, `while`, and `for` clauses with a guarded statement that does not use braces,
>   followed by an unguarded statement with the same indentation.
>
>   In the following example, the call to "bar" is misleadingly indented as if it were guarded by the "if"
>   conditional.
>
> ```
> if (some_condition ())
>   foo ();
>   bar ();  /* Gotcha: this is not guarded by the "if".  */
> ```
>
>   In the case of mixed tabs and spaces, the warning uses the **−ftabstop=** option to determine if the
>   statements line up (defaulting to 8).
>
>   The warning is not issued for code involving multiline preprocessor logic such as the following
>   example.
>
> ```
> if (flagA)
>   foo (0);
> #if SOME_CONDITION_THAT_DOES_NOT_HOLD
>   if (flagB)
> #endif
>     foo (1);
> ```
>
>   The warning is not issued after a `#line` directive, since this typically indicates autogenerated code,
>   and no assumptions can be made about the layout of the file that the directive references.
>
>   This warning is enabled by **−Wall** in C and C++.

−Wmissing−attributes
>   Warn when a declaration of a function is missing one or more attributes that a related function is
>   declared with and whose absence may adversely affect the correctness or efficiency of generated code.
>   For example, the warning is issued for declarations of aliases that use attributes to specify less
>   restrictive requirements than those of their targets. This typically represents a potential optimization
>   opportunity. By contrast, the **−Wattribute−alias=2** option controls warnings issued when the alias is
>   more restrictive than the target, which could lead to incorrect code generation. Attributes considered
>   include `alloc_align`, `alloc_size`, `cold`, `const`, `hot`, `leaf`, `malloc`, `nonnull`,

noreturn, nothrow, pure, returns_nonnull, and returns_twice.

In C++, the warning is issued when an explicit specialization of a primary template declared with attribute alloc_align, alloc_size, assume_aligned, format, format_arg, malloc, or nonnull is declared without it. Attributes deprecated, error, and warning suppress the warning..

You can use the copy attribute to apply the same set of attributes to a declaration as that on another declaration without explicitly enumerating the attributes. This attribute can be applied to declarations of functions, variables, or types.

−Wmissing−attributes is enabled by **−Wall**.

For example, since the declaration of the primary function template below makes use of both attribute malloc and alloc_size the declaration of the explicit specialization of the template is diagnosed because it is missing one of the attributes.

```
template <class T>
T* __attribute__ ((malloc, alloc_size (1)))
allocate (size_t);

template <>
void* __attribute__ ((malloc))   // missing alloc_size
allocate<void> (size_t);
```

−Wmissing−braces
Warn if an aggregate or union initializer is not fully bracketed. In the following example, the initializer for a is not fully bracketed, but that for b is fully bracketed.

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

This warning is enabled by **−Wall**.

−Wmissing−include−dirs (C, C++, Objective−C, Objective−C++ and Fortran only)
Warn if a user−supplied include directory does not exist. This option is disabled by default for C, C++, Objective−C and Objective−C++. For Fortran, it is partially enabled by default by warning for −I and −J, only.

−Wno−missing−profile
This option controls warnings if feedback profiles are missing when using the **−fprofile−use** option. This option diagnoses those cases where a new function or a new file is added between compiling with **−fprofile−generate** and with **−fprofile−use**, without regenerating the profiles. In these cases, the profile feedback data files do not contain any profile feedback information for the newly added function or file respectively. Also, in the case when profile count data (.gcda) files are removed, GCC cannot use any profile feedback information. In all these cases, warnings are issued to inform you that a profile generation step is due. Ignoring the warning can result in poorly optimized code. **−Wno−missing−profile** can be used to disable the warning, but this is not recommended and should be done only when non−existent profile data is justified.

−Wmismatched−dealloc
Warn for calls to deallocation functions with pointer arguments returned from allocation functions for which the former isn't a suitable deallocator. A pair of functions can be associated as matching allocators and deallocators by use of attribute malloc. Unless disabled by the **−fno−builtin** option the standard functions calloc, malloc, realloc, and free, as well as the corresponding forms of C++ operator new and operator delete are implicitly associated as matching allocators and deallocators. In the following example mydealloc is the deallocator for pointers returned from myalloc.

```
void mydealloc (void*);
```

```
                __attribute__ ((malloc (mydealloc, 1))) void*
                myalloc (size_t);

                void f (void)
                {
                  void *p = myalloc (32);
                  // ...use p...
                  free (p);    // warning: not a matching deallocator for myalloc
                  mydealloc (p);    // ok
                }
```

In C++, the related option **−Wmismatched−new−delete** diagnoses mismatches involving either `operator new` or `operator delete`.

Option **−Wmismatched−dealloc** is included in **−Wall**.

−Wmultistatement−macros
Warn about unsafe multiple statement macros that appear to be guarded by a clause such as `if`, `else`, `for`, `switch`, or `while`, in which only the first statement is actually guarded after the macro is expanded.

For example:

```
        #define DOIT x++; y++
        if (c)
          DOIT;
```

will increment `y` unconditionally, not just when `c` holds. The can usually be fixed by wrapping the macro in a do−while loop:

```
        #define DOIT do { x++; y++; } while (0)
        if (c)
          DOIT;
```

This warning is enabled by **−Wall** in C and C++.

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

Also warn if a comparison like `x<=y<=z` appears; this is equivalent to `(x<=y ? 1 : 0) <= z`, which is a different interpretation from that of ordinary mathematical notation.

Also warn for dangerous uses of the GNU extension to `?:` with omitted middle operand. When the condition in the `?:` operator is a boolean expression, the omitted value is always 1. Often programmers expect it to be a value computed inside the conditional expression instead.

For C++ this also warns for some cases of unnecessary parentheses in declarations, which can indicate an attempt at a function call instead of a declaration:

```
        {
          // Declares a local variable called mymutex.
          std::unique_lock<std::mutex> (mymutex);
          // User meant std::unique_lock<std::mutex> lock (mymutex);
        }
```

This warning is enabled by **−Wall**.

−Wno−self−move (C++ and Objective−C++ only)
This warning warns when a value is moved to itself with `std::move`. Such a `std::move` typically has no effect.

```
struct T {
...
};
void fn()
{
  T t;
  ...
  t = std::move (t);
}
```

This warning is enabled by **−Wall**.

−Wsequence−point

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

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

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

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

The C++17 standard will define the order of evaluation of operands in more cases: in particular it requires that the right−hand side of an assignment be evaluated before the left−hand side, so the above examples are no longer undefined. But this option will still warn about them, to help people avoid writing code that is undefined in C and earlier revisions of C++.

The standard is worded confusingly, therefore there is some debate over the precise meaning of the sequence point rules in subtle cases. Links to discussions of the problem, including proposed formal definitions, may be found on the GCC readings page, at <**https://gcc.gnu.org/readings.html**>.

This warning is enabled by **−Wall** for C and C++.

−Wno−return−local−addr

Do not warn about returning a pointer (or in C++, a reference) to a variable that goes out of scope after the function returns.

−Wreturn−mismatch

Warn about return statements without an expressions in functions which do not return `void`. Also warn about a `return` statement with an expression in a function whose return type is `void`, unless the expression type is also `void`. As a GNU extension, the latter case is accepted without a warning unless **−Wpedantic** is used.

Attempting to use the return value of a non−void function other than main that flows off the end by reaching the closing curly brace that terminates the function is undefined.

This warning is specific to C and enabled by default. In C99 and later language dialects, it is treated as an error. It can be downgraded to a warning using **−fpermissive** (along with other warnings), or for just this warning, with **−Wno−error=return−mismatch**.

−Wreturn−type

Warn whenever a function is defined with a return type that defaults to int (unless **−Wimplicit−int** is active, which takes precedence). Also warn if execution may reach the end of the function body, or if the function does not contain any return statement at all.

Attempting to use the return value of a non−void function other than main that flows off the end by reaching the closing curly brace that terminates the function is undefined.

Unlike in C, in C++, flowing off the end of a non−void function other than main results in undefined behavior even when the value of the function is not used.

This warning is enabled by default in C++ and by **−Wall** otherwise.

−Wno−shift−count−negative

Controls warnings if a shift count is negative. This warning is enabled by default.

−Wno−shift−count−overflow

Controls warnings if a shift count is greater than or equal to the bit width of the type. This warning is enabled by default.

−Wshift−negative−value

Warn if left shifting a negative value. This warning is enabled by **−Wextra** in C99 (and newer) and C++11 to C++17 modes.

−Wno−shift−overflow
−Wshift−overflow=*n*

These options control warnings about left shift overflows.

−Wshift−overflow=1

This is the warning level of **−Wshift−overflow** and is enabled by default in C99 and C++11 modes (and newer). This warning level does not warn about left−shifting 1 into the sign bit. (However, in C, such an overflow is still rejected in contexts where an integer constant expression is required.) No warning is emitted in C++20 mode (and newer), as signed left shifts always wrap.

−Wshift−overflow=2

This warning level also warns about left−shifting 1 into the sign bit, unless C++14 mode (or newer) is active.

−Wswitch

Warn whenever a switch statement has an index of enumerated 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 that do not correspond to enumerators also provoke warnings when this option is used, unless the enumeration is marked with the flag_enum attribute. This warning is enabled by **−Wall**.

−Wswitch−default

Warn whenever a switch statement does not have a default case.

−Wswitch−enum

Warn whenever a switch statement has an index of enumerated type and lacks a case for one or more of the named codes of that enumeration. case labels that do not correspond to enumerators also provoke warnings when this option is used, unless the enumeration is marked with the flag_enum attribute. The only difference between **−Wswitch** and this option is that this option gives a warning about an omitted enumeration code even if there is a default label.

−Wno−switch−bool

Do not warn when a `switch` statement has an index of boolean type and the case values are outside the range of a boolean type. It is possible to suppress this warning by casting the controlling expression to a type other than `bool`. For example:

```
switch ((int) (a == 4))
  {
  ...
  }
```

This warning is enabled by default for C and C++ programs.

−Wno−switch−outside−range

This option controls warnings when a `switch` case has a value that is outside of its respective type range. This warning is enabled by default for C and C++ programs.

−Wno−switch−unreachable

Do not warn when a `switch` statement contains statements between the controlling expression and the first case label, which will never be executed. For example:

```
switch (cond)
  {
   i = 15;
  ...
   case 5:
  ...
  }
```

−Wswitch−unreachable does not warn if the statement between the controlling expression and the first case label is just a declaration:

```
switch (cond)
  {
   int i;
  ...
   case 5:
   i = 5;
  ...
  }
```

This warning is enabled by default for C and C++ programs.

−Wsync−nand (C and C++ only)

Warn when `__sync_fetch_and_nand` and `__sync_nand_and_fetch` built−in functions are used. These functions changed semantics in GCC 4.4.

−Wtrivial−auto−var−init

Warn when `−ftrivial−auto−var−init` cannot initialize the automatic variable. A common situation is an automatic variable that is declared between the controlling expression and the first case label of a `switch` statement.

−Wunused−but−set−parameter

Warn whenever a function parameter is assigned to, but otherwise unused (aside from its declaration).

To suppress this warning use the `unused` attribute.

This warning is also enabled by **−Wunused** together with **−Wextra**.

−Wunused−but−set−variable

Warn whenever a local variable is assigned to, but otherwise unused (aside from its declaration). This warning is enabled by **−Wall**.

To suppress this warning use the `unused` attribute.

This warning is also enabled by **−Wunused**, which is enabled by **−Wall**.

−Wunused−function
Warn whenever a static function is declared but not defined or a non−inline static function is unused. This warning is enabled by **−Wall**.

−Wunused−label
Warn whenever a label is declared but not used. This warning is enabled by **−Wall**.

To suppress this warning use the `unused` attribute.

−Wunused−local−typedefs (C, Objective−C, C++ and Objective−C++ only)
Warn when a typedef locally defined in a function is not used. This warning is enabled by **−Wall**.

−Wunused−parameter
Warn whenever a function parameter is unused aside from its declaration. This option is not enabled by `−Wunused` unless `−Wextra` is also specified.

To suppress this warning use the `unused` attribute.

−Wno−unused−result
Do not warn if a caller of a function marked with attribute `warn_unused_result` does not use its return value. The default is **−Wunused−result**.

−Wunused−variable
Warn whenever a local or static variable is unused aside from its declaration. This option implies **−Wunused−const−variable=1** for C, but not for C++. This warning is enabled by **−Wall**.

To suppress this warning use the `unused` attribute.

−Wunused−const−variable
−Wunused−const−variable=*n*
Warn whenever a constant static variable is unused aside from its declaration.

To suppress this warning use the `unused` attribute.

> **−Wunused−const−variable=1**
> Warn about unused static const variables defined in the main compilation unit, but not about static const variables declared in any header included.
>
> **−Wunused−const−variable=1** is enabled by either **−Wunused−variable** or **−Wunused** for C, but not for C++. In C this declares variable storage, but in C++ this is not an error since const variables take the place of `#defines`.
>
> **−Wunused−const−variable=2**
> This warning level also warns for unused constant static variables in headers (excluding system headers). It is equivalent to the short form **−Wunused−const−variable**. This level must be explicitly requested in both C and C++ because it might be hard to clean up all headers included.

−Wunused−value
Warn whenever a statement computes a result that is explicitly not used. To suppress this warning cast the unused expression to `void`. This includes an expression−statement or the left−hand side of a comma expression that contains no side effects. For example, an expression such as `x[i,j]` causes a warning, while `x[(void)i,j]` does not.

This warning is enabled by **−Wall**.

−Wunused
All the above **−Wunused** options combined, except those documented as needing to be specified explicitly.

In order to get a warning about an unused function parameter, you must either specify **−Wextra −Wunused** (note that **−Wall** implies **−Wunused**), or separately specify **−Wunused−parameter** and/or **−Wunused−but−set−parameter**.

−Wunused enables only **−Wunused−const−variable=1** rather than **−Wunused−const−variable**, and only for C, not C++.

−Wuse−after−free (C, Objective−C, C++ and Objective−C++ only)
−Wuse−after−free=*n*
Warn about uses of pointers to dynamically allocated objects that have been rendered indeterminate by a call to a deallocation function. The warning is enabled at all optimization levels but may yield different results with optimization than without.

−Wuse−after−free=1
At level 1 the warning attempts to diagnose only unconditional uses of pointers made indeterminate by a deallocation call or a successful call to realloc, regardless of whether or not the call resulted in an actual reallocation of memory. This includes double−free calls as well as uses in arithmetic and relational expressions. Although undefined, uses of indeterminate pointers in equality (or inequality) expressions are not diagnosed at this level.

−Wuse−after−free=2
At level 2, in addition to unconditional uses, the warning also diagnoses conditional uses of pointers made indeterminate by a deallocation call. As at level 2, uses in equality (or inequality) expressions are not diagnosed. For example, the second call to free in the following function is diagnosed at this level:

```
struct A { int refcount; void *data; };

void release (struct A *p)
{
  int refcount = --p->refcount;
  free (p);
  if (refcount == 0)
    free (p->data);   // warning: p may be used after free
}
```

−Wuse−after−free=3
At level 3, the warning also diagnoses uses of indeterminate pointers in equality expressions. All uses of indeterminate pointers are undefined but equality tests sometimes appear after calls to realloc as an attempt to determine whether the call resulted in relocating the object to a different address. They are diagnosed at a separate level to aid gradually transitioning legacy code to safe alternatives. For example, the equality test in the function below is diagnosed at this level:

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

void grow (int **p, int n)
{
  int **q = (int**)realloc (p, n *= 2);
  if (q == p)
    return;
  adjust_pointers ((int**)q, n);
}
```

To avoid the warning at this level, store offsets into allocated memory instead of pointers. This approach obviates needing to adjust the stored pointers after reallocation.

−Wuse−after−free=2 is included in **−Wall**.

−Wuseless−cast (C, Objective−C, C++ and Objective−C++ only)
Warn when an expression is cast to its own type. This warning does not occur when a class object is converted to a non−reference type as that is a way to create a temporary:

```
                struct S { };
                void g (S&&);
                void f (S&& arg)
                {
                  g (S(arg)); // make arg prvalue so that it can bind to S&&
                }
```

−Wuninitialized

Warn if an object with automatic or allocated storage duration is used without having been initialized. In C++, also warn if a non−static reference or non−static `const` member appears in a class without constructors.

In addition, passing a pointer (or in C++, a reference) to an uninitialized object to a `const`−qualified argument of a built−in function known to read the object is also diagnosed by this warning. (**−Wmaybe−uninitialized** is issued for ordinary functions.)

If you want to warn about code that uses the uninitialized value of the variable in its own initializer, use the **−Winit−self** option.

These warnings occur for individual uninitialized elements of structure, union or array variables as well as for variables that are uninitialized as a whole. They do not occur for variables or elements declared `volatile`. Because these warnings depend on optimization, the exact variables or elements for which there are warnings depend on the precise optimization options and version of GCC used.

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.

In C++, this warning also warns about using uninitialized objects in member−initializer−lists. For example, GCC warns about b being uninitialized in the following snippet:

```
                struct A {
                  int a;
                  int b;
                  A() : a(b) { }
                };
```

−Wno−invalid−memory−model

This option controls warnings for invocations of **__atomic Builtins**, **__sync Builtins**, and the C11 atomic generic functions with a memory consistency argument that is either invalid for the operation or outside the range of values of the `memory_order` enumeration. For example, since the `__atomic_store` and `__atomic_store_n` built−ins are only defined for the relaxed, release, and sequentially consistent memory orders the following code is diagnosed:

```
                void store (int *i)
                {
                  __atomic_store_n (i, 0, memory_order_consume);
                }
```

−Winvalid−memory−model is enabled by default.

−Wmaybe−uninitialized

For an object with automatic or allocated storage duration, if there exists a path from the function entry to a use of the object that is initialized, but there exist some other paths for which the object is not initialized, the compiler emits a warning if it cannot prove the uninitialized paths are not executed at run time.

In addition, passing a pointer (or in C++, a reference) to an uninitialized object to a `const`−qualified function argument is also diagnosed by this warning. (**−Wuninitialized** is issued for built−in functions known to read the object.) Annotating the function with attribute `access (none)`

indicates that the argument isn't used to access the object and avoids the warning.

These warnings are only possible in optimizing compilation, because otherwise GCC does not keep track of the state of variables.

These warnings are made optional because GCC may not be able to determine when the code is correct in spite of appearing to have an error. Here is one example of how this can happen:

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

If the value of y is always 1, 2 or 3, then x is always initialized, but GCC doesn't know this. To suppress the warning, you need to provide a default case with **assert** (0) or similar code.

This option also warns when a non−volatile automatic variable might be changed by a call to longjmp. 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 that would cause a problem.

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

This warning is enabled by **−Wall** or **−Wextra**.

−Wunknown−pragmas
> Warn when a #pragma directive is encountered that is not understood by GCC. If this command−line option is used, warnings are even issued for unknown pragmas in system header files. This is not the case if the warnings are only enabled by the **−Wall** command−line option.

−Wno−pragmas
> Do not warn about misuses of pragmas, such as incorrect parameters, invalid syntax, or conflicts between pragmas. See also **−Wunknown−pragmas**.

−Wno−pragma−once−outside−header
> Do not warn when #pragma once is used in a file that is not a header file, such as a main file.

−Wno−prio−ctor−dtor
> Do not warn if a priority from 0 to 100 is used for constructor or destructor. The use of constructor and destructor attributes allow you to assign a priority to the constructor/destructor to control its order of execution before main is called or after it returns. The priority values must be greater than 100 as the compiler reserves priority values between 0−−100 for the implementation.

−Wstrict−aliasing
> This option is only active when **−fstrict−aliasing** is active. It warns about code that might break the strict aliasing rules that the compiler is using for optimization. The warning does not catch all cases, but does attempt to catch the more common pitfalls. It is included in **−Wall**. It is equivalent to **−Wstrict−aliasing=3**

−Wstrict−aliasing=n
> This option is only active when **−fstrict−aliasing** is active. It warns about code that might break the strict aliasing rules that the compiler is using for optimization. Higher levels correspond to higher

accuracy (fewer false positives). Higher levels also correspond to more effort, similar to the way **−O** works. **−Wstrict−aliasing** is equivalent to **−Wstrict−aliasing=3**.

Level 1: Most aggressive, quick, least accurate. Possibly useful when higher levels do not warn but **−fstrict−aliasing** still breaks the code, as it has very few false negatives. However, it has many false positives. Warns for all pointer conversions between possibly incompatible types, even if never dereferenced. Runs in the front end only.

Level 2: Aggressive, quick, not too precise. May still have many false positives (not as many as level 1 though), and few false negatives (but possibly more than level 1). Unlike level 1, it only warns when an address is taken. Warns about incomplete types. Runs in the front end only.

Level 3 (default for **−Wstrict−aliasing**): Should have very few false positives and few false negatives. Slightly slower than levels 1 or 2 when optimization is enabled. Takes care of the common pun+dereference pattern in the front end: `*(int*)&some_float`. If optimization is enabled, it also runs in the back end, where it deals with multiple statement cases using flow−sensitive points−to information. Only warns when the converted pointer is dereferenced. Does not warn about incomplete types.

−Wstrict−overflow
−Wstrict−overflow=*n*

This option is only active when signed overflow is undefined. It warns about cases where the compiler optimizes based on the assumption that signed overflow does not occur. Note that it does not warn about all cases where the code might overflow: it only warns about cases where the compiler implements some optimization. Thus this warning depends on the optimization level.

An optimization that assumes that signed overflow does not occur is perfectly safe if the values of the variables involved are such that overflow never does, in fact, occur. Therefore this warning can easily give a false positive: a warning about code that is not actually a problem. To help focus on important issues, several warning levels are defined. No warnings are issued for the use of undefined signed overflow when estimating how many iterations a loop requires, in particular when determining whether a loop will be executed at all.

−Wstrict−overflow=1

Warn about cases that are both questionable and easy to avoid. For example the compiler simplifies `x + 1 > x` to `1`. This level of **−Wstrict−overflow** is enabled by **−Wall**; higher levels are not, and must be explicitly requested.

−Wstrict−overflow=2

Also warn about other cases where a comparison is simplified to a constant. For example: `abs (x) >= 0`. This can only be simplified when signed integer overflow is undefined, because `abs (INT_MIN)` overflows to `INT_MIN`, which is less than zero. **−Wstrict−overflow** (with no level) is the same as **−Wstrict−overflow=2**.

−Wstrict−overflow=3

Also warn about other cases where a comparison is simplified. For example: `x + 1 > 1` is simplified to `x > 0`.

−Wstrict−overflow=4

Also warn about other simplifications not covered by the above cases. For example: `(x * 10) / 5` is simplified to `x * 2`.

−Wstrict−overflow=5

Also warn about cases where the compiler reduces the magnitude of a constant involved in a comparison. For example: `x + 2 > y` is simplified to `x + 1 >= y`. This is reported only at the highest warning level because this simplification applies to many comparisons, so this warning level gives a very large number of false positives.

−Wstring−compare

Warn for calls to `strcmp` and `strncmp` whose result is determined to be either zero or non−zero in tests for such equality owing to the length of one argument being greater than the size of the array the

other argument is stored in (or the bound in the case of strncmp). Such calls could be mistakes. For example, the call to strcmp below is diagnosed because its result is necessarily non−zero irrespective of the contents of the array a.

```
extern char a[4];
void f (char *d)
{
  strcpy (d, "string");
  ...
  if (0 == strcmp (a, d))   // cannot be true
    puts ("a and d are the same");
}
```

−Wstring−compare is enabled by **−Wextra**.

−Wno−stringop−overflow
−Wstringop−overflow
−Wstringop−overflow=*type*
Warn for code that can be statically determined to cause buffer overflows or memory overruns, such as calls to memcpy and strcpy that overflow the destination buffer. The optional argument is one greater than the type of Object Size Checking to perform to determine the size of the destination. The argument is meaningful only for string functions that operate on character arrays; raw memory functions like memcpy always use type−zero Object Size Checking.

The option also warns for calls that specify a size in excess of the largest possible object or at most SIZE_MAX / 2 bytes.

The option produces the best results with optimization enabled but can detect a small subset of simple buffer overflows even without optimization in calls to the GCC built−in functions like __builtin_memcpy that correspond to the standard functions. In any case, the option warns about just a subset of buffer overflows detected by the corresponding overflow checking built−ins, such as __builtin___memcpy_chk, which can perform run−time checking if the access cannot be identified as safe at compile time.

For example, the option issues a warning for the strcpy call below because it copies at least 5 characters (the string "blue" including the terminating NUL) into the buffer of size 4.

```
enum Color { blue, purple, yellow };
const char* f (enum Color clr)
{
  static char buf [4];
  const char *str;
  switch (clr)
    {
      case blue: str = "blue"; break;
      case purple: str = "purple"; break;
      case yellow: str = "yellow"; break;
    }

  return strcpy (buf, str);   // warning here
}
```

The effect of this option is not limited to string or memory manipulation functions. In this example, a warning is diagnosed because a 1−element array is passed to a function requiring at least a 4−element array argument:

```
void f (int[static 4]);

void g (void)
```

```
            {
              int *p = (int *) malloc (1 * sizeof(int));
              f (p);   // warning here
            }
```

Option **−Wstringop−overflow=2** is enabled by default.

−Wstringop−overflow
−Wstringop−overflow=1
   The **−Wstringop−overflow=1** option uses type−zero Object Size Checking to determine the sizes
   of destination objects. At this setting the option does not warn for writes past the end of
   subobjects of larger objects accessed by pointers unless the size of the largest surrounding object
   is known. When the destination may be one of several objects it is assumed to be the largest one
   of them. On Linux systems, when optimization is enabled at this setting the option warns for the
   same code as when the _FORTIFY_SOURCE macro is defined to a non−zero value.

−Wstringop−overflow=2
   The **−Wstringop−overflow=2** option uses type−one Object Size Checking to determine the sizes
   of destination objects. At this setting the option warns about overflows when writing to members
   of the largest complete objects whose exact size is known. However, it does not warn for
   excessive writes to the same members of unknown objects referenced by pointers since they may
   point to arrays containing unknown numbers of elements. This is the default setting of the
   option.

−Wstringop−overflow=3
   The **−Wstringop−overflow=3** option uses type−two Object Size Checking to determine the sizes
   of destination objects. At this setting the option warns about overflowing the smallest object or
   data member. This is the most restrictive setting of the option that may result in warnings for safe
   code.

−Wstringop−overflow=4
   The **−Wstringop−overflow=4** option uses type−three Object Size Checking to determine the
   sizes of destination objects. At this setting the option warns about overflowing any data
   members, and when the destination is one of several objects it uses the size of the largest of them
   to decide whether to issue a warning. Similarly to **−Wstringop−overflow=3** this setting of the
   option may result in warnings for benign code.

−Wno−stringop−overread
   Warn for calls to string manipulation functions such as memchr, or strcpy that are determined to
   read past the end of the source sequence.

   Option **−Wstringop−overread** is enabled by default.

−Wno−stringop−truncation
   Do not warn for calls to bounded string manipulation functions such as strncat, strncpy, and
   stpncpy that may either truncate the copied string or leave the destination unchanged.

   In the following example, the call to strncat specifies a bound that is less than the length of the
   source string. As a result, the copy of the source will be truncated and so the call is diagnosed. To
   avoid the warning use bufsize − strlen (buf) − 1) as the bound.

```
            void append (char *buf, size_t bufsize)
            {
              strncat (buf, ".txt", 3);
            }
```

   As another example, the following call to strncpy results in copying to d just the characters
   preceding the terminating NUL, without appending the NUL to the end. Assuming the result of
   strncpy is necessarily a NUL−terminated string is a common mistake, and so the call is diagnosed.
   To avoid the warning when the result is not expected to be NUL−terminated, call memcpy instead.

```
void copy (char *d, const char *s)
{
  strncpy (d, s, strlen (s));
}
```

In the following example, the call to `strncpy` specifies the size of the destination buffer as the bound. If the length of the source string is equal to or greater than this size the result of the copy will not be NUL−terminated. Therefore, the call is also diagnosed. To avoid the warning, specify `sizeof buf - 1` as the bound and set the last element of the buffer to `NUL`.

```
void copy (const char *s)
{
  char buf[80];
  strncpy (buf, s, sizeof buf);
  ...
}
```

In situations where a character array is intended to store a sequence of bytes with no terminating `NUL` such an array may be annotated with attribute `nonstring` to avoid this warning. Such arrays, however, are not suitable arguments to functions that expect `NUL`−terminated strings. To help detect accidental misuses of such arrays GCC issues warnings unless it can prove that the use is safe.

−Wstrict−flex−arrays (C and C++ only)
Warn about improper usages of flexible array members according to the *level* of the `strict_flex_array (level)` attribute attached to the trailing array field of a structure if it's available, otherwise according to the *level* of the option **−fstrict−flex−arrays=***level*. `-Wstrict-flex-arrays` is effective only when *level* is greater than 0.

When *level*=1, warnings are issued for a trailing array reference of a structure that have 2 or more elements if the trailing array is referenced as a flexible array member.

When *level*=2, in addition to *level*=1, additional warnings are issued for a trailing one−element array reference of a structure if the array is referenced as a flexible array member.

When *level*=3, in addition to *level*=2, additional warnings are issued for a trailing zero−length array reference of a structure if the array is referenced as a flexible array member.

This option is more effective when **−ftree−vrp** is active (the default for **−O2** and above) but some warnings may be diagnosed even without optimization.

−Wsuggest−attribute=*attribute−name*
Warn for cases where adding an attribute may be beneficial. The *attribute−name*s currently supported are listed below.

−Wsuggest−attribute=pure
−Wsuggest−attribute=const
−Wsuggest−attribute=noreturn
−Wmissing−noreturn
−Wsuggest−attribute=malloc
−Wsuggest−attribute=returns_nonnull
Warn about functions that might be candidates for attributes `pure`, `const`, `noreturn`, `malloc` or `returns_nonnull`. The compiler only warns for functions visible in other compilation units or (in the case of `pure` and `const`) if it cannot prove that the function returns normally. A function returns normally if it doesn't contain an infinite loop or return abnormally by throwing, calling `abort` or trapping. This analysis requires option **−fipa−pure−const**, which is enabled by default at **−O** and higher. Higher optimization levels improve the accuracy of the analysis.

−Wsuggest−attribute=format
−Wmissing−format−attribute
> Warn about function pointers that might be candidates for `format` attributes. Note these are only possible candidates, not absolute ones. GCC guesses that function pointers with `format` attributes that are used in assignment, initialization, parameter passing or return statements should have a corresponding `format` attribute in the resulting type. I.e. the left−hand side of the assignment or initialization, the type of the parameter variable, or the return type of the containing function respectively should also have a `format` attribute to avoid the warning.
>
> GCC also warns about function definitions that might be candidates for `format` attributes. Again, these are only possible candidates. GCC guesses that `format` attributes might be appropriate for any function that calls a function like `vprintf` or `vscanf`, but this might not always be the case, and some functions for which `format` attributes are appropriate may not be detected.

−Wsuggest−attribute=cold
> Warn about functions that might be candidates for `cold` attribute. This is based on static detection and generally only warns about functions which always leads to a call to another `cold` function such as wrappers of C++ `throw` or fatal error reporting functions leading to `abort`.

−Walloc−size
> Warn about calls to allocation functions decorated with attribute `alloc_size` that specify insufficient size for the target type of the pointer the result is assigned to, including those to the built−in forms of the functions `aligned_alloc`, `alloca`, `calloc`, `malloc`, and `realloc`.

−Walloc−zero
> Warn about calls to allocation functions decorated with attribute `alloc_size` that specify zero bytes, including those to the built−in forms of the functions `aligned_alloc`, `alloca`, `calloc`, `malloc`, and `realloc`. Because the behavior of these functions when called with a zero size differs among implementations (and in the case of `realloc` has been deprecated) relying on it may result in subtle portability bugs and should be avoided.

−Wcalloc−transposed−args
> Warn about calls to allocation functions decorated with attribute `alloc_size` with two arguments, which use `sizeof` operator as the earlier size argument and don't use it as the later size argument. This is a coding style warning. The first argument to `calloc` is documented to be number of elements in array, while the second argument is size of each element, so `calloc (n, sizeof (int))` is preferred over `calloc (sizeof (int), n)`. If `sizeof` in the earlier argument and not the latter is intentional, the warning can be suppressed by using `calloc (sizeof (struct S) + 0, n)` or `calloc (1 * sizeof (struct S), 4)` or using `sizeof` in the later argument as well.

−Walloc−size−larger−than=*byte−size*
> Warn about calls to functions decorated with attribute `alloc_size` that attempt to allocate objects larger than the specified number of bytes, or where the result of the size computation in an integer type with infinite precision would exceed the value of **PTRDIFF_MAX** on the target. **−Walloc−size−larger−than=PTRDIFF_MAX** is enabled by default. Warnings controlled by the option can be disabled either by specifying *byte−size* of **SIZE_MAX** or more or by **−Wno−alloc−size−larger−than**.

−Wno−alloc−size−larger−than
> Disable **−Walloc−size−larger−than=** warnings. The option is equivalent to **−Walloc−size−larger−than=SIZE_MAX** or larger.

−Walloca
> This option warns on all uses of `alloca` in the source.

−Walloca−larger−than=*byte−size*
> This option warns on calls to `alloca` with an integer argument whose value is either zero, or that is not bounded by a controlling predicate that limits its value to at most *byte−size*. It also warns for calls

to `alloca` where the bound value is unknown. Arguments of non−integer types are considered unbounded even if they appear to be constrained to the expected range.

For example, a bounded case of `alloca` could be:

```
void func (size_t n)
{
  void *p;
  if (n <= 1000)
    p = alloca (n);
  else
    p = malloc (n);
  f (p);
}
```

In the above example, passing `−Walloca−larger−than=1000` would not issue a warning because the call to `alloca` is known to be at most 1000 bytes. However, if `−Walloca−larger−than=500` were passed, the compiler would emit a warning.

Unbounded uses, on the other hand, are uses of `alloca` with no controlling predicate constraining its integer argument. For example:

```
void func ()
{
  void *p = alloca (n);
  f (p);
}
```

If `−Walloca−larger−than=500` were passed, the above would trigger a warning, but this time because of the lack of bounds checking.

Note, that even seemingly correct code involving signed integers could cause a warning:

```
void func (signed int n)
{
  if (n < 500)
    {
      p = alloca (n);
      f (p);
    }
}
```

In the above example, *n* could be negative, causing a larger than expected argument to be implicitly cast into the `alloca` call.

This option also warns when `alloca` is used in a loop.

−Walloca−larger−than=PTRDIFF_MAX is enabled by default but is usually only effective when **−ftree−vrp** is active (default for **−O2** and above).

See also **−Wvla−larger−than=byte−size**.

−Wno−alloca−larger−than
Disable **−Walloca−larger−than=** warnings. The option is equivalent to **−Walloca−larger−than=SIZE_MAX** or larger.

−Warith−conversion
Do warn about implicit conversions from arithmetic operations even when conversion of the operands to the same type cannot change their values. This affects warnings from **−Wconversion**, **−Wfloat−conversion**, and **−Wsign−conversion**.

```
void f (char c, int i)
{
  c = c + i; // warns with B<-Wconversion>
  c = c + 1; // only warns with B<-Warith-conversion>
}
```

−Warray−bounds
−Warray−bounds=*n*

Warn about out of bounds subscripts or offsets into arrays. This warning is enabled by **−Wall**. It is more effective when **−ftree−vrp** is active (the default for **−O2** and above) but a subset of instances are issued even without optimization.

By default, the trailing array of a structure will be treated as a flexible array member by **−Warray−bounds** or **−Warray−bounds=***n* if it is declared as either a flexible array member per C99 standard onwards (**[]**), a GCC zero−length array extension (**[0]**), or an one−element array (**[1]**). As a result, out of bounds subscripts or offsets into zero−length arrays or one−element arrays are not warned by default.

You can add the option **−fstrict−flex−arrays** or **−fstrict−flex−arrays=***level* to control how this option treat trailing array of a structure as a flexible array member:

when *level*<=1, no change to the default behavior.

when *level*=2, additional warnings will be issued for out of bounds subscripts or offsets into one−element arrays;

when *level*=3, in addition to *level*=2, additional warnings will be issued for out of bounds subscripts or offsets into zero−length arrays.

−Warray−bounds=1

This is the default warning level of **−Warray−bounds** and is enabled by **−Wall**; higher levels are not, and must be explicitly requested.

−Warray−bounds=2

This warning level also warns about the intermediate results of pointer arithmetic that may yield out of bounds values. This warning level may give a larger number of false positives and is deactivated by default.

−Wunterminated−string−initialization (C and Objective−C only)

Warn about character arrays initialized as unterminated character sequences with a string literal, unless the declaration being initialized has the `nonstring` attribute. For example:

```
char arr[3] = "foo"; /* Warning.  */
char arr2[3] __attribute__((nonstring)) = "bar"; /* No warning.  */
```

This warning is enabled by **−Wextra**. If **−Wc++−compat** is enabled, the warning has slightly different wording and warns even if the declaration being initialized has the `nonstring` warning, as in C++ such initializations are an error.

−Warray−compare

Warn about equality and relational comparisons between two operands of array type. This comparison was deprecated in C++20. For example:

```
int arr1[5];
int arr2[5];
bool same = arr1 == arr2;
```

−Warray−compare is enabled by **−Wall**.

−Warray−parameter
−Warray−parameter=*n*

Warn about redeclarations of functions involving parameters of array or pointer types of inconsistent kinds or forms, and enable the detection of out−of−bounds accesses to such parameters by warnings

such as **−Warray−bounds**.

If the first function declaration uses the array form for a parameter declaration, the bound specified in the array is assumed to be the minimum number of elements expected to be provided in calls to the function and the maximum number of elements accessed by it. Failing to provide arguments of sufficient size or accessing more than the maximum number of elements may be diagnosed by warnings such as **−Warray−bounds** or **−Wstringop−overflow**. At level 1, the warning diagnoses inconsistencies involving array parameters declared using the `T[static N]` form.

For example, the warning triggers for the second declaration of `f` because the first one with the keyword `static` specifies that the array argument must have at least four elements, while the second allows an array of any size to be passed to `f`.

```
void f (int[static 4]);
void f (int[]);               // warning (inconsistent array form)

void g (void)
{
  int *p = (int *)malloc (1 * sizeof (int));
  f (p);                      // warning (array too small)
  ...
}
```

At level 2 the warning also triggers for redeclarations involving any other inconsistency in array or pointer argument forms denoting array sizes. Pointers and arrays of unspecified bound are considered equivalent and do not trigger a warning.

```
void g (int*);
void g (int[]);      // no warning
void g (int[8]);     // warning (inconsistent array bound)
```

−Warray−parameter=2 is included in **−Wall**. The **−Wvla−parameter** option triggers warnings for similar inconsistencies involving Variable Length Array arguments.

The short form of the option **−Warray−parameter** is equivalent to **−Warray−parameter=2**. The negative form **−Wno−array−parameter** is equivalent to **−Warray−parameter=0**.

−Wattribute−alias=_n_
−Wno−attribute−alias
  Warn about declarations using the `alias` and similar attributes whose target is incompatible with the type of the alias.

  −Wattribute−alias=1
    The default warning level of the **−Wattribute−alias** option diagnoses incompatibilities between the type of the alias declaration and that of its target. Such incompatibilities are typically indicative of bugs.

  −Wattribute−alias=2
    At this level **−Wattribute−alias** also diagnoses cases where the attributes of the alias declaration are more restrictive than the attributes applied to its target. These mismatches can potentially result in incorrect code generation. In other cases they may be benign and could be resolved simply by adding the missing attribute to the target. For comparison, see the **−Wmissing−attributes** option, which controls diagnostics when the alias declaration is less restrictive than the target, rather than more restrictive.

    Attributes considered include `alloc_align`, `alloc_size`, `cold`, `const`, `hot`, `leaf`, `malloc`, `nonnull`, `noreturn`, `nothrow`, `pure`, `returns_nonnull`, and `returns_twice`.

  −Wattribute−alias is equivalent to **−Wattribute−alias=1**. This is the default. You can disable these warnings with either **−Wno−attribute−alias** or **−Wattribute−alias=0**.

−Wbidi−chars=[none|unpaired|any|ucn]
Warn about possibly misleading UTF−8 bidirectional control characters in comments, string literals, character constants, and identifiers. Such characters can change left−to−right writing direction into right−to−left (and vice versa), which can cause confusion between the logical order and visual order. This may be dangerous; for instance, it may seem that a piece of code is not commented out, whereas it in fact is.

There are three levels of warning supported by GCC. The default is **−Wbidi−chars=unpaired**, which warns about improperly terminated bidi contexts. **−Wbidi−chars=none** turns the warning off. **−Wbidi−chars=any** warns about any use of bidirectional control characters.

By default, this warning does not warn about UCNs. It is, however, possible to turn on such checking by using **−Wbidi−chars=unpaired,ucn** or **−Wbidi−chars=any,ucn**. Using **−Wbidi−chars=ucn** is valid, and is equivalent to **−Wbidi−chars=unpaired,ucn**, if no previous **−Wbidi−chars=any** was specified.

−Wbool−compare
Warn about boolean expression compared with an integer value different from `true`/`false`. For instance, the following comparison is always false:

```
int n = 5;
...
if ((n > 1) == 2) { ... }
```

This warning is enabled by **−Wall**.

−Wbool−operation
Warn about suspicious operations on expressions of a boolean type. For instance, bitwise negation of a boolean is very likely a bug in the program. For C, this warning also warns about incrementing or decrementing a boolean, which rarely makes sense. (In C++, decrementing a boolean is always invalid. Incrementing a boolean is invalid in C++17, and deprecated otherwise.)

This warning is enabled by **−Wall**.

−Wduplicated−branches
Warn when an if−else has identical branches. This warning detects cases like

```
if (p != NULL)
  return 0;
else
  return 0;
```

It doesn't warn when both branches contain just a null statement. This warning also warn for conditional operators:

```
int i = x ? *p : *p;
```

−Wduplicated−cond
Warn about duplicated conditions in an if−else−if chain. For instance, warn for the following code:

```
if (p->q != NULL) { ... }
else if (p->q != NULL) { ... }
```

−Wframe−address
Warn when the **__builtin_frame_address** or **__builtin_return_address** is called with an argument greater than 0. Such calls may return indeterminate values or crash the program. The warning is included in **−Wall**.

−Wno−discarded−qualifiers (C and Objective−C only)
Do not warn if type qualifiers on pointers are being discarded. Typically, the compiler warns if a `const char *` variable is passed to a function that takes a `char *` parameter. This option can be used to suppress such a warning.

−Wno−discarded−array−qualifiers (C and Objective−C only)
> Do not warn if type qualifiers on arrays which are pointer targets are being discarded. Typically, the compiler warns if a `const int (*)[]` variable is passed to a function that takes a `int (*)[]` parameter. This option can be used to suppress such a warning.

−Wno−incompatible−pointer−types (C and Objective−C only)
> Do not warn when there is a conversion between pointers that have incompatible types. This warning is for cases not covered by **−Wno−pointer−sign**, which warns for pointer argument passing or assignment with different signedness.
>
> By default, in C99 and later dialects of C, GCC treats this issue as an error. The error can be downgraded to a warning using **−fpermissive** (along with certain other errors), or for this error alone, with **−Wno−error=incompatible−pointer−types**.
>
> This warning is upgraded to an error by **−pedantic−errors**.

−Wno−int−conversion (C and Objective−C only)
> Do not warn about incompatible integer to pointer and pointer to integer conversions. This warning is about implicit conversions; for explicit conversions the warnings **−Wno−int−to−pointer−cast** and **−Wno−pointer−to−int−cast** may be used.
>
> By default, in C99 and later dialects of C, GCC treats this issue as an error. The error can be downgraded to a warning using **−fpermissive** (along with certain other errors), or for this error alone, with **−Wno−error=int−conversion**.
>
> This warning is upgraded to an error by **−pedantic−errors**.

−Wzero−as−null−pointer−constant
> Warn when a literal **0** is used as null pointer constant.

−Wzero−length−bounds
> Warn about accesses to elements of zero−length array members that might overlap other members of the same object. Declaring interior zero−length arrays is discouraged because accesses to them are undefined.
>
> For example, the first two stores in function `bad` are diagnosed because the array elements overlap the subsequent members `b` and `c`. The third store is diagnosed by **−Warray−bounds** because it is beyond the bounds of the enclosing object.
>
> ```
> struct X { int a[0]; int b, c; };
> struct X x;
>
> void bad (void)
> {
>   x.a[0] = 0;   // -Wzero-length-bounds
>   x.a[1] = 1;   // -Wzero-length-bounds
>   x.a[2] = 2;   // -Warray-bounds
> }
> ```
>
> Option **−Wzero−length−bounds** is enabled by **−Warray−bounds**.

−Wno−div−by−zero
> Do not warn about compile−time integer division by zero. Floating−point division by zero is not warned about, as it can be a legitimate way of obtaining infinities and NaNs.

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

−Wtautological−compare
    Warn if a self−comparison always evaluates to true or false. This warning detects various mistakes
    such as:

```
int i = 1;
...
if (i > i) { ... }
```

This warning also warns about bitwise comparisons that always evaluate to true or false, for instance:

```
if ((a & 16) == 10) { ... }
```

will always be false.

This warning is enabled by **−Wall**.

−Wtrailing−whitespace
−Wtrailing−whitespace=*kind*
    Warn about trailing whitespace at the end of lines, including inside of comments, but excluding
    trailing whitespace in raw string literals. `-Wtrailing-whitespace` is equivalent to
    `-Wtrailing-whitespace=blanks` and warns just about trailing space and horizontal tab
    characters. `-Wtrailing-whitespace=any` warns about those or trailing form feed or vertical
    tab characters. `-Wno-trailing-whitespace` or `-Wtrailing-whitespace=none` disables
    the warning, which is the default. This is a coding style warning.

−Wleading−whitespace=*kind*
    Warn about style issues in leading whitespace, but not about the amount of indentation. Some projects
    use coding styles where only spaces are used for indentation, others use only tabs, others use zero or
    more tabs (for multiples of `-ftabstop=`*n*) followed by zero or fewer than *n* spaces. No warning is
    emitted on lines which contain solely whitespace (although `-Wtrailing-whitespace=` warning
    might be emitted), no warnings are emitted inside of raw string literals. Warnings are also emitted for
    leading whitespace inside of multi−line comments. `-Wleading-whitespace=spaces` warns
    about leading whitespace other than spaces for projects which want to indent just by spaces.
    `-Wleading-whitespace=tabs` warns about leading whitespace other than horizontal tabs for
    projects which want to indent just by horizontal tabs. `-Wleading-whitespace=blanks` warns
    about leading whitespace other than spaces and horizontal tabs, or about horizontal tab after a space in
    the leading whitespace, or about *n* or more consecutive spaces in leading whitespace (where *n* is
    argument of `-ftabstop=`*n*, 8 by default). `-Wleading-whitespace=none` disables the
    warning, which is the default. This is a coding style warning.

−Wtrampolines
    Warn about trampolines generated for pointers to nested functions. A trampoline is a small piece of
    data or code that is created at run time on the stack when the address of a nested function is taken, and
    is used to call the nested function indirectly. For some targets, it is made up of data only and thus
    requires no special treatment. But, for most targets, it is made up of code and thus requires the stack
    to be made executable in order for the program to work properly.

−Wfloat−equal
    Warn if floating−point values are used in equality comparisons.

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

−Wtraditional (C and Objective−C only)
Warn about certain constructs that behave differently in traditional and ISO C. Also warn about ISO C constructs that have no traditional C equivalent, and/or problematic constructs that should be avoided.

* Macro parameters that appear within string literals in the macro body. In traditional C macro replacement takes place within string literals, but in ISO C it does not.

* In traditional C, some preprocessor directives did not exist. Traditional preprocessors only considered a line to be a directive if the **#** appeared in column 1 on the line. Therefore **−Wtraditional** warns about directives that traditional C understands but ignores because the **#** does not appear as the first character on the line. It also suggests you hide directives like #pragma not understood by traditional C by indenting them. Some traditional implementations do not recognize #elif, so this option suggests avoiding it altogether.

* A function−like macro that appears without arguments.

* The unary plus operator.

* The **U** integer constant suffix, or the **F** or **L** floating−point constant suffixes. (Traditional C does support the **L** suffix on integer constants.) Note, these suffixes appear in macros defined in the system headers of most modern systems, e.g. the **_MIN/_MAX** macros in <limits.h>. Use of these macros in user code might normally lead to spurious warnings, however GCC's integrated preprocessor has enough context to avoid warning in these cases.

* A function declared external in one block and then used after the end of the block.

* A switch statement has an operand of type long.

* A non−static function declaration follows a static one. This construct is not accepted by some traditional C compilers.

* The ISO type of an integer constant has a different width or signedness from its traditional type. This warning is only issued if the base of the constant is ten. I.e. hexadecimal or octal values, which typically represent bit patterns, are not warned about.

* Usage of ISO string concatenation is detected.

* Initialization of automatic aggregates.

* Identifier conflicts with labels. Traditional C lacks a separate namespace for labels.

* Initialization of unions. If the initializer is zero, the warning is omitted. This is done under the assumption that the zero initializer in user code appears conditioned on e.g. __STDC__ to avoid missing initializer warnings and relies on default initialization to zero in the traditional C case.

* Conversions by prototypes between fixed/floating−point values and vice versa. The absence of these prototypes when compiling with traditional C causes serious problems. This is a subset of the possible conversion warnings; for the full set use **−Wtraditional−conversion**.

* Use of ISO C style function definitions. This warning intentionally is *not* issued for prototype declarations or variadic functions because these ISO C features appear in your code when using libiberty's traditional C compatibility macros, PARAMS and VPARAMS. This warning is also bypassed for nested functions because that feature is already a GCC extension and thus not relevant to traditional C compatibility.

−Wtraditional−conversion (C and Objective−C only)
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.

−Wdeclaration−after−statement (C and Objective-C only)
Warn when a declaration is found after a statement in a block. This construct, known from C++, was introduced with ISO C99 and is by default allowed in GCC. It is not supported by ISO C90.

This warning is upgraded to an error by **−pedantic−errors**.

−Wshadow
Warn whenever a local variable or type declaration shadows another variable, parameter, type, class member (in C++), or instance variable (in Objective−C) or whenever a built−in function is shadowed. Note that in C++, the compiler warns if a local variable shadows an explicit typedef, but not if it shadows a struct/class/enum. If this warning is enabled, it includes also all instances of local shadowing. This means that **−Wno−shadow=local** and **−Wno−shadow=compatible−local** are ignored when **−Wshadow** is used. Same as **−Wshadow=global**.

−Wno−shadow−ivar (Objective−C only)
Do not warn whenever a local variable shadows an instance variable in an Objective−C method.

−Wshadow=global
Warn for any shadowing. Same as **−Wshadow**.

−Wshadow=local
Warn when a local variable shadows another local variable or parameter.

−Wshadow=compatible−local
Warn when a local variable shadows another local variable or parameter whose type is compatible with that of the shadowing variable. In C++, type compatibility here means the type of the shadowing variable can be converted to that of the shadowed variable. The creation of this flag (in addition to **−Wshadow=local**) is based on the idea that when a local variable shadows another one of incompatible type, it is most likely intentional, not a bug or typo, as shown in the following example:

```
for (SomeIterator i = SomeObj.begin(); i != SomeObj.end(); ++i)
{
  for (int i = 0; i < N; ++i)
  {
    ...
  }
  ...
}
```

Since the two variable `i` in the example above have incompatible types, enabling only **−Wshadow=compatible−local** does not emit a warning. Because their types are incompatible, if a programmer accidentally uses one in place of the other, type checking is expected to catch that and emit an error or warning. Use of this flag instead of **−Wshadow=local** can possibly reduce the number of warnings triggered by intentional shadowing. Note that this also means that shadowing `const char *i` by `char *i` does not emit a warning.

This warning is also enabled by **−Wshadow=local**.

−Wlarger−than=*byte−size*
Warn whenever an object is defined whose size exceeds *byte−size*. **−Wlarger−than=PTRDIFF_MAX** is enabled by default. Warnings controlled by the option can be disabled either by specifying *byte−size* of **SIZE_MAX** or more or by **−Wno−larger−than**.

Also warn for calls to bounded functions such as `memchr` or `strnlen` that specify a bound greater than the largest possible object, which is **PTRDIFF_MAX** bytes by default. These warnings can only be disabled by **−Wno−larger−than**.

−Wno−larger−than
Disable **−Wlarger−than=** warnings. The option is equivalent to **−Wlarger−than=SIZE_MAX** or larger.

−Wframe−larger−than=*byte−size*
Warn if the size of a function frame exceeds *byte−size*. The computation done to determine the stack frame size is approximate and not conservative. The actual requirements may be somewhat greater than *byte−size* even if you do not get a warning. In addition, any space allocated via `alloca`,

variable−length arrays, or related constructs is not included by the compiler when determining whether or not to issue a warning. **−Wframe−larger−than=PTRDIFF_MAX** is enabled by default. Warnings controlled by the option can be disabled either by specifying *byte−size* of **SIZE_MAX** or more or by **−Wno−frame−larger−than**.

−Wno−frame−larger−than
Disable   **−Wframe−larger−than=**   warnings.   The   option   is   equivalent   to **−Wframe−larger−than=SIZE_MAX** or larger.

−Wfree−nonheap−object
Warn when attempting to deallocate an object that was either not allocated on the heap, or by using a pointer that was not returned from a prior call to the corresponding allocation function. For example, because the call to stpcpy returns a pointer to the terminating nul character and not to the beginning of the object, the call to free below is diagnosed.

```
void f (char *p)
{
  p = stpcpy (p, "abc");
  // ...
  free (p);   // warning
}
```

−Wfree−nonheap−object is included in **−Wall**.

−Wstack−usage=*byte−size*
Warn if the stack usage of a function might exceed *byte−size*. The computation done to determine the stack usage is conservative. Any space allocated via alloca, variable−length arrays, or related constructs is included by the compiler when determining whether or not to issue a warning.

The message is in keeping with the output of **−fstack−usage**.

*   If the stack usage is fully static but exceeds the specified amount, it's:

                    warning: stack usage is 1120 bytes

*   If the stack usage is (partly) dynamic but bounded, it's:

                    warning: stack usage might be 1648 bytes

*   If the stack usage is (partly) dynamic and not bounded, it's:

                    warning: stack usage might be unbounded

−Wstack−usage=PTRDIFF_MAX is enabled by default. Warnings controlled by the option can be disabled either by specifying *byte−size* of **SIZE_MAX** or more or by **−Wno−stack−usage**.

−Wno−stack−usage
Disable **−Wstack−usage=** warnings. The option is equivalent to **−Wstack−usage=SIZE_MAX** or larger.

−Wunsafe−loop−optimizations
Warn if the loop cannot be optimized because the compiler cannot assume anything on the bounds of the loop indices. With **−funsafe−loop−optimizations** warn if the compiler makes such assumptions.

−Wno−pedantic−ms−format (MinGW targets only)
When used in combination with **−Wformat** and **−pedantic** without GNU extensions, this option disables the warnings about non−ISO printf / scanf format width specifiers I32, I64, and I used on Windows targets, which depend on the MS runtime.

−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. In C++, warn also when an arithmetic operation involves NULL. This warning is also enabled by **−Wpedantic**.

This warning is upgraded to an error by **−pedantic−errors**.

−Wno−pointer−compare
>   Do not warn if a pointer is compared with a zero character constant. This usually means that the pointer was meant to be dereferenced. For example:

```
const char *p = foo ();
if (p == '\0')
  return 42;
```

>   Note that the code above is invalid in C++11.

>   This warning is enabled by default.

−Wno−tsan
>   Disable warnings about unsupported features in ThreadSanitizer.

>   ThreadSanitizer does not support `std::atomic_thread_fence` and can report false positives.

−Wtype−limits
>   Warn if a comparison is always true or always false due to the limited range of the data type, but do not warn for constant expressions. For example, warn if an unsigned variable is compared against zero with < or >=. This warning is also enabled by **−Wextra**.

−Wabsolute−value (C and Objective−C only)
>   Warn for calls to standard functions that compute the absolute value of an argument when a more appropriate standard function is available. For example, calling `abs(3.14)` triggers the warning because the appropriate function to call to compute the absolute value of a double argument is `fabs`. The option also triggers warnings when the argument in a call to such a function has an unsigned type. This warning can be suppressed with an explicit type cast and it is also enabled by **−Wextra**.

−Wcomment
−Wcomments
>   Warn whenever a comment−start sequence **/*** appears in a **/*** comment, or whenever a backslash−newline appears in a **//** comment. This warning is enabled by **−Wall**.

−Wtrigraphs
>   Warn if any trigraphs are encountered that might change the meaning of the program. Trigraphs within comments are not warned about, except those that would form escaped newlines.

>   This option is implied by **−Wall**. If **−Wall** is not given, this option is still enabled unless trigraphs are enabled. To get trigraph conversion without warnings, but get the other **−Wall** warnings, use **−trigraphs −Wall −Wno−trigraphs**.

−Wundef
>   Warn if an undefined identifier is evaluated in an `#if` directive. Such identifiers are replaced with zero.

−Wexpansion−to−defined
>   Warn whenever **defined** is encountered in the expansion of a macro (including the case where the macro is expanded by an **#if** directive). Such usage is not portable. This warning is also enabled by **−Wpedantic** and **−Wextra**.

−Wunused−macros
>   Warn about macros defined in the main file that are unused. A macro is *used* if it is expanded or tested for existence at least once. The preprocessor also warns if the macro has not been used at the time it is redefined or undefined.

>   Built−in macros, macros defined on the command line, and macros defined in include files are not warned about.

>   *Note:* If a macro is actually used, but only used in skipped conditional blocks, then the preprocessor reports it as unused. To avoid the warning in such a case, you might improve the scope of the macro's

definition by, for example, moving it into the first skipped block. Alternatively, you could provide a dummy use with something like:

```
#if defined the_macro_causing_the_warning
#endif
```

−Wno−endif−labels

Do not warn whenever an `#else` or an `#endif` are followed by text. This sometimes happens in older programs with code of the form

```
#if FOO
...
#else FOO
...
#endif FOO
```

The second and third `FOO` should be in comments. This warning is on by default.

−Wbad−function−cast (C and Objective−C only)

Warn when a function call is cast to a non−matching type. For example, warn if a call to a function returning an integer type is cast to a pointer type.

−Wc90−c99−compat (C and Objective−C only)

Warn about features not present in ISO C90, but present in ISO C99. For instance, warn about use of variable length arrays, `long long` type, `bool` type, compound literals, designated initializers, and so on. This option is independent of the standards mode. Warnings are disabled in the expression that follows `__extension__`.

−Wc99−c11−compat (C and Objective−C only)

Warn about features not present in ISO C99, but present in ISO C11. For instance, warn about use of anonymous structures and unions, `_Atomic` type qualifier, `_Thread_local` storage−class specifier, `_Alignas` specifier, `Alignof` operator, `_Generic` keyword, and so on. This option is independent of the standards mode. Warnings are disabled in the expression that follows `__extension__`.

−Wc11−c23−compat (C and Objective−C only)
−Wc11−c2x−compat (C and Objective−C only)

Warn about features not present in ISO C11, but present in ISO C23. For instance, warn about omitting the string in `_Static_assert`, use of **[[]]** syntax for attributes, use of decimal floating−point types, and so on. This option is independent of the standards mode. Warnings are disabled in the expression that follows `__extension__`. The name **−Wc11−c2x−compat** is deprecated.

When not compiling in C23 mode, these warnings are upgraded to errors by **−pedantic−errors**.

−Wc23−c2y−compat (C and Objective−C only)
−Wc23−c2y−compat (C and Objective−C only)

Warn about features not present in ISO C23, but present in ISO C2Y. For instance, warn about `_Generic` selecting with a type name instead of an expression. This option is independent of the standards mode. Warnings are disabled in the expression that follows `__extension__`.

When not compiling in C2Y mode, these warnings are upgraded to errors by **−pedantic−errors**.

−Wc++−compat (C and Objective−C only)

Warn about ISO C constructs that are outside of the common subset of ISO C and ISO C++, e.g. request for implicit conversion from `void *` to a pointer to non−`void` type.

−Wc++11−compat (C++ and Objective−C++ only)

Warn about C++ constructs whose meaning differs between ISO C++ 1998 and ISO C++ 2011, e.g., identifiers in ISO C++ 1998 that are keywords in ISO C++ 2011. This warning turns on **−Wnarrowing** and is enabled by **−Wall**.

−Wc++14−compat (C++ and Objective−C++ only)
>   Warn about C++ constructs whose meaning differs between ISO C++ 2011 and ISO C++ 2014. This warning is enabled by **−Wall**.

−Wc++17−compat (C++ and Objective−C++ only)
>   Warn about C++ constructs whose meaning differs between ISO C++ 2014 and ISO C++ 2017. This warning is enabled by **−Wall**.

−Wc++20−compat (C++ and Objective-C++ only)
>   Warn about C++ constructs whose meaning differs between ISO C++ 2017 and ISO C++ 2020. This warning is enabled by **−Wall**.

−Wno−c++11−extensions (C++ and Objective−C++ only)
>   Do not warn about C++11 constructs in code being compiled using an older C++ standard. Even without this option, some C++11 constructs will only be diagnosed if **−Wpedantic** is used.

−Wno−c++14−extensions (C++ and Objective−C++ only)
>   Do not warn about C++14 constructs in code being compiled using an older C++ standard. Even without this option, some C++14 constructs will only be diagnosed if **−Wpedantic** is used.

−Wno−c++17−extensions (C++ and Objective−C++ only)
>   Do not warn about C++17 constructs in code being compiled using an older C++ standard. Even without this option, some C++17 constructs will only be diagnosed if **−Wpedantic** is used.

−Wno−c++20−extensions (C++ and Objective−C++ only)
>   Do not warn about C++20 constructs in code being compiled using an older C++ standard. Even without this option, some C++20 constructs will only be diagnosed if **−Wpedantic** is used.

−Wno−c++23−extensions (C++ and Objective−C++ only)
>   Do not warn about C++23 constructs in code being compiled using an older C++ standard. Even without this option, some C++23 constructs will only be diagnosed if **−Wpedantic** is used.

−Wno−c++26−extensions (C++ and Objective−C++ only)
>   Do not warn about C++26 constructs in code being compiled using an older C++ standard. Even without this option, some C++26 constructs will only be diagnosed if **−Wpedantic** is used.

−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 *`.
>
>   Also warn when making a cast that introduces a type qualifier in an unsafe way. For example, casting `char **` to `const char **` is unsafe, as in this example:
>
> ```
> /* p is char ** value.  */
> const char **q = (const char **) p;
> /* Assignment of readonly string to const char * is OK.  */
> *q = "string";
> /* Now char** pointer points to read-only memory.  */
> **p = 'b';
> ```

−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.

−Wcast−align=strict
>   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 *` regardless of the target machine.

−Wcast−function−type
>   Warn when a function pointer is cast to an incompatible function pointer. In a cast involving function types with a variable argument list only the types of initial arguments that are provided are considered. Any parameter of pointer−type matches any other pointer−type. Any benign differences in integral

types are ignored, like `int` vs. `long` on ILP32 targets. Likewise type qualifiers are ignored. The function type `void (*) (void)` is special and matches everything, which can be used to suppress this warning. In a cast involving pointer to member types this warning warns whenever the type cast is changing the pointer to member type. This warning is enabled by **−Wextra**.

−Wcast−user−defined

Warn when a cast to reference type does not involve a user−defined conversion that the programmer might expect to be called.

```
struct A { operator const int&(); } a;
auto r = (int&)a; // warning
```

This warning is enabled by default.

−Wwrite−strings

When compiling C, give string constants the type `const  char[`*length*`]` so that copying the address of one into a non−`const char  *` pointer produces a warning. These warnings 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 is just a nuisance. This is why we did not make **−Wall** request these warnings.

When compiling C++, warn about the deprecated conversion from string literals to `char  *`. This warning is enabled by default for C++ programs.

This warning is upgraded to an error by **−pedantic−errors** in **C++11** mode or later.

−Wclobbered

Warn for variables that might be changed by `longjmp` or `vfork`. This warning is also enabled by **−Wextra**.

−Wno−complain−wrong−lang

By default, language front ends complain when a command−line option is valid, but not applicable to that front end. This may be disabled with **−Wno−complain−wrong−lang**, which is mostly useful when invoking a single compiler driver for multiple source files written in different languages, for example:

```
$ g++ −fno−rtti a.cc b.f90
```

The driver *g++* invokes the C++ front end to compile *a.cc* and the Fortran front end to compile *b.f90*. The latter front end diagnoses **f951: Warning: command−line option '−fno−rtti' is valid for C++/D/ObjC++ but not for Fortran**, which may be disabled with **−Wno−complain−wrong−lang**.

−Wcompare−distinct−pointer−types (C and Objective−C only)

Warn if pointers of distinct types are compared without a cast. This warning is enabled by default.

−Wconversion

Warn for implicit conversions that may alter a value. This includes conversions between real and integer, like `abs  (x)` when `x` is `double`; conversions between signed and unsigned, like `unsigned ui = −1`; and conversions to smaller types, like `sqrtf  (M_PI)`. Do not warn for explicit casts like `abs  ((int)  x)` and `ui = (unsigned)  −1`, or if the value is not changed by the conversion like in `abs  (2.0)`. Warnings about conversions between signed and unsigned integers can be disabled by using **−Wno−sign−conversion**.

For C++, also warn for confusing overload resolution for user−defined conversions; and conversions that never use a type conversion operator: conversions to `void`, the same type, a base class or a reference to them. Warnings about conversions between signed and unsigned integers are disabled by default in C++ unless **−Wsign−conversion** is explicitly enabled.

Warnings about conversion from arithmetic on a small type back to that type are only given with **−Warith−conversion**.

−Wdangling−else

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

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

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

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

> This warning is enabled by **−Wparentheses**.

−Wdangling−pointer

−Wdangling−pointer=*n*

> Warn about uses of pointers (or C++ references) to objects with automatic storage duration after their lifetime has ended. This includes local variables declared in nested blocks, compound literals and other unnamed temporary objects. In addition, warn about storing the address of such objects in escaped pointers. The warning is enabled at all optimization levels but may yield different results with optimization than without.

> **−Wdangling−pointer=1**

>> At level 1, the warning diagnoses only unconditional uses of dangling pointers.

> **−Wdangling−pointer=2**

>> At level 2, in addition to unconditional uses the warning also diagnoses conditional uses of dangling pointers.

> The short form **−Wdangling−pointer** is equivalent to **−Wdangling−pointer=2**, while **−Wno−dangling−pointer** and **−Wdangling−pointer=0** have the same effect of disabling the warnings. **−Wdangling−pointer=2** is included in **−Wall**.

> This example triggers the warning at level 1; the address of the unnamed temporary is unconditionally referenced outside of its scope.

```
char f (char c1, char c2, char c3)
{
  char *p;
  {
    p = (char[]) { c1, c2, c3 };
  }
  // warning: using dangling pointer 'p' to an unnamed temporary
  return *p;
}
```

In the following function the store of the address of the local variable x in the escaped pointer `*p` triggers the warning at level 1.

```
void g (int **p)
{
  int x = 7;
  // warning: storing the address of local variable 'x' in '*p'
  *p = &x;
}
```

In this example, the array *a* is out of scope when the pointer *s* is used. Since the code that sets s is conditional, the warning triggers at level 2.

```
extern void frob (const char *);
void h (char *s)
{
  if (!s)
    {
      char a[12] = "tmpname";
      s = a;
    }
  // warning: dangling pointer 's' to 'a' may be used
  frob (s);
}
```

−Wdate−time

Warn when macros `__TIME__`, `__DATE__` or `__TIMESTAMP__` are encountered as they might prevent bit−wise−identical reproducible compilations.

−Wempty−body

Warn if an empty body occurs in an if, else or do while statement. This warning is also enabled by **−Wextra**.

−Wno−endif−labels

Do not warn about stray tokens after #else and #endif.

−Wenum−compare

Warn about a comparison between values of different enumerated types. In C++ enumerated type mismatches in conditional expressions are also diagnosed and the warning is enabled by default. In C this warning is enabled by **−Wall**.

−Wenum−conversion

Warn when a value of enumerated type is implicitly converted to a different enumerated type. This warning is enabled by **−Wextra** in C.

−Wenum−int−mismatch (C and Objective−C only)

Warn about mismatches between an enumerated type and an integer type in declarations. For example:

```
enum E { l = -1, z = 0, g = 1 };
int foo(void);
enum E foo(void);
```

In C, an enumerated type is compatible with char, a signed integer type, or an unsigned integer type. However, since the choice of the underlying type of an enumerated type is implementation−defined, such mismatches may cause portability issues. In C++, such mismatches are an error. In C, this warning is enabled by **−Wall** and **−Wc++−compat**.

−Wjump−misses−init (C, Objective−C only)

Warn if a goto statement or a switch statement jumps forward across the initialization of a variable, or jumps backward to a label after the variable has been initialized. This only warns about variables that are initialized when they are declared. This warning is only supported for C and Objective−C; in C++ this sort of branch is an error in any case.

−Wjump−misses−init is included in **−Wc++−compat**. It can be disabled with the **−Wno−jump−misses−init** option.

−Wsign−compare

Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. In C++, this warning is also enabled by **−Wall**. In C, it is also enabled by **−Wextra**.

−Wsign−conversion

Warn for implicit conversions that may change the sign of an integer value, like assigning a signed integer expression to an unsigned integer variable. An explicit cast silences the warning. In C, this option is enabled also by **−Wconversion**.

−Wflex−array−member−not−at−end (C and C++ only)

Warn when a structure containing a C99 flexible array member as the last field is not at the end of another structure. This warning warns e.g. about

```
struct flex  { int length; char data[]; };
struct mid_flex { int m; struct flex flex_data; int n; };
```

−Wfloat−conversion

Warn for implicit conversions that reduce the precision of a real value. This includes conversions from real to integer, and from higher precision real to lower precision real values. This option is also enabled by **−Wconversion**.

−Wno−scalar−storage−order

Do not warn on suspicious constructs involving reverse scalar storage order.

−Wsizeof−array−div

Warn about divisions of two sizeof operators when the first one is applied to an array and the divisor does not equal the size of the array element. In such a case, the computation will not yield the number of elements in the array, which is likely what the user intended. This warning warns e.g. about

```
int fn ()
{
  int arr[10];
  return sizeof (arr) / sizeof (short);
}
```

This warning is enabled by **−Wall**.

−Wsizeof−pointer−div

Warn for suspicious divisions of two sizeof expressions that divide the pointer size by the element size, which is the usual way to compute the array size but won't work out correctly with pointers. This warning warns e.g. about sizeof (ptr) / sizeof (ptr[0]) if ptr is not an array, but a pointer. This warning is enabled by **−Wall**.

−Wsizeof−pointer−memaccess
Warn for suspicious length parameters to certain string and memory built−in functions if the argument uses `sizeof`. This warning triggers for example for `memset (ptr, 0, sizeof (ptr));` if `ptr` is not an array, but a pointer, and suggests a possible fix, or about `memcpy (&foo, ptr, sizeof (&foo));`. **−Wsizeof−pointer−memaccess** also warns about calls to bounded string copy functions like `strncat` or `strncpy` that specify as the bound a `sizeof` expression of the source array. For example, in the following function the call to `strncat` specifies the size of the source string as the bound. That is almost certainly a mistake and so the call is diagnosed.

```
void make_file (const char *name)
{
  char path[PATH_MAX];
  strncpy (path, name, sizeof path - 1);
  strncat (path, ".text", sizeof ".text");
  ...
}
```

The **−Wsizeof−pointer−memaccess** option is enabled by **−Wall**.

−Wno−sizeof−array−argument
Do not warn when the `sizeof` operator is applied to a parameter that is declared as an array in a function definition. This warning is enabled by default for C and C++ programs.

−Wmemset−elt−size
Warn for suspicious calls to the `memset` built−in function, if the first argument references an array, and the third argument is a number equal to the number of elements, but not equal to the size of the array in memory. This indicates that the user has omitted a multiplication by the element size. This warning is enabled by **−Wall**.

−Wmemset−transposed−args
Warn for suspicious calls to the `memset` built−in function where the second argument is not zero and the third argument is zero. For example, the call `memset (buf, sizeof buf, 0)` is diagnosed because `memset (buf, 0, sizeof buf)` was meant instead. The diagnostic is only emitted if the third argument is a literal zero. Otherwise, if it is an expression that is folded to zero, or a cast of zero to some type, it is far less likely that the arguments have been mistakenly transposed and no warning is emitted. This warning is enabled by **−Wall**.

−Waddress
Warn about suspicious uses of address expressions. These include comparing the address of a function or a declared object to the null pointer constant such as in

```
void f (void);
void g (void)
{
  if (!f)   // warning: expression evaluates to false
    abort ();
}
```

comparisons of a pointer to a string literal, such as in

```
void f (const char *x)
{
  if (x == "abc")   // warning: expression evaluates to false
    puts ("equal");
}
```

and tests of the results of pointer addition or subtraction for equality to null, such as in

```
void f (const int *p, int i)
{
  return p + i == NULL;
}
```

Such uses typically indicate a programmer error: the address of most functions and objects necessarily evaluates to true (the exception are weak symbols), so their use in a conditional might indicate missing parentheses in a function call or a missing dereference in an array expression. The subset of the warning for object pointers can be suppressed by casting the pointer operand to an integer type such as intptr_t or uintptr_t. Comparisons against string literals result in unspecified behavior and are not portable, and suggest the intent was to call strcmp. The warning is suppressed if the suspicious expression is the result of macro expansion. **−Waddress** warning is enabled by **−Wall**.

−Wno−address−of−packed−member
Do not warn when the address of packed member of struct or union is taken, which usually results in an unaligned pointer value. This is enabled by default.

−Wlogical−op
Warn about suspicious uses of logical operators in expressions. This includes using logical operators in contexts where a bit−wise operator is likely to be expected. Also warns when the operands of a logical operator are the same:

```
extern int a;
if (a < 0 && a < 0) { ... }
```

−Wlogical−not−parentheses
Warn about logical not used on the left hand side operand of a comparison. This option does not warn if the right operand is considered to be a boolean expression. Its purpose is to detect suspicious code like the following:

```
int a;
...
if (!a > 1) { ... }
```

It is possible to suppress the warning by wrapping the LHS into parentheses:

```
if ((!a) > 1) { ... }
```

This warning is enabled by **−Wall**.

−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.)

−Wno−aggressive−loop−optimizations
Do not warn if the compiler detects undefined behavior in a loop with a constant number of iterations. **−Waggressive−loop−optimizations** is enabled by default.

−Wno−attributes
Do not warn if an unexpected __attribute__ is used, such as unrecognized attributes, function attributes applied to variables, etc. This does not stop errors for incorrect use of supported attributes.

Warnings about ill−formed uses of standard attributes are upgraded to errors by **−pedantic−errors**.

Additionally, using **−Wno−attributes=**, it is possible to suppress warnings about unknown scoped attributes (in C++11 and C23). For example, **−Wno−attributes=vendor::attr** disables warning about the following declaration:

```
[[vendor::attr]] void f();
```

It is also possible to disable warning about all attributes in a namespace using **−Wno−attributes=vendor::** which prevents warning about both of these declarations:

```
[[vendor::safe]] void f();
[[vendor::unsafe]] void f2();
```

Note that **−Wno−attributes=** does not imply **−Wno−attributes**.

−Wno−builtin−declaration−mismatch

Warn if a built−in function is declared with an incompatible signature or as a non−function, or when a built−in function declared with a type that does not include a prototype is called with arguments whose promoted types do not match those expected by the function. When **−Wextra** is specified, also warn when a built−in function that takes arguments is declared without a prototype. The **−Wbuiltin−declaration−mismatch** warning is enabled by default. To avoid the warning include the appropriate header to bring the prototypes of built−in functions into scope.

For example, the call to memset below is diagnosed by the warning because the function expects a value of type size_t as its argument but the type of 32 is int. With **−Wextra**, the declaration of the function is diagnosed as well.

```
extern void* memset ();
void f (void *d)
{
  memset (d, '\0', 32);
}
```

−Wno−builtin−macro−redefined

Do not warn if certain built−in macros are redefined. This suppresses warnings for redefinition of __TIMESTAMP__, __TIME__, __DATE__, __FILE__, and __BASE_FILE__.

−Wfree−labels (C and Objective−C only)

Warn if a label is applied to a non−statement, or occurs at the end of a compound statement. Such labels are allowed by C23 and later dialects of C, and are available as a GCC extension in all other dialects.

This warning is also enabled by **−Wc11−c23−compat**. It is turned into an error if building for a C version before C23 by **−pedantic−errors**.

−Wheader−guard

Warn if a valid preprocessor header multiple inclusion guard has a #define directive right after #ifndef or #if !defined directive for the multiple inclusion guard, which defines a different macro from the guard macro with a similar name, the actual multiple inclusion guard macro isn't defined at the corresponding #ifndef directive at the end of the header, and the #define directive defines an object−like macro with empty definition. In such case, it often is just a misspelled guard name, either in the #ifndef or #if !defined directive or in the subsequent #define directive. This warning is enabled by **−Wall**.

−Wstrict−prototypes (C and Objective−C only)

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

−Wold−style−declaration (C and Objective−C only)

Warn for obsolescent usages, according to the C Standard, in a declaration. For example, warn if storage−class specifiers like static are not the first things in a declaration. This warning is also enabled by **−Wextra**.

−Wold−style−definition (C and Objective−C only)

Warn if an old−style function definition is used. A warning is given even if there is a previous prototype. A definition using **()** is not considered an old−style definition in C23 mode, because it is equivalent to **(void)** in that case, but is considered an old−style definition for older standards.

−Wdeprecated−non−prototype (C and Objective−C only)
   Warn if a function declared with an empty parameter list **()** is called with one or more arguments, or if
   a function definition with one or more parameters is encountered after such a declaration. Both cases
   are errors in C23 and later dialects of C.

   This warning is also enabled by **−Wc11−c23−compat**.

−Wmissing−parameter−name (C and Objective−C only)
   Warn if a function definition omits a parameter name, specifying only its type. This can be used to
   document that a parameter is unused in the definition. It is part of C23 and later dialects of C, and
   available as a GCC extension in all other dialects.

   This warning is also enabled by **−Wc11−c23−compat**. It is turned into an error if building for a C
   version before C23 by **−pedantic−errors**.

−Wmissing−parameter−type (C and Objective−C only)
   A function parameter is declared without a type specifier in K&R−style functions:

         void foo(bar) { }

   This warning is also enabled by **−Wextra**.

−Wno−declaration−missing−parameter−type (C and Objective−C only)
   Do not warn if a function declaration contains a parameter name without a type. Such function
   declarations do not provide a function prototype and prevent most type checking in function calls.

   This warning is enabled by default. In C99 and later dialects of C, it is treated as an error. The error
   can be downgraded to a warning using **−fpermissive** (along with certain other errors), or for this error
   alone, with **−Wno−error=declaration−missing−parameter−type**.

   This warning is upgraded to an error by **−pedantic−errors**.

−Wmissing−prototypes (C and Objective−C only)
   Warn if a global function is defined without a previous prototype declaration. This warning is issued
   even if the definition itself provides a prototype. Use this option to detect global functions that do not
   have a matching prototype declaration in a header file. This option is not valid for C++ because all
   function declarations provide prototypes and a non−matching declaration declares an overload rather
   than conflict with an earlier declaration. Use **−Wmissing−declarations** to detect missing declarations
   in C++.

−Wmissing−variable−declarations (C and Objective−C only)
   Warn if a global variable is defined without a previous declaration. Use this option to detect global
   variables that do not have a matching extern declaration in a header file.

−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.
   In C, no warnings are issued for functions with previous non−prototype declarations; use
   −Wmissing−prototypes to detect missing prototypes. In C++, no warnings are issued for function
   templates, or for inline functions, or for functions in anonymous namespaces.

−Wmissing−field−initializers
   Warn if a structure's initializer has some fields missing. For example, the following code causes such a
   warning, because x.h is implicitly zero:

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

   In C this option does not warn about designated initializers, so the following modification does not
   trigger a warning:

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

In C this option does not warn about the universal zero initializer **{ 0 }**:

```
struct s { int f, g, h; };
struct s x = { 0 };
```

Likewise, in C++ this option does not warn about the empty { } initializer, for example:

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

This warning is included in **−Wextra**. To get other **−Wextra** warnings without this one, use **−Wextra −Wno−missing−field−initializers**.

−Wno−missing−requires
By default, the compiler warns about a concept−id appearing as a C++20 simple−requirement:

```
bool satisfied = requires { C<T> };
```

Here **satisfied** will be true if **C<T>** is a valid expression, which it is for all T. Presumably the user meant to write

```
bool satisfied = requires { requires C<T> };
```

so **satisfied** is only true if concept **C** is satisfied for type **T**.

This warning can be disabled with **−Wno−missing−requires**.

−Wno−missing−template−keyword
The member access tokens ., −> and :: must be followed by the template keyword if the parent object is dependent and the member being named is a template.

```
template <class X>
void DoStuff (X x)
{
  x.template DoSomeOtherStuff<X>(); // Good.
  x.DoMoreStuff<X>(); // Warning, x is dependent.
}
```

In rare cases it is possible to get false positives. To silence this, wrap the expression in parentheses. For example, the following is treated as a template, even where m and N are integers:

```
void NotATemplate (my_class t)
{
  int N = 5;

  bool test = t.m < N > (0); // Treated as a template.
  test = (t.m < N) > (0); // Same meaning, but not treated as a template.
}
```

This warning can be disabled with **−Wno−missing−template−keyword**.

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

−Wnormalized=[none|id|nfc|nfkc]
In ISO C and ISO C++, two identifiers are different if they are different sequences of characters. However, sometimes when characters outside the basic ASCII character set are used, you can have two different character sequences that look the same. To avoid confusion, the ISO 10646 standard sets out some *normalization rules* which when applied ensure that two sequences that look the same are turned into the same sequence. GCC can warn you if you are using identifiers that have not been normalized;

this option controls that warning.

There are four levels of warning supported by GCC. The default is **−Wnormalized=nfc**, which warns about any identifier that is not in the ISO 10646 "C" normalized form, *NFC*. NFC is the recommended form for most uses. It is equivalent to **−Wnormalized**.

Unfortunately, there are some characters allowed in identifiers by ISO C and ISO C++ that, when turned into NFC, are not allowed in identifiers. That is, there's no way to use these symbols in portable ISO C or C++ and have all your identifiers in NFC. **−Wnormalized=id** suppresses the warning for these characters. It is hoped that future versions of the standards involved will correct this, which is why this option is not the default.

You can switch the warning off for all characters by writing **−Wnormalized=none** or **−Wno−normalized**. You should only do this if you are using some other normalization scheme (like "D"), because otherwise you can easily create bugs that are literally impossible to see.

Some characters in ISO 10646 have distinct meanings but look identical in some fonts or display methodologies, especially once formatting has been applied. For instance \u207F, "SUPERSCRIPT LATIN SMALL LETTER N", displays just like a regular n that has been placed in a superscript. ISO 10646 defines the *NFKC* normalization scheme to convert all these into a standard form as well, and GCC warns if your code is not in NFKC if you use **−Wnormalized=nfkc**. This warning is comparable to warning about every identifier that contains the letter O because it might be confused with the digit 0, and so is not the default, but may be useful as a local coding convention if the programming environment cannot be fixed to display these characters distinctly.

−Wno−attribute−warning
Do not warn about usage of functions declared with warning attribute. By default, this warning is enabled. **−Wno−attribute−warning** can be used to disable the warning or **−Wno−error=attribute−warning** can be used to disable the error when compiled with **−Werror** flag.

−Wno−deprecated
Do not warn about usage of deprecated features.

In C++, explicitly specifying **−Wdeprecated** also enables warnings about some features that are deprecated in later language standards, specifically **−Wcomma−subscript**, **−Wvolatile**, **−Wdeprecated−enum−float−conversion**, **−Wdeprecated−enum−enum−conversion**, **−Wdeprecated−literal−operator**, and **−Wdeprecated−variadic−comma−omission**.

−Wno−deprecated−declarations
Do not warn about uses of functions, variables, and types marked as deprecated by using the deprecated attribute.

−Wno−overflow
Do not warn about compile−time overflow in constant expressions.

−Wno−odr
Warn about One Definition Rule violations during link−time optimization. Enabled by default.

−Wopenacc−parallelism
Warn about potentially suboptimal choices related to OpenACC parallelism.

−Wno−openmp
Warn about suspicious OpenMP code.

−Wopenmp−simd
Warn if the vectorizer cost model overrides the OpenMP simd directive set by user. The **−fsimd−cost−model=unlimited** option can be used to relax the cost model.

−Woverride−init (C and Objective−C only)
Warn if an initialized field without side effects is overridden when using designated initializers.

This warning is included in **−Wextra**. To get other **−Wextra** warnings without this one, use **−Wextra −Wno−override−init**.

−Wno−override−init−side−effects (C and Objective−C only)
　　　Do not warn if an initialized field with side effects is overridden when using designated initializers.
　　　This warning is enabled by default.

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

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

−Wnopacked−bitfield−compat
　　　The 4.1, 4.2 and 4.3 series of GCC ignore the `packed` attribute on bit−fields of type `char`. This was
　　　fixed in GCC 4.4 but the change can lead to differences in the structure layout. GCC informs you
　　　when the offset of such a field has changed in GCC 4.4. For example there is no longer a 4−bit
　　　padding between field `a` and `b` in this structure:

```
struct foo
{
  char a:4;
  char b:8;
} __attribute__ ((packed));
```

　　　This warning is enabled by default. Use **−Wno−packed−bitfield−compat** to disable this warning.

−Wpacked−not−aligned (C, C++, Objective−C and Objective−C++ only)
　　　Warn if a structure field with explicitly specified alignment in a packed struct or union is misaligned.
　　　For example, a warning will be issued on `struct S`, like, `warning: alignment 1 of
　　　'struct S' is less than 8`, in this code:

```
struct __attribute__ ((aligned (8))) S8 { char a[8]; };
struct __attribute__ ((packed)) S {
  struct S8 s8;
};
```

　　　This warning is enabled by **−Wall**.

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

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

−Wrestrict
　　　Warn when an object referenced by a `restrict`−qualified parameter (or, in C++, a
　　　`__restrict`−qualified parameter) is aliased by another argument, or when copies between such
　　　objects overlap. For example, the call to the `strcpy` function below attempts to truncate the string
　　　by replacing its initial characters with the last four. However, because the call writes the terminating
　　　NUL into `a[4]`, the copies overlap and the call is diagnosed.

```
                    void foo (void)
                    {
                      char a[] = "abcd1234";
                      strcpy (a, a + 4);
                      ...
                    }
```

The **−Wrestrict** option detects some instances of simple overlap even without optimization but works best at **−O2** and above. It is included in **−Wall**.

−Wnested−externs (C and Objective−C only)
Warn if an `extern` declaration is encountered within a function.

−Winline
Warn if a function that is declared as inline cannot be inlined. Even with this option, the compiler does not warn about failures to inline functions declared in system headers.

The compiler uses a variety of heuristics to determine whether or not to inline a function. For example, the compiler takes into account the size of the function being inlined and the amount of inlining that has already been done in the current function. Therefore, seemingly insignificant changes in the source program can cause the warnings produced by **−Winline** to appear or disappear.

−Winterference−size
Warn about use of C++17 `std::hardware_destructive_interference_size` without specifying its value with **−−param destructive−interference−size**. Also warn about questionable values for that option.

This variable is intended to be used for controlling class layout, to avoid false sharing in concurrent code:

```
            struct independent_fields {
              alignas(std::hardware_destructive_interference_size)
                std::atomic<int> one;
              alignas(std::hardware_destructive_interference_size)
                std::atomic<int> two;
            };
```

Here **one** and **two** are intended to be far enough apart that stores to one won't require accesses to the other to reload the cache line.

By default, **−−param destructive−interference−size** and **−−param constructive−interference−size** are set based on the current **−mtune** option, typically to the L1 cache line size for the particular target CPU, sometimes to a range if tuning for a generic target. So all translation units that depend on ABI compatibility for the use of these variables must be compiled with the same **−mtune** (or **−mcpu**).

If ABI stability is important, such as if the use is in a header for a library, you should probably not use the hardware interference size variables at all. Alternatively, you can force a particular value with **−−param**.

If you are confident that your use of the variable does not affect ABI outside a single build of your project, you can turn off the warning with **−Wno−interference−size**.

−Wint−in−bool−context
Warn for suspicious use of integer values where boolean values are expected, such as conditional expressions (?:) using non−boolean integer constants in boolean context, like `if (a <= b ? 2 : 3)`. Or left shifting of signed integers in boolean context, like `for (a = 0; 1 << a; a++);`. Likewise for all kinds of multiplications regardless of the data type. This warning is enabled by **−Wall**.

−Wno−int−to−pointer−cast
> Suppress warnings from casts to pointer type of an integer of a different size. In C++, casting to a pointer type of smaller size is an error. **Wint−to−pointer−cast** is enabled by default.

−Wno−pointer−to−int−cast (C and Objective−C only)
> Suppress warnings from casts from a pointer to an integer type of a different size.

−Winvalid−pch
> Warn if a precompiled header is found in the search path but cannot be used.

−Winvalid−utf8
> Warn if an invalid UTF−8 character is found. This warning is on by default for C++23 if **−finput−charset=UTF−8** is used and turned into error with **−pedantic−errors**.

−Wno−unicode
> Don't diagnose invalid forms of delimited or named escape sequences which are treated as separate tokens. **Wunicode** is enabled by default.

−Wlong−long
> Warn if `long long` type is used. This is enabled by either **−Wpedantic** or **−Wtraditional** in ISO C90 and C++98 modes. To inhibit the warning messages, use **−Wno−long−long**.
>
> This warning is upgraded to an error by **−pedantic−errors**.

−Wvariadic−macros
> Warn if variadic macros are used in ISO C90 mode, or if the GNU alternate syntax is used in ISO C99 mode. This is enabled by either **−Wpedantic** or **−Wtraditional**. To inhibit the warning messages, use **−Wno−variadic−macros**.

−Wno−varargs
> Do not warn upon questionable usage of the macros used to handle variable arguments like `va_start`. These warnings are enabled by default.

−Wvector−operation−performance
> Warn if vector operation is not implemented via SIMD capabilities of the architecture. Mainly useful for the performance tuning. Vector operation can be implemented `piecewise`, which means that the scalar operation is performed on every vector element; `in parallel`, which means that the vector operation is implemented using scalars of wider type, which normally is more performance efficient; and `as a single scalar`, which means that vector fits into a scalar type.

−Wvla
> Warn if a variable−length array is used in the code. **−Wno−vla** prevents the **−Wpedantic** warning of the variable−length array.
>
> This warning is upgraded to an error by **−pedantic−errors**.

−Wvla−larger−than=*byte−size*
> If this option is used, the compiler warns for declarations of variable−length arrays whose size is either unbounded, or bounded by an argument that allows the array size to exceed *byte−size* bytes. This is similar to how **−Walloca−larger−than=***byte−size* works, but with variable−length arrays.
>
> Note that GCC may optimize small variable−length arrays of a known value into plain arrays, so this warning may not get triggered for such arrays.
>
> **−Wvla−larger−than=PTRDIFF_MAX** is enabled by default but is typically only effective when **−ftree−vrp** is active (default for **−O2** and above).
>
> See also **−Walloca−larger−than=***byte−size*.

−Wno−vla−larger−than
> Disable **−Wvla−larger−than=** warnings. The option is equivalent to **−Wvla−larger−than=SIZE_MAX** or larger.

−Wvla−parameter

Warn about redeclarations of functions involving arguments of Variable Length Array types of inconsistent kinds or forms, and enable the detection of out−of−bounds accesses to such parameters by warnings such as **−Warray−bounds**.

If the first function declaration uses the VLA form the bound specified in the array is assumed to be the minimum number of elements expected to be provided in calls to the function and the maximum number of elements accessed by it. Failing to provide arguments of sufficient size or accessing more than the maximum number of elements may be diagnosed.

For example, the warning triggers for the following redeclarations because the first one allows an array of any size to be passed to f while the second one specifies that the array argument must have at least n elements. In addition, calling f with the associated VLA bound parameter in excess of the actual VLA bound triggers a warning as well.

```
void f (int n, int[n]);
// warning: argument 2 previously declared as a VLA
void f (int, int[]);

void g (int n)
{
    if (n > 4)
      return;
    int a[n];
    // warning: access to a by f may be out of bounds
    f (sizeof a, a);
  ...
}
```

−Wvla−parameter is included in **−Wall**. The **−Warray−parameter** option triggers warnings for similar problems involving ordinary array arguments.

−Wvolatile−register−var

Warn if a register variable is declared volatile. The volatile modifier does not inhibit all optimizations that may eliminate reads and/or writes to register variables. This warning is enabled by **−Wall**.

−Wno−xor−used−as−pow (C, C++, Objective−C and Objective−C++ only)

Disable warnings about uses of ˆ, the exclusive or operator, where it appears the code meant exponentiation. Specifically, the warning occurs when the left−hand side is the decimal constant 2 or 10 and the right−hand side is also a decimal constant.

In C and C++, ˆ means exclusive or, whereas in some other languages (e.g. TeX and some versions of BASIC) it means exponentiation.

This warning can be silenced by converting one of the operands to hexadecimal as well as by compiling with **−Wno−xor−used−as−pow**.

−Wdisabled−optimization

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

−Wpointer−sign (C and Objective−C only)

Warn for pointer argument passing or assignment with different signedness. This option is only supported for C and Objective−C. It is implied by **−Wall** and by **−Wpedantic**, which can be disabled with **−Wno−pointer−sign**.

This warning is upgraded to an error by **−pedantic−errors**.

−Wstack−protector

This option is only active when **−fstack−protector** is active. It warns about functions that are not protected against stack smashing.

−Woverlength−strings

Warn about string constants that are longer than the "minimum maximum" length specified in the C standard. Modern compilers generally allow string constants that are much longer than the standard's minimum limit, but very portable programs should avoid using longer strings.

The limit applies *after* string constant concatenation, and does not count the trailing NUL. In C90, the limit was 509 characters; in C99, it was raised to 4095. C++98 does not specify a normative minimum maximum, so we do not diagnose overlength strings in C++.

This option is implied by **−Wpedantic**, and can be disabled with **−Wno−overlength−strings**.

−Wunsuffixed−float−constants (C and Objective−C only)

Issue a warning for any floating constant that does not have a suffix. When used together with **−Wsystem−headers** it warns about such constants in system header files. This can be useful when preparing code to use with the FLOAT_CONST_DECIMAL64 pragma from the decimal floating−point extension to C99.

−Wno−lto−type−mismatch

During the link−time optimization, do not warn about type mismatches in global declarations from different compilation units. Requires **−flto** to be enabled. Enabled by default.

−Wno−designated−init (C and Objective−C only)

Suppress warnings when a positional initializer is used to initialize a structure that has been marked with the designated_init attribute.

Options That Control Static Analysis

−fanalyzer

This option enables an static analysis of program flow which looks for "interesting" interprocedural paths through the code, and issues warnings for problems found on them.

This analysis is much more expensive than other GCC warnings.

In technical terms, it performs coverage−guided symbolic execution of the code being compiled. It is neither sound nor complete: it can have false positives and false negatives. It is a bug−finding tool, rather than a tool for proving program correctness.

The analyzer is only suitable for use on C code in this release.

Enabling this option effectively enables the following warnings:

−Wanalyzer−allocation−size	**−Wanalyzer−deref−before−check**	**−Wanalyzer−double−fclose**
−Wanalyzer−double−free		**−Wanalyzer−exposure−through−output−file**
−Wanalyzer−exposure−through−uninit−copy		**−Wanalyzer−fd−access−mode−mismatch**
−Wanalyzer−fd−double−close	**−Wanalyzer−fd−leak**	**−Wanalyzer−fd−phase−mismatch**
−Wanalyzer−fd−type−mismatch		**−Wanalyzer−fd−use−after−close**
−Wanalyzer−fd−use−without−check	**−Wanalyzer−file−leak**	**−Wanalyzer−free−of−non−heap**
−Wanalyzer−imprecise−fp−arithmetic		**−Wanalyzer−infinite−loop**
−Wanalyzer−infinite−recursion	**−Wanalyzer−jump−through−null**	**−Wanalyzer−malloc−leak**
−Wanalyzer−mismatching−deallocation		**−Wanalyzer−null−argument**
−Wanalyzer−null−dereference	**−Wanalyzer−out−of−bounds**	**−Wanalyzer−overlapping−buffers**
−Wanalyzer−possible−null−argument		**−Wanalyzer−possible−null−dereference**
−Wanalyzer−putenv−of−auto−var		**−Wanalyzer−shift−count−negative**
−Wanalyzer−shift−count−overflow		**−Wanalyzer−stale−setjmp−buffer**
−Wanalyzer−tainted−allocation−size		**−Wanalyzer−tainted−array−index**
−Wanalyzer−tainted−assertion	**−Wanalyzer−tainted−divisor**	**−Wanalyzer−tainted−offset**
−Wanalyzer−tainted−size		**−Wanalyzer−undefined−behavior−ptrdiff**
−Wanalyzer−undefined−behavior−strtok		**−Wanalyzer−unsafe−call−within−signal−handler**

> **−Wanalyzer−use−after−free**   **−Wanalyzer−use−of−pointer−in−stale−stack−frame**
> **−Wanalyzer−use−of−uninitialized−value**   **−Wanalyzer−va−arg−type−mismatch**
> **−Wanalyzer−va−list−exhausted**   **−Wanalyzer−va−list−leak**
> **−Wanalyzer−va−list−use−after−va−end**   **−Wanalyzer−write−to−const**
> **−Wanalyzer−write−to−string−literal**

This option is only available if GCC was configured with analyzer support enabled.

−Wanalyzer−symbol−too−complex

If **−fanalyzer** is enabled, the analyzer uses various heuristics to attempt to track the state of memory, but these can be defeated by sufficiently complicated code.

By default, the analysis silently stops tracking values of expressions if they exceed the threshold defined by **−−param analyzer−max−svalue−depth=***value*, and falls back to an imprecise representation for such expressions. The **−Wanalyzer−symbol−too−complex** option warns if this occurs.

−Wanalyzer−too−complex

If **−fanalyzer** is enabled, the analyzer uses various heuristics to attempt to explore the control flow and data flow in the program, but these can be defeated by sufficiently complicated code.

By default, the analysis silently stops if the code is too complicated for the analyzer to fully explore and it reaches an internal limit. The **−Wanalyzer−too−complex** option warns if this occurs.

−Wno−analyzer−allocation−size

This   warning   requires   **−fanalyzer**,   which   enables   it;   to   disable   it,   use **−Wno−analyzer−allocation−size**.

This diagnostic warns for paths through the code in which a pointer to a buffer is assigned to point at a buffer with a size that is not a multiple of `sizeof (*pointer)`.

See   CWE−131:   Incorrect   Calculation   of   Buffer   Size (`https://cwe.mitre.org/data/definitions/131.html`).

−Wno−analyzer−deref−before−check

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−deref−before−check** to disable it.

This diagnostic warns for paths through the code in which a pointer is checked for `NULL` *after* it has already been dereferenced, suggesting that the pointer could have been NULL. Such cases suggest that the check for NULL is either redundant, or that it needs to be moved to before the pointer is dereferenced.

This diagnostic also considers values passed to a function argument marked with `__attribute__((nonnull))` as requiring a non−NULL value, and thus will complain if such values are checked for `NULL` after returning from such a function call.

This diagnostic is unlikely to be reported when any level of optimization is enabled, as GCC's optimization logic will typically consider such checks for NULL as being redundant, and optimize them away before the analyzer "sees" them. Hence optimization should be disabled when attempting to trigger this diagnostic.

−Wno−analyzer−double−fclose

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−double−fclose** to disable it.

This diagnostic warns for paths through the code in which a `FILE *` can have `fclose` called on it more than once.

See   CWE−1341:   Multiple   Releases   of   Same   Resource   or   Handle (`https://cwe.mitre.org/data/definitions/1341.html`).

−Wno−analyzer−double−free

    This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−double−free** to disable it.

    This diagnostic warns for paths through the code in which a pointer can have a deallocator called on it more than once, either `free`, or a deallocator referenced by attribute `malloc`.

    See CWE−415: Double Free (`https://cwe.mitre.org/data/definitions/415.html`).

−Wno−analyzer−exposure−through−output−file

    This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−exposure−through−output−file** to disable it.

    This diagnostic warns for paths through the code in which a security−sensitive value is written to an output file (such as writing a password to a log file).

    See CWE−532: Information Exposure Through Log Files (`https://cwe.mitre.org/data/definitions/532.html`).

−Wanalyzer−exposure−through−uninit−copy

    This warning requires both **−fanalyzer** and the use of a plugin to specify a function that copies across a "trust boundary". Use **−Wno−analyzer−exposure−through−uninit−copy** to disable it.

    This diagnostic warns for "infoleaks" − paths through the code in which uninitialized values are copied across a security boundary (such as code within an OS kernel that copies a partially−initialized struct on the stack to user space).

    See CWE−200: Exposure of Sensitive Information to an Unauthorized Actor (`https://cwe.mitre.org/data/definitions/200.html`).

−Wno−analyzer−fd−access−mode−mismatch

    This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−fd−access−mode−mismatch** to disable it.

    This diagnostic warns for paths through code in which a `read` on a write−only file descriptor is attempted, or vice versa.

    This diagnostic also warns for code paths in a which a function with attribute `fd_arg_read (N)` is called with a file descriptor opened with `O_WRONLY` at referenced argument N or a function with attribute `fd_arg_write (N)` is called with a file descriptor opened with `O_RDONLY` at referenced argument *N*.

−Wno−analyzer−fd−double−close

    This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−fd−double−close** to disable it.

    This diagnostic warns for paths through code in which a file descriptor can be closed more than once.

    See CWE−1341: Multiple Releases of Same Resource or Handle (`https://cwe.mitre.org/data/definitions/1341.html`).

−Wno−analyzer−fd−leak

    This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−fd−leak** to disable it.

    This diagnostic warns for paths through code in which an open file descriptor is leaked.

    See CWE−775: Missing Release of File Descriptor or Handle after Effective Lifetime (`https://cwe.mitre.org/data/definitions/775.html`).

−Wno−analyzer−fd−phase−mismatch

    This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−fd−phase−mismatch** to disable it.

    This diagnostic warns for paths through code in which an operation is attempted in the wrong phase of a file descriptor's lifetime. For example, it will warn on attempts to call `accept` on a stream socket that has not yet had `listen` successfully called on it.

See CWE−666: Operation on Resource in Wrong Phase of Lifetime (https://cwe.mitre.org/data/definitions/666.html).

−Wno−analyzer−fd−type−mismatch

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−fd−type−mismatch** to disable it.

This diagnostic warns for paths through code in which an operation is attempted on the wrong type of file descriptor. For example, it will warn on attempts to use socket operations on a file descriptor obtained via open, or when attempting to use a stream socket operation on a datagram socket.

−Wno−analyzer−fd−use−after−close

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−fd−use−after−close** to disable it.

This diagnostic warns for paths through code in which a read or write is called on a closed file descriptor.

This diagnostic also warns for paths through code in which a function with attribute fd_arg (N) or fd_arg_read (N) or fd_arg_write (N) is called with a closed file descriptor at referenced argument N.

−Wno−analyzer−fd−use−without−check

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−fd−use−without−check** to disable it.

This diagnostic warns for paths through code in which a file descriptor is used without being checked for validity.

This diagnostic also warns for paths through code in which a function with attribute fd_arg (N) or fd_arg_read (N) or fd_arg_write (N) is called with a file descriptor, at referenced argument N, without being checked for validity.

−Wno−analyzer−file−leak

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−file−leak** to disable it.

This diagnostic warns for paths through the code in which a <stdio.h> FILE * stream object is leaked.

See CWE−775: Missing Release of File Descriptor or Handle after Effective Lifetime (https://cwe.mitre.org/data/definitions/775.html).

−Wno−analyzer−free−of−non−heap

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−free−of−non−heap** to disable it.

This diagnostic warns for paths through the code in which free is called on a non−heap pointer (e.g. an on−stack buffer, or a global).

See CWE−590: Free of Memory not on the Heap (https://cwe.mitre.org/data/definitions/590.html).

−Wno−analyzer−imprecise−fp−arithmetic

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−imprecise−fp−arithmetic** to disable it.

This diagnostic warns for paths through the code in which floating−point arithmetic is used in locations where precise computation is needed. This diagnostic only warns on use of floating−point operands inside the calculation of an allocation size at the moment.

−Wno−analyzer−infinite−loop

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−infinite−loop** to disable it.

This diagnostics warns for paths through the code which appear to lead to an infinite loop.

Specifically, the analyzer will issue this warning when it "sees" a loop in which:

* no externally−visible work could be being done within the loop

* there is no way to escape from the loop

* the analyzer is sufficiently confident about the program state throughout the loop to know that the above are true

One way for this warning to be emitted is when there is an execution path through a loop for which taking the path on one iteration implies that the same path will be taken on all subsequent iterations.

For example, consider:

```
while (1)
  {
    char opcode = *cpu_state.pc;
    switch (opcode)
     {
     case OPCODE_FOO:
       handle_opcode_foo (&cpu_state);
       break;
     case OPCODE_BAR:
       handle_opcode_bar (&cpu_state);
       break;
     }
  }
```

The analyzer will complain for the above case because if `opcode` ever matches none of the cases, the `switch` will follow the implicit `default` case, making the body of the loop be a "no−op" with `cpu_state.pc` unchanged, and thus using the same value of `opcode` on all subseqent iterations, leading to an infinite loop.

See CWE−835: Loop with Unreachable Exit Condition ('Infinite Loop') (https://cwe.mitre.org/data/definitions/835.html).

−Wno−analyzer−infinite−recursion
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−infinite−recursion** to disable it.

This diagnostics warns for paths through the code which appear to lead to infinite recursion.

Specifically, when the analyzer "sees" a recursive call, it will compare the state of memory at the entry to the new frame with that at the entry to the previous frame of that function on the stack. The warning is issued if nothing in memory appears to be changing; any changes observed to parameters or globals are assumed to lead to termination of the recursion and thus suppress the warning.

This diagnostic is likely to miss cases of infinite recursion that are convered to iteration by the optimizer before the analyzer "sees" them. Hence optimization should be disabled when attempting to trigger this diagnostic.

Compare with **−Winfinite−recursion**, which provides a similar diagnostic, but is implemented in a different way.

See CWE−674: Uncontrolled Recursion (https://cwe.mitre.org/data/definitions/674.html).

−Wno−analyzer−jump−through−null
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−jump−through−null** to disable it.

This diagnostic warns for paths through the code in which a `NULL` function pointer is called.

−Wno−analyzer−malloc−leak

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−malloc−leak** to disable it.
>
> This diagnostic warns for paths through the code in which a pointer allocated via an allocator is leaked: either `malloc`, or a function marked with attribute `malloc`.
>
> See CWE−401: Missing Release of Memory after Effective Lifetime (`https://cwe.mitre.org/data/definitions/401.html`).

−Wno−analyzer−mismatching−deallocation

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−mismatching−deallocation** to disable it.
>
> This diagnostic warns for paths through the code in which the wrong deallocation function is called on a pointer value, based on which function was used to allocate the pointer value. The diagnostic will warn about mismatches between `free`, scalar `delete` and vector `delete[]`, and those marked as allocator/deallocator pairs using attribute `malloc`.
>
> See CWE−762: Mismatched Memory Management Routines (`https://cwe.mitre.org/data/definitions/762.html`).

−Wno−analyzer−out−of−bounds

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−out−of−bounds** to disable it.
>
> This diagnostic warns for paths through the code in which a buffer is definitely read or written out−of−bounds. The diagnostic applies for cases where the analyzer is able to determine a constant offset and for accesses past the end of a buffer, also a constant capacity. Further, the diagnostic does limited checking for accesses past the end when the offset as well as the capacity is symbolic.
>
> See CWE−119: Improper Restriction of Operations within the Bounds of a Memory Buffer (`https://cwe.mitre.org/data/definitions/119.html`).
>
> For cases where the analyzer is able, it will emit a text art diagram visualizing the spatial relationship between the memory region that the analyzer predicts would be accessed, versus the range of memory that is valid to access: whether they overlap, are touching, are close or far apart; which one is before or after in memory, the relative sizes involved, the direction of the access (read vs write), and, in some cases, the values of data involved. This diagram can be suppressed using **−fdiagnostics−text−art−charset=none**.

−Wno−analyzer−overlapping−buffers

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−overlapping−buffers** to disable it.
>
> This diagnostic warns for paths through the code in which overlapping buffers are passed to an API for which the behavior on such buffers is undefined.
>
> Specifically, the diagnostic occurs on calls to the following functions
>
> ```
> *<memcpy>
> *<strcat>
> *<strcpy>
> ```
>
> for cases where the buffers are known to overlap.

−Wno−analyzer−possible−null−argument

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−possible−null−argument** to disable it.
>
> This diagnostic warns for paths through the code in which a possibly−NULL value is passed to a function argument marked with `__attribute__((nonnull))` as requiring a non−NULL value.
>
> See CWE−690: Unchecked Return Value to NULL Pointer Dereference

(`https://cwe.mitre.org/data/definitions/690.html`).

−Wno−analyzer−possible−null−dereference

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−possible−null−dereference** to disable it.

This diagnostic warns for paths through the code in which a possibly−NULL value is dereferenced.

See CWE−690: Unchecked Return Value to NULL Pointer Dereference (`https://cwe.mitre.org/data/definitions/690.html`).

−Wno−analyzer−null−argument

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−null−argument** to disable it.

This diagnostic warns for paths through the code in which a value known to be NULL is passed to a function argument marked with `__attribute__((nonnull))` as requiring a non−NULL value.

See CWE−476: NULL Pointer Dereference (`https://cwe.mitre.org/data/definitions/476.html`).

−Wno−analyzer−null−dereference

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−null−dereference** to disable it.

This diagnostic warns for paths through the code in which a value known to be NULL is dereferenced.

See CWE−476: NULL Pointer Dereference (`https://cwe.mitre.org/data/definitions/476.html`).

−Wno−analyzer−putenv−of−auto−var

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−putenv−of−auto−var** to disable it.

This diagnostic warns for paths through the code in which a call to `putenv` is passed a pointer to an automatic variable or an on−stack buffer.

See POS34−C. Do not call **putenv()** with a pointer to an automatic variable as the argument (`https://wiki.sei.cmu.edu/confluence/x/6NYxBQ`).

−Wno−analyzer−shift−count−negative

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−shift−count−negative** to disable it.

This diagnostic warns for paths through the code in which a shift is attempted with a negative count. It is analogous to the **−Wshift−count−negative** diagnostic implemented in the C/C++ front ends, but is implemented based on analyzing interprocedural paths, rather than merely parsing the syntax tree. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

−Wno−analyzer−shift−count−overflow

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−shift−count−overflow** to disable it.

This diagnostic warns for paths through the code in which a shift is attempted with a count greater than or equal to the precision of the operand's type. It is analogous to the **−Wshift−count−overflow** diagnostic implemented in the C/C++ front ends, but is implemented based on analyzing interprocedural paths, rather than merely parsing the syntax tree. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

−Wno−analyzer−stale−setjmp−buffer

This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−stale−setjmp−buffer** to disable it.

This diagnostic warns for paths through the code in which `longjmp` is called to rewind to a `jmp_buf` relating to a `setjmp` call in a function that has returned.

When `setjmp` is called on a `jmp_buf` to record a rewind location, it records the stack frame. The stack frame becomes invalid when the function containing the `setjmp` call returns. Attempting to rewind to it via `longjmp` would reference a stack frame that no longer exists, and likely lead to a crash (or worse).

−Wno−analyzer−tainted−allocation−size
This warning requires **−fanalyzer** which enables it; use **−Wno−analyzer−tainted−allocation−size** to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the size of an allocation without being sanitized, so that an attacker could inject an excessively large allocation and potentially cause a denial of service attack.

See CWE−789: Memory Allocation with Excessive Size Value (https://cwe.mitre.org/data/definitions/789.html).

−Wno−analyzer−tainted−assertion
This warning requires **−fanalyzer** which enables it; use **−Wno−analyzer−tainted−assertion** to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as part of a condition without being first sanitized, and that condition guards a call to a function marked with attribute `noreturn` (such as the function `__builtin_unreachable`). Such functions typically indicate abnormal termination of the program, such as for assertion failure handlers. For example:

```
assert (some_tainted_value < SOME_LIMIT);
```

In such cases:

*   when assertion−checking is enabled: an attacker could trigger a denial of service by injecting an assertion failure

*   when assertion−checking is disabled, such as by defining `NDEBUG`, an attacker could inject data that subverts the process, since it presumably violates a precondition that is being assumed by the code.

Note that when assertion−checking is disabled, the assertions are typically removed by the preprocessor before the analyzer has a chance to "see" them, so this diagnostic can only generate warnings on builds in which assertion−checking is enabled.

For the purpose of this warning, any function marked with attribute `noreturn` is considered as a possible assertion failure handler, including `__builtin_unreachable`. Note that these functions are sometimes removed by the optimizer before the analyzer "sees" them. Hence optimization should be disabled when attempting to trigger this diagnostic.

See CWE−617: Reachable Assertion (https://cwe.mitre.org/data/definitions/617.html).

The warning can also report problematic constructions such as

```
switch (some_tainted_value) {
case 0:
  /* [...etc; various valid cases omitted...] */
  break;

default:
  __builtin_unreachable (); /* BUG: attacker can trigger this  */
}
```

despite the above not being an assertion failure, strictly speaking.

−Wno−analyzer−tainted−array−index
> This warning requires **−fanalyzer** which enables it; use **−Wno−analyzer−tainted−array−index** to disable it.
>
> This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the index of an array access without being sanitized, so that an attacker could inject an out−of−bounds access.
>
> See CWE−129: Improper Validation of Array Index (https://cwe.mitre.org/data/definitions/129.html).

−Wno−analyzer−tainted−divisor
> This warning requires **−fanalyzer** which enables it; use **−Wno−analyzer−tainted−divisor** to disable it.
>
> This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the divisor in a division or modulus operation without being sanitized, so that an attacker could inject a division−by−zero.
>
> See CWE−369: Divide By Zero (https://cwe.mitre.org/data/definitions/369.html).

−Wno−analyzer−tainted−offset
> This warning requires **−fanalyzer** which enables it; use **−Wno−analyzer−tainted−offset** to disable it.
>
> This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as a pointer offset without being sanitized, so that an attacker could inject an out−of−bounds access.
>
> See CWE−823: Use of Out−of−range Pointer Offset (https://cwe.mitre.org/data/definitions/823.html).

−Wno−analyzer−tainted−size
> This warning requires **−fanalyzer** which enables it; use **−Wno−analyzer−tainted−size** to disable it.
>
> This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the size of an operation such as `memset` without being sanitized, so that an attacker could inject an out−of−bounds access.
>
> See CWE−129: Improper Validation of Array Index (https://cwe.mitre.org/data/definitions/129.html).

−Wno−analyzer−undefined−behavior−ptrdiff
> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−undefined−behavior−ptrdiff** to disable it.
>
> This diagnostic warns for paths through the code in which a pointer subtraction occurs where the pointers refer to different chunks of memory. Such code relies on undefined behavior, as pointer subtraction is only defined for cases where both pointers point to within (or just after) the same array.
>
> See CWE−469: Use of Pointer Subtraction to Determine Size (https://cwe.mitre.org/data/definitions/469.html).

−Wno−analyzer−undefined−behavior−strtok
> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−undefined−behavior−strtok** to disable it.
>
> This diagnostic warns for paths through the code in which a call is made to `strtok` with undefined behavior.
>
> Specifically, passing NULL as the first parameter for the initial call to `strtok` within a process has undefined behavior.

−Wno−analyzer−unsafe−call−within−signal−handler
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−unsafe−call−within−signal−handler** to disable it.

This diagnostic warns for paths through the code in which a function known to be async−signal−unsafe (such as `fprintf`) is called from a signal handler.

See CWE−479: Signal Handler Use of a Non−reentrant Function (`https://cwe.mitre.org/data/definitions/479.html`).

−Wno−analyzer−use−after−free
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−use−after−free** to disable it.

This diagnostic warns for paths through the code in which a pointer is used after a deallocator is called on it: either `free`, or a deallocator referenced by attribute `malloc`.

See CWE−416: Use After Free (`https://cwe.mitre.org/data/definitions/416.html`).

−Wno−analyzer−use−of−pointer−in−stale−stack−frame
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−use−of−pointer−in−stale−stack−frame** to disable it.

This diagnostic warns for paths through the code in which a pointer is dereferenced that points to a variable in a stale stack frame.

−Wno−analyzer−va−arg−type−mismatch
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−va−arg−type−mismatch** to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects an attempt to use `va_arg` to extract a value passed to a variadic call, but uses a type that does not match that of the expression passed to the call.

See CWE−686: Function Call With Incorrect Argument Type (`https://cwe.mitre.org/data/definitions/686.html`).

−Wno−analyzer−va−list−exhausted
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−va−list−exhausted** to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects an attempt to use `va_arg` to access the next value passed to a variadic call, but all of the values in the `va_list` have already been consumed.

See CWE−685: Function Call With Incorrect Number of Arguments (`https://cwe.mitre.org/data/definitions/685.html`).

−Wno−analyzer−va−list−leak
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−va−list−leak** to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects that `va_start` or `va_copy` has been called on a `va_list` without a corresponding call to `va_end`.

−Wno−analyzer−va−list−use−after−va−end
This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−va−list−use−after−va−end** to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects an attempt to use a `va_list` after `va_end` has been called on it. `va_list`.

−Wno−analyzer−write−to−const

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−write−to−const** to disable it.

> This diagnostic warns for paths through the code in which the analyzer detects an attempt to write through a pointer to a `const` object. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

−Wno−analyzer−write−to−string−literal

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−write−to−string−literal** to disable it.

> This diagnostic warns for paths through the code in which the analyzer detects an attempt to write through a pointer to a string literal. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

−Wno−analyzer−use−of−uninitialized−value

> This warning requires **−fanalyzer**, which enables it; use **−Wno−analyzer−use−of−uninitialized−value** to disable it.

> This diagnostic warns for paths through the code in which an uninitialized value is used.

> See CWE−457: Use of Uninitialized Variable (`https://cwe.mitre.org/data/definitions/457.html`).

The analyzer has hardcoded knowledge about the behavior of the following memory−management functions:

```
*<alloca>
*<The built−in functions __builtin_alloc,>
    __builtin_alloc_with_align, @item __builtin_calloc, __builtin_free,
    __builtin_malloc,        __builtin_memcpy,        __builtin_memcpy_chk,
    __builtin_memset,        __builtin_memset_chk,        __builtin_realloc,
    __builtin_stack_restore, and __builtin_stack_save
```

```
*<calloc>
*<free>
*<malloc>
*<memset>
*<operator delete>
*<operator delete []>
*<operator new>
*<operator new []>
*<realloc>
*<strdup>
*<strndup>
```

of the following functions for working with file descriptors:

```
*<open>
*<close>
*<creat>
*<dup, dup2 and dup3>
*<isatty>
*<pipe, and pipe2>
*<read>
*<write>
*<socket, bind, listen, accept, and connect>
```

of the following functions for working with `<stdio.h>` streams:

*<The built−in functions `__builtin_fprintf`,>
   `__builtin_fprintf_unlocked`, `__builtin_fputc`, `__builtin_fputc_unlocked`,
   `__builtin_fputs`,   `__builtin_fputs_unlocked`,   `__builtin_fwrite`,
   `__builtin_fwrite_unlocked`,   `__builtin_printf`,
   `__builtin_printf_unlocked`,   `__builtin_putc`,   `__builtin_putchar`,
   `__builtin_putchar_unlocked`, `__builtin_putc_unlocked`, `__builtin_puts`,
   `__builtin_puts_unlocked`, `__builtin_vfprintf`, and `__builtin_vprintf`

*<`fopen`>
*<`fclose`>
*<`ferror`>
*<`fgets`>
*<`fgets_unlocked`>
*<`fileno`>
*<`fread`>
*<`getc`>
*<`getchar`>
*<`fprintf`>
*<`printf`>
*<`fwrite`>

and of the following functions:

*<The built−in functions `__builtin_expect`,>
   `__builtin_expect_with_probability`, `__builtin_strchr`, `__builtin_strcpy`,
   `__builtin_strcpy_chk`,   `__builtin_strlen`,   `__builtin_va_copy`,   and
   `__builtin_va_start`

*<The GNU extensions `error` and `error_at_line`>
*<`getpass`>
*<`longjmp`>
*<`putenv`>
*<`setjmp`>
*<`siglongjmp`>
*<`signal`>
*<`sigsetjmp`>
*<`strcat`>
*<`strchr`>
*<`strlen`>

In addition, various functions with an `__analyzer_` prefix have special meaning to the analyzer, described in the GCC Internals manual.

Pertinent parameters for controlling the exploration are:

*<**−−param analyzer−bb−explosion−factor=***value*>
*<**−−param analyzer−max−enodes−per−program−point=***value*>
*<**−−param analyzer−max−recursion−depth=***value*>
*<**−−param analyzer−min−snodes−for−call−summary=***value*>

The following options control the analyzer.

−fanalyzer−call−summaries
   Simplify interprocedural analysis by computing the effect of certain calls, rather than exploring all paths through the function from callsite to each possible return.

   If enabled, call summaries are only used for functions with more than one call site, and that are sufficiently complicated (as per **−−param analyzer−min−snodes−for−call−summary=***value*).

−fanalyzer−checker=*name*
> Restrict the analyzer to run just the named checker, and enable it.

−fanalyzer−debug−text−art−headings
> This option is intended for analyzer developers. If enabled, the analyzer will add extra annotations to any diagrams it generates.

−fno−analyzer−feasibility
> This option is intended for analyzer developers.

> By default the analyzer verifies that there is a feasible control flow path for each diagnostic it emits: that the conditions that hold are not mutually exclusive. Diagnostics for which no feasible path can be found are rejected. This filtering can be suppressed with **−fno−analyzer−feasibility**, for debugging issues in this code.

−fanalyzer−fine−grained
> This option is intended for analyzer developers.

> Internally the analyzer builds an "exploded graph" that combines control flow graphs with data flow information.

> By default, an edge in this graph can contain the effects of a run of multiple statements within a basic block. With **−fanalyzer−fine−grained**, each statement gets its own edge.

−fanalyzer−show−duplicate−count
> This option is intended for analyzer developers: if multiple diagnostics have been detected as being duplicates of each other, it emits a note when reporting the best diagnostic, giving the number of additional diagnostics that were suppressed by the deduplication logic.

−fanalyzer−show−events−in−system−headers
> By default the analyzer emits simplified diagnostics paths by hiding events fully located within a system header. With **−fanalyzer−show−events−in−system−headers** such events are no longer suppressed.

−fno−analyzer−state−merge
> This option is intended for analyzer developers.

> By default the analyzer attempts to simplify analysis by merging sufficiently similar states at each program point as it builds its "exploded graph". With **−fno−analyzer−state−merge** this merging can be suppressed, for debugging state−handling issues.

−fno−analyzer−state−purge
> This option is intended for analyzer developers.

> By default the analyzer attempts to simplify analysis by purging aspects of state at a program point that appear to no longer be relevant e.g. the values of locals that aren't accessed later in the function and which aren't relevant to leak analysis.

> With **−fno−analyzer−state−purge** this purging of state can be suppressed, for debugging state−handling issues.

−fno−analyzer−suppress−followups
> This option is intended for analyzer developers.

> By default the analyzer will stop exploring an execution path after encountering certain diagnostics, in order to avoid potentially issuing a cascade of follow−up diagnostics.

> The diagnostics that terminate analysis along a path are:

> *<**−Wanalyzer−null−argument**>
> *<**−Wanalyzer−null−dereference**>
> *<**−Wanalyzer−use−after−free**>

    *<−Wanalyzer−use−of−pointer−in−stale−stack−frame>
    *<−Wanalyzer−use−of−uninitialized−value>

With **−fno−analyzer−suppress−followups** the analyzer will continue to explore such paths even after such diagnostics, which may be helpful for debugging issues in the analyzer, or for microbenchmarks for detecting undefined behavior.

−fanalyzer−transitivity
> This option enables transitivity of constraints within the analyzer.

−fno−analyzer−undo−inlining
> This option is intended for analyzer developers.

> **−fanalyzer** runs relatively late compared to other code analysis tools, and some optimizations have already been applied to the code. In particular function inlining may have occurred, leading to the interprocedural execution paths emitted by the analyzer containing function frames that don't correspond to those in the original source code.

> By default the analyzer attempts to reconstruct the original function frames, and to emit events showing the inlined calls.

> With **−fno−analyzer−undo−inlining** this attempt to reconstruct the original frame information can be disabled, which may be of help when debugging issues in the analyzer.

−fanalyzer−verbose−edges
> This option is intended for analyzer developers. It enables more verbose, lower−level detail in the descriptions of control flow within diagnostic paths.

−fanalyzer−verbose−state−changes
> This option is intended for analyzer developers. It enables more verbose, lower−level detail in the descriptions of events relating to state machines within diagnostic paths.

−fanalyzer−verbosity=_level_
> This option controls the complexity of the control flow paths that are emitted for analyzer diagnostics.

> The _level_ can be one of:

> **0**   At this level, interprocedural call and return events are displayed, along with the most pertinent state−change events relating to a diagnostic. For example, for a double−`free` diagnostic, both calls to `free` will be shown.

> **1**   As per the previous level, but also show events for the entry to each function.

> **2**   As per the previous level, but also show events relating to control flow that are significant to triggering the issue (e.g. "true path taken" at a conditional).

>   This level is the default.

> **3**   As per the previous level, but show all control flow events, not just significant ones.

> **4**   This level is intended for analyzer developers; it adds various other events intended for debugging the analyzer.

−fdump−analyzer
> Dump internal details about what the analyzer is doing to _file.analyzer.txt_. **−fdump−analyzer−stderr** overrides this option.

−fdump−analyzer−stderr
> Dump internal details about what the analyzer is doing to stderr. This option overrides **−fdump−analyzer**.

−fdump−analyzer−callgraph
> Dump a representation of the call graph suitable for viewing with GraphViz to _file.callgraph.dot_.

−fdump−analyzer−exploded−graph

Dump a representation of the "exploded graph" suitable for viewing with GraphViz to *file.eg.dot*. Nodes are color−coded based on state−machine states to emphasize state changes.

−fdump−analyzer−exploded−nodes

Emit diagnostics showing where nodes in the "exploded graph" are in relation to the program source.

−fdump−analyzer−exploded−nodes−2

Dump a textual representation of the "exploded graph" to *file.eg.txt*.

−fdump−analyzer−exploded−nodes−3

Dump a textual representation of the "exploded graph" to one dump file per node, to *file.eg−id.txt*. This is typically a large number of dump files.

−fdump−analyzer−exploded−paths

Dump a textual representation of the "exploded path" for each diagnostic to *file.idx.kind.epath.txt*.

−fdump−analyzer−feasibility

Dump internal details about the analyzer's search for feasible paths. The details are written in a form suitable for viewing with GraphViz to filenames of the form *file.*.fg.dot*, *file.*.tg.dot*, and *file.*.fpath.txt*.

−fdump−analyzer−infinite−loop

Dump internal details about the analyzer's search for infinite loops. The details are written in a form suitable for viewing with GraphViz to filenames of the form *file.*.infinite−loop.dot*.

−fdump−analyzer−json

Dump a compressed JSON representation of analyzer internals to *file.analyzer.json.gz*. The precise format is subject to change.

−fdump−analyzer−state−purge

As per **−fdump−analyzer−supergraph**, dump a representation of the "supergraph" suitable for viewing with GraphViz, but annotate the graph with information on what state will be purged at each node. The graph is written to *file.state−purge.dot*.

−fdump−analyzer−supergraph

Dump representations of the "supergraph" suitable for viewing with GraphViz to *file.supergraph.dot* and to *file.supergraph−eg.dot*. These show all of the control flow graphs in the program, with interprocedural edges for calls and returns. The second dump contains annotations showing nodes in the "exploded graph" and diagnostics associated with them.

−fdump−analyzer−untracked

Emit custom warnings with internal details intended for analyzer developers.

Options for Debugging Your Program

To tell GCC to emit extra information for use by a debugger, in almost all cases you need only to add **−g** to your other options. Some debug formats can co−exist (like DWARF with CTF) when each of them is enabled explicitly by adding the respective command line option to your other options.

GCC allows you to use **−g** with **−O**. The shortcuts taken by optimized code may occasionally be surprising: 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 are already at hand; some statements may execute in different places because they have been moved out of loops. Nevertheless it is possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

If you are not using some other optimization option, consider using **−Og** with **−g**. With no **−O** option at all, some compiler passes that collect information useful for debugging do not run at all, so that **−Og** may result in a better debugging experience.

−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 probably makes other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use **−gvms** (see below).

−ggdb

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

−gdwarf
−gdwarf−*version*

Produce debugging information in DWARF format (if that is supported). The value of *version* may be either 2, 3, 4 or 5; the default version for most targets is 5 (with the exception of VxWorks, TPF and Darwin / macOS, which default to version 2, and AIX, which defaults to version 4).

Note that with DWARF Version 2, some ports require and always use some non−conflicting DWARF 3 extensions in the unwind tables.

Version 4 may require GDB 7.0 and **−fvar−tracking−assignments** for maximum benefit. Version 5 requires GDB 8.0 or higher.

GCC no longer supports DWARF Version 1, which is substantially different than Version 2 and later. For historical reasons, some other DWARF−related options such as **−fno−dwarf2−cfi−asm**) retain a reference to DWARF Version 2 in their names, but apply to all currently−supported versions of DWARF.

−gbtf

Request BTF debug information. BTF is the default debugging format for the eBPF target. On other targets, like x86, BTF debug information can be generated along with DWARF debug information when both of the debug formats are enabled explicitly via their respective command line options.

−gprune−btf
−gno−prune−btf

Prune BTF information before emission. When pruning, only type information for types used by global variables and file−scope functions will be emitted. If compiling for the BPF target with BPF CO−RE enabled, type information will also be emitted for types used in BPF CO−RE relocations. In addition, struct and union types which are only referred to via pointers from members of other struct or union types shall be pruned and replaced with BTF_KIND_FWD, as though those types were only present in the input as forward declarations.

This option substantially reduces the size of produced BTF information, but at significant loss in the amount of detailed type information. It is primarily useful when compiling for the BPF target, to minimize the size of the resulting object, and to eliminate BTF information which is not immediately relevant to the BPF program loading process.

This option is enabled by default for the BPF target when generating BTF information.

−gctf
−gctf*level*

Request CTF debug information and use level to specify how much CTF debug information should be produced. If **−gctf** is specified without a value for level, the default level of CTF debug information is 2.

CTF debug information can be generated along with DWARF debug information when both of the debug formats are enabled explicitly via their respective command line options.

Level 0 produces no CTF debug information at all. Thus, **−gctf0** negates **−gctf**.

Level 1 produces CTF information for tracebacks only. This includes callsite information, but does not include type information.

Level 2 produces type information for entities (functions, data objects etc.) at file−scope or global−scope only.

−gvms
Produce debugging information in Alpha/VMS debug format (if that is supported). This is the format used by DEBUG on Alpha/VMS systems.

−gcodeview
Produce debugging information in CodeView debug format (if that is supported). This is the format used by Microsoft Visual C++ on Windows.

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

Level 0 produces no debug information at all. Thus, **−g0** negates **−g**.

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, and line number tables, but no information about local variables.

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

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

−gdwarf does not accept a concatenated debug level, to avoid confusion with **−gdwarf−***level*. Instead use an additional **−g***level* option to change the debug level for DWARF.

−fno−eliminate−unused−debug−symbols
By default, no debug information is produced for symbols that are not actually used. Use this option if you want debug information for all symbols.

−femit−class−debug−always
Instead of emitting debugging information for a C++ class in only one object file, emit it in all object files using the class. This option should be used only with debuggers that are unable to handle the way GCC normally emits debugging information for classes because using this option increases the size of debugging information by as much as a factor of two.

−fno−merge−debug−strings
Direct the linker to not merge together strings in the debugging information that are identical in different object files. Merging is not supported by all assemblers or linkers. Merging decreases the size of the debug information in the output file at the cost of increasing link processing time. Merging is enabled by default.

−fdebug−prefix−map=*old***=***new*
When compiling files residing in directory *old*, record debugging information describing them as if the files resided in directory *new* instead. This can be used to replace a build−time path with an install−time path in the debug info. It can also be used to change an absolute path to a relative path by using . for *new*. This can give more reproducible builds, which are location independent, but may require an extra command to tell GDB where to find the source files. See also **−ffile−prefix−map** and **−fcanon−prefix−map**.

−fvar−tracking
Run variable tracking pass. It computes where variables are stored at each position in code. Better debugging information is then generated (if the debugging information format supports this information).

It is enabled by default when compiling with optimization (**−Os**, **−O**, **−O2**, ...), debugging information

(**−g**) and the debug info format supports it.

−fvar−tracking−assignments

Annotate assignments to user variables early in the compilation and attempt to carry the annotations over throughout the compilation all the way to the end, in an attempt to improve debug information while optimizing. Use of **−gdwarf−4** is recommended along with it.

It can be enabled even if var−tracking is disabled, in which case annotations are created and maintained, but discarded at the end. By default, this flag is enabled together with **−fvar−tracking**, except when selective scheduling is enabled.

−gsplit−dwarf

If DWARF debugging information is enabled, separate as much debugging information as possible into a separate output file with the extension *.dwo*. This option allows the build system to avoid linking files with debug information. To be useful, this option requires a debugger capable of reading *.dwo* files.

−gdwarf32
−gdwarf64

If DWARF debugging information is enabled, the **−gdwarf32** selects the 32−bit DWARF format and the **−gdwarf64** selects the 64−bit DWARF format. The default is target specific, on most targets it is **−gdwarf32** though. The 32−bit DWARF format is smaller, but can't support more than 2GiB of debug information in any of the DWARF debug information sections. The 64−bit DWARF format allows larger debug information and might not be well supported by all consumers yet.

−gdescribe−dies

Add description attributes to some DWARF DIEs that have no name attribute, such as artificial variables, external references and call site parameter DIEs.

−gpubnames

Generate DWARF `.debug_pubnames` and `.debug_pubtypes` sections.

−ggnu−pubnames

Generate `.debug_pubnames` and `.debug_pubtypes` sections in a format suitable for conversion into a GDB index. This option is only useful with a linker that can produce GDB index version 7.

−fdebug−types−section

When using DWARF Version 4 or higher, type DIEs can be put into their own `.debug_types` section instead of making them part of the `.debug_info` section. It is more efficient to put them in a separate comdat section since the linker can then remove duplicates. But not all DWARF consumers support `.debug_types` sections yet and on some objects `.debug_types` produces larger instead of smaller debugging information.

−grecord−gcc−switches
−gno−record−gcc−switches

This switch causes the command−line options used to invoke the compiler that may affect code generation to be appended to the DW_AT_producer attribute in DWARF debugging information. The options are concatenated with spaces separating them from each other and from the compiler version. It is enabled by default. See also **−frecord−gcc−switches** for another way of storing compiler options into the object file.

−gstrict−dwarf

Disallow using extensions of later DWARF standard version than selected with **−gdwarf−***version*. On most targets using non−conflicting DWARF extensions from later standard versions is allowed.

−gno−strict−dwarf

Allow using extensions of later DWARF standard version than selected with **−gdwarf−***version*.

−gas−loc−support

Inform the compiler that the assembler supports `.loc` directives. It may then use them for the assembler to generate DWARF2+ line number tables.

This is generally desirable, because assembler−generated line−number tables are a lot more compact than those the compiler can generate itself.

This option will be enabled by default if, at GCC configure time, the assembler was found to support such directives.

−gno−as−loc−support

Force GCC to generate DWARF2+ line number tables internally, if DWARF2+ line number tables are to be generated.

−gas−locview−support

Inform the compiler that the assembler supports `view` assignment and reset assertion checking in `.loc` directives.

This option will be enabled by default if, at GCC configure time, the assembler was found to support them.

−gno−as−locview−support

Force GCC to assign view numbers internally, if **−gvariable−location−views** are explicitly requested.

−gcolumn−info
−gno−column−info

Emit location column information into DWARF debugging information, rather than just file and line. This option is enabled by default.

−gstatement−frontiers
−gno−statement−frontiers

This option causes GCC to create markers in the internal representation at the beginning of statements, and to keep them roughly in place throughout compilation, using them to guide the output of `is_stmt` markers in the line number table. This is enabled by default when compiling with optimization (**−Os**, **−O1**, **−O2**, ...), and outputting DWARF 2 debug information at the normal level.

−gvariable−location−views
−gvariable−location−views=incompat5
−gno−variable−location−views

Augment variable location lists with progressive view numbers implied from the line number table. This enables debug information consumers to inspect state at certain points of the program, even if no instructions associated with the corresponding source locations are present at that point. If the assembler lacks support for view numbers in line number tables, this will cause the compiler to emit the line number table, which generally makes them somewhat less compact. The augmented line number tables and location lists are fully backward−compatible, so they can be consumed by debug information consumers that are not aware of these augmentations, but they won't derive any benefit from them either.

This is enabled by default when outputting DWARF 2 debug information at the normal level, as long as there is assembler support, **−fvar−tracking−assignments** is enabled and **−gstrict−dwarf** is not. When assembler support is not available, this may still be enabled, but it will force GCC to output internal line number tables, and if **−ginternal−reset−location−views** is not enabled, that will most certainly lead to silently mismatching location views.

There is a proposed representation for view numbers that is not backward compatible with the location list format introduced in DWARF 5, that can be enabled with **−gvariable−location−views=incompat5**. This option may be removed in the future, is only provided as a reference implementation of the proposed representation. Debug information consumers are not expected to support this extended format, and they would be rendered unable to decode location lists using it.

−ginternal−reset−location−views
−gno−internal−reset−location−views

Attempt to determine location views that can be omitted from location view lists. This requires the compiler to have very accurate insn length estimates, which isn't always the case, and it may cause

incorrect view lists to be generated silently when using an assembler that does not support location view lists. The GNU assembler will flag any such error as a `view number mismatch`. This is only enabled on ports that define a reliable estimation function.

−ginline−points
−gno−inline−points
>   Generate extended debug information for inlined functions. Location view tracking markers are inserted at inlined entry points, so that address and view numbers can be computed and output in debug information. This can be enabled independently of location views, in which case the view numbers won't be output, but it can only be enabled along with statement frontiers, and it is only enabled by default if location views are enabled.

−gz[**=***type*]
>   Produce compressed debug sections in DWARF format, if that is supported. If *type* is not given, the default type depends on the capabilities of the assembler and linker used. *type* may be one of **none** (don't compress debug sections), or **zlib** (use zlib compression in ELF gABI format). If the linker doesn't support writing compressed debug sections, the option is rejected. Otherwise, if the assembler does not support them, **−gz** is silently ignored when producing object files.

−femit−struct−debug−baseonly
>   Emit debug information for struct−like types only when the base name of the compilation source file matches the base name of file in which the struct is defined.

>   This option substantially reduces the size of debugging information, but at significant potential loss in type information to the debugger. See **−femit−struct−debug−reduced** for a less aggressive option. See **−femit−struct−debug−detailed** for more detailed control.

>   This option works only with DWARF debug output.

−femit−struct−debug−reduced
>   Emit debug information for struct−like types only when the base name of the compilation source file matches the base name of file in which the type is defined, unless the struct is a template or defined in a system header.

>   This option significantly reduces the size of debugging information, with some potential loss in type information to the debugger. See **−femit−struct−debug−baseonly** for a more aggressive option. See **−femit−struct−debug−detailed** for more detailed control.

>   This option works only with DWARF debug output.

−femit−struct−debug−detailed[**=***spec−list*]
>   Specify the struct−like types for which the compiler generates debug information. The intent is to reduce duplicate struct debug information between different object files within the same program.

>   This option is a detailed version of **−femit−struct−debug−reduced** and **−femit−struct−debug−baseonly**, which serves for most needs.

>   A specification has the syntax[**dir:|ind:**][**ord:|gen:**](**any|sys|base|none**)

>   The optional first word limits the specification to structs that are used directly (**dir:**) or used indirectly (**ind:**). A struct type is used directly when it is the type of a variable, member. Indirect uses arise through pointers to structs. That is, when use of an incomplete struct is valid, the use is indirect. An example is **struct one direct; struct two * indirect;**.

>   The optional second word limits the specification to ordinary structs (**ord:**) or generic structs (**gen:**). Generic structs are a bit complicated to explain. For C++, these are non−explicit specializations of template classes, or non−template classes within the above. Other programming languages have generics, but **−femit−struct−debug−detailed** does not yet implement them.

>   The third word specifies the source files for those structs for which the compiler should emit debug information. The values **none** and **any** have the normal meaning. The value **base** means that the base of name of the file in which the type declaration appears must match the base of the name of the main

compilation file. In practice, this means that when compiling *foo.c*, debug information is generated for types declared in that file and *foo.h*, but not other header files. The value **sys** means those types satisfying **base** or declared in system or compiler headers.

You may need to experiment to determine the best settings for your application.

The default is **−femit−struct−debug−detailed=all**.

This option works only with DWARF debug output.

−fno−dwarf2−cfi−asm
Emit DWARF unwind info as compiler generated `.eh_frame` section instead of using GAS `.cfi_*` directives.

−fno−eliminate−unused−debug−types
Normally, when producing DWARF output, GCC avoids producing debug symbol output for types that are nowhere used in the source file being compiled. Sometimes it is useful to have GCC emit debugging information for all types declared in a compilation unit, regardless of whether or not they are actually used in that compilation unit, for example if, in the debugger, you want to cast a value to a type that is not actually used in your program (but is declared). More often, however, this results in a significant amount of wasted space.

Options That Control Optimization
These options control various sorts of optimizations.

Without any optimization option, 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 expect from the source code.

Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

The compiler performs optimization based on the knowledge it has of the program. Compiling multiple files at once to a single output file mode allows the compiler to use information gained from all of the files when compiling each of them.

Not all optimizations are controlled directly by a flag. Only optimizations that have a flag are listed in this section.

Most optimizations are completely disabled at **−O0** or if an **−O** level is not set on the command line, even if individual optimization flags are specified. Similarly, **−Og** suppresses many optimization passes.

Depending on the target and how GCC was configured, a slightly different set of optimizations may be enabled at each **−O** level than those listed here. You can invoke GCC with **−Q −−help=optimizers** to find out the exact set of optimizations that are enabled at each level.

−O
−O1
Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

With **−O**, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

−O is the recommended optimization level for large machine−generated code as a sensible balance between time taken to compile and memory use: higher optimization levels perform optimizations with greater algorithmic complexity than at **−O**.

−O turns on the following optimization flags:

−fauto−inc−dec −fbranch−count−reg −fcombine−stack−adjustments −fcompare−elim −fcprop−registers −fdce −fdefer−pop −fdelayed−branch −fdse −fforward−propagate −fguess−branch−probability −fif−conversion −fif−conversion2 −finline−functions−called−once

−fipa−modref −fipa−profile −fipa−pure−const −fipa−reference −fipa−reference−addressable −fivopts −fmerge−constants −fmove−loop−invariants −fmove−loop−stores −fomit−frame−pointer −freorder−blocks −fshrink−wrap −fshrink−wrap−separate −fsplit−wide−types −fssa−backprop −fssa−phiopt −ftree−bit−ccp −ftree−ccp −ftree−ch −ftree−coalesce−vars −ftree−copy−prop −ftree−dce −ftree−dominator−opts −ftree−dse −ftree−forwprop −ftree−fre −ftree−phiprop −ftree−pta −ftree−scev−cprop −ftree−sink −ftree−slsr −ftree−sra −ftree−ter −funit−at−a−time

−O2

Optimize even more. GCC performs nearly all supported optimizations that do not involve a space−speed tradeoff. As compared to **−O**, this option increases both compilation time and the performance of the generated code.

−O2 turns on all optimization flags specified by **−O1**. It also turns on the following optimization flags:

−falign−functions −falign−jumps −falign−labels −falign−loops −fcaller−saves −fcode−hoisting −fcrossjumping −fcse−follow−jumps −fcse−skip−blocks −fdelete−null−pointer−checks −fdevirtualize −fdevirtualize−speculatively −fexpensive−optimizations −ffinite−loops −fgcse −fgcse−lm −fhoist−adjacent−loads −finline−functions −finline−small−functions −findirect−inlining −fipa−bit−cp −fipa−cp −fipa−icf −fipa−ra −fipa−sra −fipa−vrp −fisolate−erroneous−paths−dereference −flra−remat −foptimize−crc −foptimize−sibling−calls −foptimize−strlen −fpartial−inlining −fpeephole2 −freorder−blocks−algorithm=stc −freorder−blocks−and−partition −freorder−functions −frerun−cse−after−loop −fschedule−insns −fschedule−insns2 −fsched−interblock −fsched−spec −fstore−merging −fstrict−aliasing −fthread−jumps −ftree−builtin−call−dce −ftree−loop−vectorize −ftree−pre −ftree−slp−vectorize −ftree−switch−conversion −ftree−tail−merge −ftree−vrp −fvect−cost−model=very−cheap

Please note the warning under **−fgcse** about invoking **−O2** on programs that use computed gotos.

−O3

Optimize yet more. **−O3** turns on all optimizations specified by **−O2** and also turns on the following optimization flags:

−fgcse−after−reload −fipa−cp−clone −floop−interchange −floop−unroll−and−jam −fpeel−loops −fpredictive−commoning −fsplit−loops −fsplit−paths −ftree−loop−distribution −ftree−partial−pre −funswitch−loops −fvect−cost−model=dynamic −fversion−loops−for−strides

−O0

Reduce compilation time and make debugging produce the expected results. This is the default.

At **−O0**, GCC completely disables most optimization passes; they are not run even if you explicitly enable them on the command line, or are listed by **−Q −−help=optimizers** as being enabled by default. Many optimizations performed by GCC depend on code analysis or canonicalization passes that are enabled by **−O**, and it would not be useful to run individual optimization passes in isolation.

−Os

Optimize for size. **−Os** enables all **−O2** optimizations except those that often increase code size:

−falign−functions −falign−jumps −falign−labels −falign−loops −fprefetch−loop−arrays −freorder−blocks−algorithm=stc

It also enables **−finline−functions**, causes the compiler to tune for code size rather than execution speed, and performs further optimizations designed to reduce code size.

−Ofast

Disregard strict standards compliance. **−Ofast** enables all **−O3** optimizations. It also enables optimizations that are not valid for all standard−compliant programs. It turns on **−ffast−math**, **−fallow−store−data−races** and the Fortran−specific **−fstack−arrays**, unless **−fmax−stack−var−size** is specified, and **−fno−protect−parens**. It turns off **−fsemantic−interposition**.

–Og

Optimize debugging experience. **–Og** should be the optimization level of choice for the standard edit–compile–debug cycle, offering a reasonable level of optimization while maintaining fast compilation and a good debugging experience. It is a better choice than **–O0** for producing debuggable code because some compiler passes that collect debug information are disabled at **–O0**.

Like **–O0**, **–Og** completely disables a number of optimization passes so that individual options controlling them have no effect. Otherwise **–Og** enables all **–O1** optimization flags except for those that may interfere with debugging:

–fbranch–count–reg   **–fdelayed–branch**   **–fdse**   **–fif–conversion**   **–fif–conversion2** **–finline–functions–called–once** **–fmove–loop–invariants** **–fmove–loop–stores** **–fssa–phiopt** **–ftree–bit–ccp –ftree–dse –ftree–pta –ftree–sra**

–Oz

Optimize aggressively for size rather than speed. This may increase the number of instructions executed if those instructions require fewer bytes to encode. **–Oz** behaves similarly to **–Os** including enabling most **–O2** optimizations.

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** is **–fno–foo**. In the table below, only one of the forms is listed–––the one you typically use. You can figure out the other form by either removing **no–** or adding it.

The following options control specific optimizations. They are either activated by **–O** options or are related to ones that are. You can use the following flags in the rare cases when "fine–tuning" of optimizations to be performed is desired.

–fno–defer–pop

For machines that must pop arguments after a function call, always pop the arguments as soon as each function returns. At levels **–O1** and higher, **–fdefer–pop** is the default; this allows the compiler to let arguments accumulate on the stack for several function calls and pop them all at once.

–fforward–propagate

Perform a forward propagation pass on RTL. The pass tries to combine two instructions and checks if the result can be simplified. If loop unrolling is active, two passes are performed and the second is scheduled after loop unrolling.

This option is enabled by default at optimization levels **–O1**, **–O2**, **–O3**, **–Os**.

–favoid–store–forwarding
–fno–avoid–store–forwarding

Many CPUs will stall for many cycles when a load partially depends on previous smaller stores. This pass tries to detect such cases and avoid the penalty by changing the order of the load and store and then fixing up the loaded value.

Disabled by default.

–ffp–contract=*style*

–ffp–contract=off disables floating–point expression contraction. **–ffp–contract=fast** enables floating–point expression contraction such as forming of fused multiply–add operations if the target has native support for them. **–ffp–contract=on** enables floating–point expression contraction if allowed by the language standard. This is implemented for C and C++, where it enables contraction within one expression, but not across different statements.

The default is **–ffp–contract=off** for C in a standards compliant mode (**–std=c11** or similar), **–ffp–contract=fast** otherwise.

−fomit−frame−pointer

> Omit the frame pointer in functions that don't need one. This avoids the instructions to save, set up and restore the frame pointer; on many targets it also makes an extra register available.
>
> On some targets this flag has no effect because the standard calling sequence always uses a frame pointer, so it cannot be omitted.
>
> Note that **−fno−omit−frame−pointer** doesn't guarantee the frame pointer is used in all functions. Several targets always omit the frame pointer in leaf functions.
>
> Enabled by default at **−O1** and higher.

−foptimize−crc

> Detect loops calculating CRC (performing polynomial long division) and replace them with a faster implementation. Detect 8, 16, 32, and 64 bit CRC, with a constant polynomial without the leading 1 bit, for both bit−forward and bit−reversed cases. If the target supports a CRC instruction and the polynomial used in the source code matches the polynomial used in the CRC instruction, generate that CRC instruction. Otherwise, if the target supports a carry−less−multiplication instruction, generate CRC using it; otherwise generate table−based CRC.
>
> Enabled by default at **−O2** and higher.

−foptimize−sibling−calls

> Optimize sibling and tail recursive calls.
>
> Enabled at levels **−O2**, **−O3**, **−Os**.

−foptimize−strlen

> Optimize various standard C string functions (e.g. `strlen`, `strchr` or `strcpy`) and their `_FORTIFY_SOURCE` counterparts into faster alternatives.
>
> Enabled at levels **−O2**, **−O3**.

−finline−stringops[=*fn***]**

> Expand memory and string operations (for now, only `memset`) inline, even when the length is variable or big enough as to require looping. This is most useful along with **−ffreestanding** and **−fno−builtin**.
>
> In some circumstances, it enables the compiler to generate code that takes advantage of known alignment and length multipliers, but even then it may be less efficient than optimized runtime implementations, and grow code size so much that even a less performant but shared implementation runs faster due to better use of code caches. This option is disabled by default.

−fno−inline

> Do not expand any functions inline apart from those marked with the `always_inline` attribute. This is the default when not optimizing.
>
> Single functions can be exempted from inlining by marking them with the `noinline` attribute.

−finline−small−functions

> Integrate functions into their callers when their body is smaller than expected function call code (so overall size of program gets smaller). The compiler heuristically decides which functions are simple enough to be worth integrating in this way. This inlining applies to all functions, even those not declared inline.
>
> Enabled at levels **−O2**, **−O3**, **−Os**.

−findirect−inlining

> Inline also indirect calls that are discovered to be known at compile time thanks to previous inlining. This option has any effect only when inlining itself is turned on by the **−finline−functions** or **−finline−small−functions** options.
>
> Enabled at levels **−O2**, **−O3**, **−Os**.

−finline−functions

Consider all functions for inlining, even if they are not declared inline. The compiler heuristically decides which functions are worth integrating in this way.

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

Enabled at levels **−O2**, **−O3**, **−Os**. Also enabled by **−fprofile−use** and **−fauto−profile**.

−finline−functions−called−once

Consider all `static` functions called once for inlining into their caller even if they are not marked `inline`. If a call to a given function is integrated, then the function is not output as assembler code in its own right.

Enabled at levels **−O1**, **−O2**, **−O3** and **−Os**, but not **−Og**.

−fearly−inlining

Inline functions marked by `always_inline` and functions whose body seems smaller than the function call overhead early before doing **−fprofile−generate** instrumentation and real inlining pass. Doing so makes profiling significantly cheaper and usually inlining faster on programs having large chains of nested wrapper functions.

Enabled by default.

−fipa−sra

Perform interprocedural scalar replacement of aggregates, removal of unused parameters and replacement of parameters passed by reference by parameters passed by value.

Enabled at levels **−O2**, **−O3** and **−Os**.

−finline−limit=_n_

By default, GCC limits the size of functions that can be inlined. This flag allows coarse control of this limit. _n_ is the size of functions that can be inlined in number of pseudo instructions.

Inlining is actually controlled by a number of parameters, which may be specified individually by using **−−param** _name_=_value_. The **−finline−limit=**_n_ option sets some of these parameters as follows:

max−inline−insns−single
is set to _n_/2.

max−inline−insns−auto
is set to _n_/2.

See below for a documentation of the individual parameters controlling inlining and for the defaults of these parameters.

Note: there may be no value to **−finline−limit** that results in default behavior.

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

−fno−keep−inline−dllexport

This is a more fine−grained version of **−fkeep−inline−functions**, which applies only to functions that are declared using the `dllexport` attribute or declspec.

−fkeep−inline−functions

In C, emit `static` functions that are declared `inline` into the object file, even if the function has been inlined into all of its callers. This switch does not affect functions using the `extern inline` extension in GNU C90. In C++, emit any and all inline functions into the object file.

−fkeep−static−functions

Emit `static` functions into the object file, even if the function is never used.

−fkeep−static−consts

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

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

−fmerge−constants

Attempt to merge identical constants (string constants and floating−point constants) across compilation units.

This option is the default for optimized compilation if the assembler and linker support it. Use **−fno−merge−constants** to inhibit this behavior.

Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**.

−fmerge−all−constants

Attempt to merge identical constants and identical variables.

This option implies **−fmerge−constants**. In addition to **−fmerge−constants** this considers e.g. even constant initialized arrays or initialized constant variables with integral or floating−point types. Languages like C or C++ require each variable, including multiple instances of the same variable in recursive calls, to have distinct locations, so using this option results in non−conforming behavior.

−fmodulo−sched

Perform swing modulo scheduling immediately before the first scheduling pass. This pass looks at innermost loops and reorders their instructions by overlapping different iterations.

−fmodulo−sched−allow−regmoves

Perform more aggressive SMS−based modulo scheduling with register moves allowed. By setting this flag certain anti−dependences edges are deleted, which triggers the generation of reg−moves based on the life−range analysis. This option is effective only with **−fmodulo−sched** enabled.

−fno−branch−count−reg

Disable the optimization pass that scans for opportunities to use "decrement and branch" instructions on a count register instead of instruction sequences that decrement a register, compare it against zero, and then branch based upon the result. This option is only meaningful on architectures that support such instructions, which include x86, PowerPC, IA−64 and S/390. Note that the **−fno−branch−count−reg** option doesn't remove the decrement and branch instructions from the generated instruction stream introduced by other optimization passes.

The default is **−fbranch−count−reg** at **−O1** and higher, except for **−Og**.

−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.

The default is **−ffunction−cse**

−fno−zero−initialized−in−bss

If the target supports a BSS section, GCC by default puts variables that are initialized to zero into BSS. This can save space in the resulting code.

This option turns off this behavior because some programs explicitly rely on variables going to the data section−−−e.g., so that the resulting executable can find the beginning of that section and/or make assumptions based on that.

The default is **−fzero−initialized−in−bss** except in Ada.

−fthread−jumps

Perform optimizations that 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.

Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**.

−fsplit−wide−types

When using a type that occupies multiple registers, such as long long on a 32−bit system, split the registers apart and allocate them independently. This normally generates better code for those types, but may make debugging more difficult.

Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**.

−fsplit−wide−types−early

Fully split wide types early, instead of very late. This option has no effect unless **−fsplit−wide−types** is turned on.

This is the default on some targets.

−fcse−follow−jumps

In common subexpression elimination (CSE), 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 follows the jump when the condition tested is false.

Enabled at levels **−O2**, **−O3**, **−Os**.

−fcse−skip−blocks

This is similar to **−fcse−follow−jumps**, but causes CSE to follow jumps that conditionally skip over blocks. When CSE encounters a simple if statement with no else clause, **−fcse−skip−blocks** causes CSE to follow the jump around the body of the if.

Enabled at levels **−O2**, **−O3**, **−Os**.

−frerun−cse−after−loop

Re−run common subexpression elimination after loop optimizations are performed.

Enabled at levels **−O2**, **−O3**, **−Os**.

−fgcse

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

Note: When compiling a program using computed gotos, a GCC extension, you may get better run−time performance if you disable the global common subexpression elimination pass by adding **−fno−gcse** to the command line.

Enabled at levels **−O2**, **−O3**, **−Os**.

−fgcse−lm

When **−fgcse−lm** is enabled, global common subexpression elimination attempts to move loads that are only killed by stores into themselves. This allows a loop containing a load/store sequence to be changed to a load outside the loop, and a copy/store within the loop.

Enabled by default when **−fgcse** is enabled.

−fgcse−sm

When **−fgcse−sm** is enabled, a store motion pass is run after global common subexpression elimination. This pass attempts to move stores out of loops. When used in conjunction with **−fgcse−lm**, loops containing a load/store sequence can be changed to a load before the loop and a store after the loop.

Not enabled at any optimization level.

−fgcse−las

   When **−fgcse−las** is enabled, the global common subexpression elimination pass eliminates redundant loads that come after stores to the same memory location (both partial and full redundancies).

   Not enabled at any optimization level.

−fgcse−after−reload

   When **−fgcse−after−reload** is enabled, a redundant load elimination pass is performed after reload. The purpose of this pass is to clean up redundant spilling.

   Enabled by **−O3**, **−fprofile−use** and **−fauto−profile**.

−faggressive−loop−optimizations

   This option tells the loop optimizer to use language constraints to derive bounds for the number of iterations of a loop. This assumes that loop code does not invoke undefined behavior by for example causing signed integer overflows or out−of−bound array accesses. The bounds for the number of iterations of a loop are used to guide loop unrolling and peeling and loop exit test optimizations. This option is enabled by default.

−funconstrained−commons

   This option tells the compiler that variables declared in common blocks (e.g. Fortran) may later be overridden with longer trailing arrays. This prevents certain optimizations that depend on knowing the array bounds.

−fcrossjumping

   Perform cross−jumping transformation. This transformation unifies equivalent code and saves code size. The resulting code may or may not perform better than without cross−jumping.

   Enabled at levels **−O2**, **−O3**, **−Os**.

−fauto−inc−dec

   Combine increments or decrements of addresses with memory accesses. This pass is always skipped on architectures that do not have instructions to support this. Enabled by default at **−O1** and higher on architectures that support this.

−fdce

   Perform dead code elimination (DCE) on RTL. Enabled by default at **−O1** and higher.

−fdse

   Perform dead store elimination (DSE) on RTL. Enabled by default at **−O1** and higher.

−fif−conversion

   Attempt to transform conditional jumps into branch−less equivalents. This includes use of conditional moves, min, max, set flags and abs instructions, and some tricks doable by standard arithmetics. The use of conditional execution on chips where it is available is controlled by **−fif−conversion2**.

   Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**, but not with **−Og**.

−fif−conversion2

   Use conditional execution (where available) to transform conditional jumps into branch−less equivalents.

   Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**, but not with **−Og**.

−fdeclone−ctor−dtor

   The C++ ABI requires multiple entry points for constructors and destructors: one for a base subobject, one for a complete object, and one for a virtual destructor that calls operator delete afterwards. For a hierarchy with virtual bases, the base and complete variants are clones, which means two copies of the function. With this option, the base and complete variants are changed to be thunks that call a common implementation.

   Enabled by **−Os**.

−fdelete−null−pointer−checks

Assume that programs cannot safely dereference null pointers, and that no code or data element resides at address zero. This option enables simple constant folding optimizations at all optimization levels. In addition, other optimization passes in GCC use this flag to control global dataflow analyses that eliminate useless checks for null pointers; these assume that a memory access to address zero always results in a trap, so that if a pointer is checked after it has already been dereferenced, it cannot be null.

Note however that in some environments this assumption is not true. Use **−fno−delete−null−pointer−checks** to disable this optimization for programs that depend on that behavior.

This option is enabled by default on most targets. On AVR and MSP430, this option is completely disabled.

Passes that use the dataflow information are enabled independently at different optimization levels.

−fdevirtualize

Attempt to convert calls to virtual functions to direct calls. This is done both within a procedure and interprocedurally as part of indirect inlining (**−findirect−inlining**) and interprocedural constant propagation (**−fipa−cp**). Enabled at levels **−O2**, **−O3**, **−Os**.

−fdevirtualize-speculatively

Attempt to convert calls to virtual functions to speculative direct calls. Based on the analysis of the type inheritance graph, determine for a given call the set of likely targets. If the set is small, preferably of size 1, change the call into a conditional deciding between direct and indirect calls. The speculative calls enable more optimizations, such as inlining. When they seem useless after further optimization, they are converted back into original form.

−fdevirtualize−at−ltrans

Stream extra information needed for aggressive devirtualization when running the link−time optimizer in local transformation mode. This option enables more devirtualization but significantly increases the size of streamed data. For this reason it is disabled by default.

−fexpensive−optimizations

Perform a number of minor optimizations that are relatively expensive.

Enabled at levels **−O2**, **−O3**, **−Os**.

−free

Attempt to remove redundant extension instructions. This is especially helpful for the x86−64 architecture, which implicitly zero−extends in 64−bit registers after writing to their lower 32−bit half.

Enabled for Alpha, AArch64, LoongArch, PowerPC, RISC−V, SPARC, h83000 and x86 at levels **−O2**, **−O3**, **−Os**.

−fno−lifetime−dse

In C++ the value of an object is only affected by changes within its lifetime: when the constructor begins, the object has an indeterminate value, and any changes during the lifetime of the object are dead when the object is destroyed. Normally dead store elimination will take advantage of this; if your code relies on the value of the object storage persisting beyond the lifetime of the object, you can use this flag to disable this optimization. To preserve stores before the constructor starts (e.g. because your operator new clears the object storage) but still treat the object as dead after the destructor, you can use **−flifetime−dse=1**. The default behavior can be explicitly selected with **−flifetime−dse=2**. **−flifetime−dse=0** is equivalent to **−fno−lifetime−dse**.

−flive−range−shrinkage

Attempt to decrease register pressure through register live range shrinkage. This is helpful for fast processors with small or moderate size register sets.

−fira−algorithm=*algorithm*

Use the specified coloring algorithm for the integrated register allocator. The *algorithm* argument can be **priority**, which specifies Chow's priority coloring, or **CB**, which specifies Chaitin−Briggs coloring. Chaitin−Briggs coloring is not implemented for all architectures, but for those targets that do support it, it is the default because it generates better code.

−fira−region=*region*

Use specified regions for the integrated register allocator. The *region* argument should be one of the following:

all   Use all loops as register allocation regions. This can give the best results for machines with a small and/or irregular register set.

mixed

Use all loops except for loops with small register pressure as the regions. This value usually gives the best results in most cases and for most architectures, and is enabled by default when compiling with optimization for speed (**−O**, **−O2**, ...).

one   Use all functions as a single region. This typically results in the smallest code size, and is enabled by default for **−Os** or **−O0**.

−fira−hoist−pressure

Use IRA to evaluate register pressure in the code hoisting pass for decisions to hoist expressions. This option usually results in smaller code, but it can slow the compiler down.

This option is enabled at level **−Os** for all targets.

−fira−loop−pressure

Use IRA to evaluate register pressure in loops for decisions to move loop invariants. This option usually results in generation of faster and smaller code on machines with large register files (>= 32 registers), but it can slow the compiler down.

This option is enabled at level **−O3** for some targets.

−fno−ira−share−save−slots

Disable sharing of stack slots used for saving call−used hard registers living through a call. Each hard register gets a separate stack slot, and as a result function stack frames are larger.

−fno−ira−share−spill−slots

Disable sharing of stack slots allocated for pseudo−registers. Each pseudo−register that does not get a hard register gets a separate stack slot, and as a result function stack frames are larger.

−flra−remat

Enable CFG−sensitive rematerialization in LRA. Instead of loading values of spilled pseudos, LRA tries to rematerialize (recalculate) values if it is profitable.

Enabled at levels **−O2**, **−O3**, **−Os**.

−fdelayed−branch

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

Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**, but not at **−Og**.

−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.

Conventionally enabled at optimization levels **−O2** and **−O3**. However, many targets override this behavior. For example, on x86, it is disabled at all levels, while on AArch64, it is enabled only at **−O3**.

−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.

Enabled at levels **−O2**, **−O3**, **−Os**.

−fno−sched−interblock

Disable instruction scheduling across basic blocks, which is normally enabled when scheduling before register allocation, i.e. with **−fschedule−insns** or at **−O2** or higher.

−fno−sched−spec

Disable speculative motion of non−load instructions, which is normally enabled when scheduling before register allocation, i.e. with **−fschedule−insns** or at **−O2** or higher.

−fsched−pressure

Enable register pressure sensitive insn scheduling before register allocation. This only makes sense when scheduling before register allocation is enabled, i.e. with **−fschedule−insns** or at **−O2** or higher. Usage of this option can improve the generated code and decrease its size by preventing register pressure increase above the number of available hard registers and subsequent spills in register allocation.

−fsched−spec−load

Allow speculative motion of some load instructions. This only makes sense when scheduling before register allocation, i.e. with **−fschedule−insns** or at **−O2** or higher.

−fsched−spec−load−dangerous

Allow speculative motion of more load instructions. This only makes sense when scheduling before register allocation, i.e. with **−fschedule−insns** or at **−O2** or higher.

−fsched−stalled−insns
−fsched−stalled−insns=*n*

Define how many insns (if any) can be moved prematurely from the queue of stalled insns into the ready list during the second scheduling pass. **−fno−sched−stalled−insns** means that no insns are moved prematurely, **−fsched−stalled−insns=0** means there is no limit on how many queued insns can be moved prematurely. **−fsched−stalled−insns** without a value is equivalent to **−fsched−stalled−insns=1**.

−fsched−stalled−insns−dep
−fsched−stalled−insns−dep=*n*

Define how many insn groups (cycles) are examined for a dependency on a stalled insn that is a candidate for premature removal from the queue of stalled insns. This has an effect only during the second scheduling pass, and only if **−fsched−stalled−insns** is used. **−fno−sched−stalled−insns−dep** is equivalent to **−fsched−stalled−insns−dep=0**. **−fsched−stalled−insns−dep** without a value is equivalent to **−fsched−stalled−insns−dep=1**.

−fsched2−use−superblocks

When scheduling after register allocation, use superblock scheduling. This allows motion across basic block boundaries, resulting in faster schedules. This option is experimental, as not all machine descriptions used by GCC model the CPU closely enough to avoid unreliable results from the algorithm.

This only makes sense when scheduling after register allocation, i.e. with **−fschedule−insns2** or at **−O2** or higher.

−fsched−group−heuristic

Enable the group heuristic in the scheduler. This heuristic favors the instruction that belongs to a schedule group. This is enabled by default when scheduling is enabled, i.e. with **−fschedule−insns** or **−fschedule−insns2** or at **−O2** or higher.

−fsched−critical−path−heuristic

Enable the critical−path heuristic in the scheduler. This heuristic favors instructions on the critical path. This is enabled by default when scheduling is enabled, i.e. with **−fschedule−insns** or **−fschedule−insns2** or at **−O2** or higher.

−fsched−spec−insn−heuristic

Enable the speculative instruction heuristic in the scheduler. This heuristic favors speculative instructions with greater dependency weakness. This is enabled by default when scheduling is enabled, i.e. with **−fschedule−insns** or **−fschedule−insns2** or at **−O2** or higher.

−fsched−rank−heuristic

Enable the rank heuristic in the scheduler. This heuristic favors the instruction belonging to a basic block with greater size or frequency. This is enabled by default when scheduling is enabled, i.e. with **−fschedule−insns** or **−fschedule−insns2** or at **−O2** or higher.

−fsched−last−insn−heuristic

Enable the last−instruction heuristic in the scheduler. This heuristic favors the instruction that is less dependent on the last instruction scheduled. This is enabled by default when scheduling is enabled, i.e. with **−fschedule−insns** or **−fschedule−insns2** or at **−O2** or higher.

−fsched−dep−count−heuristic

Enable the dependent−count heuristic in the scheduler. This heuristic favors the instruction that has more instructions depending on it. This is enabled by default when scheduling is enabled, i.e. with **−fschedule−insns** or **−fschedule−insns2** or at **−O2** or higher.

−freschedule−modulo−scheduled−loops

Modulo scheduling is performed before traditional scheduling. If a loop is modulo scheduled, later scheduling passes may change its schedule. Use this option to control that behavior.

−fselective−scheduling

Schedule instructions using selective scheduling algorithm. Selective scheduling runs instead of the first scheduler pass.

−fselective−scheduling2

Schedule instructions using selective scheduling algorithm. Selective scheduling runs instead of the second scheduler pass.

−fsel−sched−pipelining

Enable software pipelining of innermost loops during selective scheduling. This option has no effect unless one of **−fselective−scheduling** or **−fselective−scheduling2** is turned on.

−fsel−sched−pipelining−outer−loops

When pipelining loops during selective scheduling, also pipeline outer loops. This option has no effect unless **−fsel−sched−pipelining** is turned on.

−fsemantic−interposition

Some object formats, like ELF, allow interposing of symbols by the dynamic linker. This means that for symbols exported from the DSO, the compiler cannot perform interprocedural propagation, inlining and other optimizations in anticipation that the function or variable in question may change. While this feature is useful, for example, to rewrite memory allocation functions by a debugging implementation, it is expensive in the terms of code quality. With **−fno−semantic−interposition** the compiler assumes that if interposition happens for functions the overwriting function will have precisely the same semantics (and side effects). Similarly if interposition happens for variables, the constructor of the variable will be the same. The flag has no effect for functions explicitly declared inline (where it is never allowed for interposition to change semantics) and for symbols explicitly declared weak.

−fshrink−wrap

Emit function prologues only before parts of the function that need it, rather than at the top of the function. This flag is enabled by default at **−O** and higher.

−fshrink−wrap−separate

Shrink−wrap separate parts of the prologue and epilogue separately, so that those parts are only executed when needed. This option is on by default, but has no effect unless **−fshrink−wrap** is also turned on and the target supports this.

−fcaller−saves

Enable allocation of values to registers that are 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.

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

Enabled at levels **−O2**, **−O3**, **−Os**.

−fcombine−stack−adjustments

Tracks stack adjustments (pushes and pops) and stack memory references and then tries to find ways to combine them.

Enabled by default at **−O1** and higher.

−fipa−ra

Use caller save registers for allocation if those registers are not used by any called function. In that case it is not necessary to save and restore them around calls. This is only possible if called functions are part of same compilation unit as current function and they are compiled before it.

Enabled at levels **−O2**, **−O3**, **−Os**, however the option is disabled if generated code will be instrumented for profiling (**−p**, or **−pg**) or if callee's register usage cannot be known exactly (this happens on targets that do not expose prologues and epilogues in RTL).

−fconserve−stack

Attempt to minimize stack usage. The compiler attempts to use less stack space, even if that makes the program slower. This option implies setting the **large−stack−frame** parameter to 100 and the **large−stack−frame−growth** parameter to 400.

−ftree−reassoc

Perform reassociation on trees. This flag is enabled by default at **−O1** and higher.

−fcode−hoisting

Perform code hoisting. Code hoisting tries to move the evaluation of expressions executed on all paths to the function exit as early as possible. This is especially useful as a code size optimization, but it often helps for code speed as well. This flag is enabled by default at **−O2** and higher.

−ftree−pre

Perform partial redundancy elimination (PRE) on trees. This flag is enabled by default at **−O2** and **−O3**.

−ftree−partial−pre

Make partial redundancy elimination (PRE) more aggressive. This flag is enabled by default at **−O3**.

−ftree−forwprop

Perform forward propagation on trees. This flag is enabled by default at **−O1** and higher.

−ftree−fre

Perform full redundancy elimination (FRE) on trees. The difference between FRE and PRE is that FRE only considers expressions that are computed on all paths leading to the redundant computation. This analysis is faster than PRE, though it exposes fewer redundancies. This flag is enabled by default at **−O1** and higher.

−ftree−phiprop

Perform hoisting of loads from conditional pointers on trees. This pass is enabled by default at **−O1** and higher.

−fhoist−adjacent−loads

Speculatively hoist loads from both branches of an if−then−else if the loads are from adjacent locations in the same structure and the target architecture has a conditional move instruction. This flag is enabled by default at **−O2** and higher.

−ftree−copy−prop

Perform copy propagation on trees. This pass eliminates unnecessary copy operations. This flag is enabled by default at **−O1** and higher.

−fipa−pure−const

Discover which functions are pure or constant. Enabled by default at **−O1** and higher.

−fipa−reference

Discover which static variables do not escape the compilation unit. Enabled by default at **−O1** and higher.

−fipa−reference−addressable

Discover read−only, write−only and non−addressable static variables. Enabled by default at **−O1** and higher.

−fipa−reorder−for−locality

Group call chains close together in the binary layout to improve code locality and minimize jump distances between frequently called functions. Unlike **−freorder−functions** this pass considers the call chains between functions and groups them together, rather than grouping all hot/normal/cold/never−executed functions into separate sections. Unlike **−fprofile−reorder−functions** it aims to improve code locality throughout the runtime of the program rather than focusing on program startup. This option is incompatible with an explicit **−flto−partition=** option since it enforces a custom partitioning scheme. If using this option it is recommended to also use profile feedback, but this option is not enabled by default otherwise.

−fipa−stack−alignment

Reduce stack alignment on call sites if possible. Enabled by default.

−fipa−pta

Perform interprocedural pointer analysis and interprocedural modification and reference analysis. This option can cause excessive memory and compile−time usage on large compilation units. It is not enabled by default at any optimization level.

−fipa−profile

Perform interprocedural profile propagation. The functions called only from cold functions are marked as cold. Also functions executed once (such as `cold`, `noreturn`, static constructors or destructors) are identified. Cold functions and loop less parts of functions executed once are then optimized for size. Enabled by default at **−O1** and higher.

−fipa−modref

Perform interprocedural mod/ref analysis. This optimization analyzes the side effects of functions (memory locations that are modified or referenced) and enables better optimization across the function call boundary. This flag is enabled by default at **−O1** and higher.

−fipa−cp

Perform interprocedural constant propagation. This optimization analyzes the program to determine when values passed to functions are constants and then optimizes accordingly. This optimization can substantially increase performance if the application has constants passed to functions. This flag is enabled by default at **−O2**, **−Os** and **−O3**. It is also enabled by **−fprofile−use** and **−fauto−profile**.

−fipa−cp−clone

Perform function cloning to make interprocedural constant propagation stronger. When enabled, interprocedural constant propagation performs function cloning when externally visible function can be called with constant arguments. Because this optimization can create multiple copies of functions, it may significantly increase code size (see **−−param ipa−cp−unit−growth=***value*). This flag is enabled by default at **−O3**. It is also enabled by **−fprofile−use** and **−fauto−profile**.

−fipa−bit−cp

When enabled, perform interprocedural bitwise constant propagation. This flag is enabled by default at **−O2** and by **−fprofile−use** and **−fauto−profile**. It requires that **−fipa−cp** is enabled.

−fipa−vrp

When enabled, perform interprocedural propagation of value ranges. This flag is enabled by default at **−O2**. It requires that **−fipa−cp** is enabled.

−fipa−icf

Perform Identical Code Folding for functions and read−only variables. The optimization reduces code size and may disturb unwind stacks by replacing a function by equivalent one with a different name. The optimization works more effectively with link−time optimization enabled.

Although the behavior is similar to the Gold Linker's ICF optimization, GCC ICF works on different levels and thus the optimizations are not same − there are equivalences that are found only by GCC and equivalences found only by Gold.

This flag is enabled by default at **−O2** and **−Os**.

−flate−combine−instructions

Enable two instruction combination passes that run relatively late in the compilation process. One of the passes runs before register allocation and the other after register allocation. The main aim of the passes is to substitute definitions into all uses.

Most targets enable this flag by default at **−O2** and **−Os**.

−flive−patching=_level_

Control GCC's optimizations to produce output suitable for live−patching.

If the compiler's optimization uses a function's body or information extracted from its body to optimize/change another function, the latter is called an impacted function of the former. If a function is patched, its impacted functions should be patched too.

The impacted functions are determined by the compiler's interprocedural optimizations. For example, a caller is impacted when inlining a function into its caller, cloning a function and changing its caller to call this new clone, or extracting a function's pureness/constness information to optimize its direct or indirect callers, etc.

Usually, the more IPA optimizations enabled, the larger the number of impacted functions for each function. In order to control the number of impacted functions and more easily compute the list of impacted function, IPA optimizations can be partially enabled at two different levels.

The _level_ argument should be one of the following:

inline−clone

Only enable inlining and cloning optimizations, which includes inlining, cloning, interprocedural scalar replacement of aggregates and partial inlining. As a result, when patching a function, all its callers and its clones' callers are impacted, therefore need to be patched as well.

−flive−patching=inline−clone disables the following optimization flags: **−fwhole−program −fipa−pta −fipa−reference −fipa−ra −fipa−icf −fipa−icf−functions −fipa−icf−variables −fipa−bit−cp −fipa−vrp −fipa−pure−const −fipa−reference−addressable −fipa−stack−alignment −fipa−modref**

inline−only−static

Only enable inlining of static functions. As a result, when patching a static function, all its callers are impacted and so need to be patched as well.

In addition to all the flags that **−flive−patching=inline−clone** disables, **−flive−patching=inline−only−static** disables the following additional optimization flags: **−fipa−cp−clone −fipa−sra −fpartial−inlining −fipa−cp**

When **−flive−patching** is specified without any value, the default value is _inline−clone_.

This flag is disabled by default.

Note that **−flive−patching** is not supported with link−time optimization (**−flto**).

−fisolate−erroneous−paths−dereference

Detect paths that trigger erroneous or undefined behavior due to dereferencing a null pointer. Isolate those paths from the main control flow and turn the statement with erroneous or undefined behavior into a trap. This flag is enabled by default at **−O2** and higher and depends on **−fdelete−null−pointer−checks** also being enabled.

−fisolate−erroneous−paths−attribute

Detect paths that trigger erroneous or undefined behavior due to a null value being used in a way forbidden by a returns_nonnull or nonnull attribute. Isolate those paths from the main control flow and turn the statement with erroneous or undefined behavior into a trap. This is not currently enabled, but may be enabled by **−O2** in the future.

−ftree−sink

Perform forward store motion on trees. This flag is enabled by default at **−O1** and higher.

−ftree−bit−ccp

Perform sparse conditional bit constant propagation on trees and propagate pointer alignment information. This pass only operates on local scalar variables and is enabled by default at **−O1** and higher, except for **−Og**. It requires that **−ftree−ccp** is enabled.

−ftree−ccp

Perform sparse conditional constant propagation (CCP) on trees. This pass only operates on local scalar variables and is enabled by default at **−O1** and higher.

−fssa−backprop

Propagate information about uses of a value up the definition chain in order to simplify the definitions. For example, this pass strips sign operations if the sign of a value never matters. The flag is enabled by default at **−O1** and higher.

−fssa−phiopt

Perform pattern matching on SSA PHI nodes to optimize conditional code. This pass is enabled by default at **−O1** and higher, except for **−Og**.

−ftree−switch−conversion

Perform conversion of simple initializations in a switch to initializations from a scalar array. This flag is enabled by default at **−O2** and higher.

−ftree−tail−merge

Look for identical code sequences. When found, replace one with a jump to the other. This optimization is known as tail merging or cross jumping. This flag is enabled by default at **−O2** and higher. The compilation time in this pass can be limited using **max−tail−merge−comparisons** parameter and **max−tail−merge−iterations** parameter.

−ftree−cselim

Perform conditional store elimination on trees. This flag is enabled by default at **−O1** and higher on targets that have conditional move instructions.

−ftree−dce

Perform dead code elimination (DCE) on trees. This flag is enabled by default at **−O1** and higher.

−ftree−builtin−call−dce

Perform conditional dead code elimination (DCE) for calls to built−in functions that may set errno but are otherwise free of side effects. This flag is enabled by default at **−O2** and higher if **−Os** is not also specified.

−ffinite−loops

Assume that a loop with an exit will eventually take the exit and not loop indefinitely. This allows the compiler to remove loops that otherwise have no side−effects, not considering eventual endless looping as such.

This option is enabled by default at **−O2** for C++ with −std=c++11 or higher.

−ftree−dominator−opts

Perform a variety of simple scalar cleanups (constant/copy propagation, redundancy elimination, range propagation and expression simplification) based on a dominator tree traversal. This also performs jump threading (to reduce jumps to jumps). This flag is enabled by default at **−O1** and higher.

−ftree−dse

Perform dead store elimination (DSE) on trees. A dead store is a store into a memory location that is later overwritten by another store without any intervening loads. In this case the earlier store can be deleted. This flag is enabled by default at **−O1** and higher.

−ftree−ch

Perform loop header copying on trees. This is beneficial since it increases effectiveness of code motion optimizations. It also saves one jump. This flag is enabled by default at **−O1** and higher. It is not enabled for **−Os**, since it usually increases code size.

−ftree−loop−optimize

Perform loop optimizations on trees. This flag is enabled by default at **−O1** and higher.

−ftree−loop−linear
−floop−strip−mine
−floop−block

Perform loop nest optimizations. Same as **−floop−nest−optimize**. To use this code transformation, GCC has to be configured with **−−with−isl** to enable the Graphite loop transformation infrastructure.

−fgraphite−identity

Enable the identity transformation for graphite. For every SCoP we generate the polyhedral representation and transform it back to gimple. Using **−fgraphite−identity** we can check the costs or benefits of the GIMPLE −> GRAPHITE −> GIMPLE transformation. Some minimal optimizations are also performed by the code generator isl, like index splitting and dead code elimination in loops.

−floop−nest−optimize

Enable the isl based loop nest optimizer. This is a generic loop nest optimizer based on the Pluto optimization algorithms. It calculates a loop structure optimized for data−locality and parallelism. This option is experimental.

−floop−parallelize−all

Use the Graphite data dependence analysis to identify loops that can be parallelized. Parallelize all the loops that can be analyzed to not contain loop carried dependences without checking that it is profitable to parallelize the loops.

−ftree−coalesce−vars

While transforming the program out of the SSA representation, attempt to reduce copying by coalescing versions of different user−defined variables, instead of just compiler temporaries. This may severely limit the ability to debug an optimized program compiled with **−fno−var−tracking−assignments**. In the negated form, this flag prevents SSA coalescing of user variables. This option is enabled by default if optimization is enabled, and it does very little otherwise.

−ftree−loop−if−convert

Attempt to transform conditional jumps in the innermost loops to branch−less equivalents. The intent is to remove control−flow from the innermost loops in order to improve the ability of the vectorization pass to handle these loops. This is enabled by default if vectorization is enabled.

−ftree−loop−distribution

Perform loop distribution. This flag can improve cache performance on big loop bodies and allow further loop optimizations, like parallelization or vectorization, to take place. For example, the loop

```
DO I = 1, N
  A(I) = B(I) + C
  D(I) = E(I) * F
ENDDO
```

is transformed to

```
DO I = 1, N
   A(I) = B(I) + C
ENDDO
DO I = 1, N
   D(I) = E(I) * F
ENDDO
```

This flag is enabled by default at **−O3**. It is also enabled by **−fprofile−use** and **−fauto−profile**.

−ftree−loop−distribute−patterns

Perform loop distribution of patterns that can be code generated with calls to a library. This flag is enabled by default at **−O2** and higher, and by **−fprofile−use** and **−fauto−profile**.

This pass distributes the initialization loops and generates a call to memset zero. For example, the loop

```
DO I = 1, N
  A(I) = 0
  B(I) = A(I) + I
ENDDO
```

is transformed to

```
DO I = 1, N
   A(I) = 0
ENDDO
DO I = 1, N
   B(I) = A(I) + I
ENDDO
```

and the initialization loop is transformed into a call to memset zero.

−floop−interchange

Perform loop interchange outside of graphite. This flag can improve cache performance on loop nest and allow further loop optimizations, like vectorization, to take place. For example, the loop

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

is transformed to

```
for (int i = 0; i < N; i++)
  for (int k = 0; k < N; k++)
    for (int j = 0; j < N; j++)
      c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

This flag is enabled by default at **−O3**. It is also enabled by **−fprofile−use** and **−fauto−profile**.

−floop−unroll−and−jam

Apply unroll and jam transformations on feasible loops. In a loop nest this unrolls the outer loop by some factor and fuses the resulting multiple inner loops. This flag is enabled by default at **−O3**. It is also enabled by **−fprofile−use** and **−fauto−profile**.

−ftree−loop−im

Perform loop invariant motion on trees. This pass moves only invariants that are hard to handle at RTL level (function calls, operations that expand to nontrivial sequences of insns). With **−funswitch−loops** it also moves operands of conditions that are invariant out of the loop, so that we can use just trivial invariantness analysis in loop unswitching. The pass also includes store motion.

−ftree−loop−ivcanon

Create a canonical counter for number of iterations in loops for which determining number of iterations requires complicated analysis. Later optimizations then may determine the number easily. Useful especially in connection with unrolling.

−ftree−scev−cprop

Perform final value replacement. If a variable is modified in a loop in such a way that its value when exiting the loop can be determined using only its initial value and the number of loop iterations, replace uses of the final value by such a computation, provided it is sufficiently cheap. This reduces data dependencies and may allow further simplifications. Enabled by default at **−O1** and higher.

−fivopts

Perform induction variable optimizations (strength reduction, induction variable merging and induction variable elimination) on trees. Enabled by default at **−O1** and higher.

−ftree−parallelize−loops=n

Parallelize loops, i.e., split their iteration space to run in n threads. This is only possible for loops whose iterations are independent and can be arbitrarily reordered. The optimization is only profitable on multiprocessor machines, for loops that are CPU−intensive, rather than constrained e.g. by memory bandwidth. This option implies **−pthread**, and thus is only supported on targets that have support for **−pthread**.

−ftree−pta

Perform function−local points−to analysis on trees. This flag is enabled by default at **−O1** and higher, except for **−Og**.

−ftree−sra

Perform scalar replacement of aggregates. This pass replaces structure references with scalars to prevent committing structures to memory too early. This flag is enabled by default at **−O1** and higher, except for **−Og**.

−fstore−merging

Perform merging of narrow stores to consecutive memory addresses. This pass merges contiguous stores of immediate values narrower than a word into fewer wider stores to reduce the number of instructions. This is enabled by default at **−O2** and higher as well as **−Os**.

−ftree−ter

Perform temporary expression replacement during the SSA−>normal phase. Single use/single def temporaries are replaced at their use location with their defining expression. This results in non−GIMPLE code, but gives the expanders much more complex trees to work on resulting in better RTL generation. This is enabled by default at **−O1** and higher.

−ftree−slsr

Perform straight−line strength reduction on trees. This recognizes related expressions involving multiplications and replaces them by less expensive calculations when possible. This is enabled by default at **−O1** and higher.

−ftree−vectorize

Perform vectorization on trees. This flag enables **−ftree−loop−vectorize** and **−ftree−slp−vectorize** if not explicitly specified.

−ftree−loop−vectorize

Perform loop vectorization on trees. This flag is enabled by default at **−O2** and by **−ftree−vectorize**, **−fprofile−use**, and **−fauto−profile**.

−ftree−slp−vectorize

Perform basic block vectorization on trees. This flag is enabled by default at **−O2** and by **−ftree−vectorize**, **−fprofile−use**, and **−fauto−profile**.

−ftrivial−auto−var−init=*choice*

Initialize automatic variables with either a pattern or with zeroes to increase the security and predictability of a program by preventing uninitialized memory disclosure and use. GCC still considers an automatic variable that doesn't have an explicit initializer as uninitialized, **−Wuninitialized** and **−Wanalyzer−use−of−uninitialized−value** will still report warning messages on such automatic variables and the compiler will perform optimization as if the variable were uninitialized. With this option, GCC will also initialize any padding of automatic variables that have structure or union types to zeroes. However, the current implementation cannot initialize automatic variables that are declared between the controlling expression and the first case of a `switch` statement. Using **−Wtrivial−auto−var−init** to report all such cases.

The three values of *choice* are:

*   **uninitialized** doesn't initialize any automatic variables. This is C and C++'s default.

*   **pattern** Initialize automatic variables with values which will likely transform logic bugs into crashes down the line, are easily recognized in a crash dump and without being values that programmers can rely on for useful program semantics. The current value is byte−repeatable pattern with byte "0xFE". The values used for pattern initialization might be changed in the future.

*   **zero** Initialize automatic variables with zeroes.

The default is **uninitialized**.

Note that the initializer values, whether **zero** or **pattern**, refer to data representation (in memory or machine registers), rather than to their interpretation as numerical values. This distinction may be important in languages that support types with biases or implicit multipliers, and with such extensions as **hardbool**. For example, a variable that uses 8 bits to represent (biased) quantities in the `range 160..400` will be initialized with the bit patterns `0x00` or `0xFE`, depending on *choice*, whether or not these representations stand for values in that range, and even if they do, the interpretation of the value held by the variable will depend on the bias. A **hardbool** variable that uses say `0X5A` and `0xA5` for `false` and `true`, respectively, will trap with either **choice** of trivial initializer, i.e., **zero** initialization will not convert to the representation for `false`, even if it would for a `static` variable of the same type. This means the initializer pattern doesn't generally depend on the type of the initialized variable. One notable exception is that (non−hardened) boolean variables that fit in registers are initialized with `false` (zero), even when **pattern** is requested.

You can control this behavior for a specific variable by using the variable attribute `uninitialized`.

−fvect−cost−model=*model*

Alter the cost model used for vectorization. The *model* argument should be one of **unlimited**, **dynamic**, **cheap** or **very−cheap**. With the **unlimited** model the vectorized code−path is assumed to be profitable while with the **dynamic** model a runtime check guards the vectorized code−path to enable it only for iteration counts that will likely execute faster than when executing the original scalar loop. The **cheap** model disables vectorization of loops where doing so would be cost prohibitive for example due to required runtime checks for data dependence or alignment but otherwise is equal to the **dynamic** model. The **very−cheap** model disables vectorization of loops when any runtime check for data dependence or alignment is required, it also disables vectorization of epilogue loops but otherwise is equal to the **cheap** model.

The default cost model depends on other optimization flags and is either **dynamic** or **cheap**.

−fsimd−cost−model=*model*

Alter the cost model used for vectorization of loops marked with the OpenMP simd directive. The *model* argument should be one of **unlimited**, **dynamic**, **cheap**. All values of *model* have the same meaning as described in **−fvect−cost−model** and by default a cost model defined with

−fvect−cost−model is used.

−ftree−vrp

Perform Value Range Propagation on trees. This is similar to the constant propagation pass, but instead of values, ranges of values are propagated. This allows the optimizers to remove unnecessary range checks like array bound checks and null pointer checks. This is enabled by default at **−O2** and higher. Null pointer check elimination is only done if **−fdelete−null−pointer−checks** is enabled.

−fsplit−paths

Split paths leading to loop backedges. This can improve dead code elimination and common subexpression elimination. This is enabled by default at **−O3** and above.

−fsplit−ivs−in−unroller

Enables expression of values of induction variables in later iterations of the unrolled loop using the value in the first iteration. This breaks long dependency chains, thus improving efficiency of the scheduling passes.

A combination of **−fweb** and CSE is often sufficient to obtain the same effect. However, that is not reliable in cases where the loop body is more complicated than a single basic block. It also does not work at all on some architectures due to restrictions in the CSE pass.

This optimization is enabled by default.

−fvariable−expansion−in−unroller

With this option, the compiler creates multiple copies of some local variables when unrolling a loop, which can result in superior code.

This optimization is enabled by default for PowerPC targets, but disabled by default otherwise.

−fpartial−inlining

Inline parts of functions. This option has any effect only when inlining itself is turned on by the **−finline−functions** or **−finline−small−functions** options.

Enabled at levels **−O2**, **−O3**, **−Os**.

−fpredictive−commoning

Perform predictive commoning optimization, i.e., reusing computations (especially memory loads and stores) performed in previous iterations of loops.

This option is enabled at level **−O3**. It is also enabled by **−fprofile−use** and **−fauto−profile**.

−fprefetch−loop−arrays

If supported by the target machine, generate instructions to prefetch memory to improve the performance of loops that access large arrays.

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

Disabled at level **−Os**.

−fno−printf−return−value

Do not substitute constants for known return value of formatted output functions such as `sprintf`, `snprintf`, `vsprintf`, and `vsnprintf` (but not `printf` of `fprintf`). This transformation allows GCC to optimize or even eliminate branches based on the known return value of these functions called with arguments that are either constant, or whose values are known to be in a range that makes determining the exact return value possible. For example, when **−fprintf−return−value** is in effect, both the branch and the body of the `if` statement (but not the call to `snprint`) can be optimized away when `i` is a 32−bit or smaller integer because the return value is guaranteed to be at most 8.

```
            char buf[9];
            if (snprintf (buf, "%08x", i) >= sizeof buf)
                ...
```

The **−fprintf−return−value** option relies on other optimizations and yields best results with **−O2** and above. It works in tandem with the **−Wformat−overflow** and **−Wformat−truncation** options. The **−fprintf−return−value** option is enabled by default.

−fno−peephole
−fno−peephole2

Disable any machine−specific peephole optimizations. The difference between **−fno−peephole** and **−fno−peephole2** is in how they are implemented in the compiler; some targets use one, some use the other, a few use both.

−fpeephole is enabled by default. **−fpeephole2** enabled at levels **−O2**, **−O3**, **−Os**.

−fno−guess−branch−probability

Do not guess branch probabilities using heuristics.

GCC uses heuristics to guess branch probabilities if they are not provided by profiling feedback (**−fprofile−arcs**). These heuristics are based on the control flow graph. If some branch probabilities are specified by `__builtin_expect`, then the heuristics are used to guess branch probabilities for the rest of the control flow graph, taking the `__builtin_expect` info into account. The interactions between the heuristics and `__builtin_expect` can be complex, and in some cases, it may be useful to disable the heuristics so that the effects of `__builtin_expect` are easier to understand.

It is also possible to specify expected probability of the expression with `__builtin_expect_with_probability` built−in function.

The default is **−fguess−branch−probability** at levels **−O**, **−O2**, **−O3**, **−Os**.

−freorder−blocks

Reorder basic blocks in the compiled function in order to reduce number of taken branches and improve code locality.

Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**.

−freorder−blocks−algorithm=_algorithm_

Use the specified algorithm for basic block reordering. The _algorithm_ argument can be **simple**, which does not increase code size (except sometimes due to secondary effects like alignment), or **stc**, the "software trace cache" algorithm, which tries to put all often executed code together, minimizing the number of branches executed by making extra copies of code.

The default is **simple** at levels **−O1**, **−Os**, and **stc** at levels **−O2**, **−O3**.

−freorder−blocks−and−partition

In addition to reordering basic blocks in the compiled function, in order to reduce number of taken branches, partitions hot and cold basic blocks into separate sections of the assembly and *.o* files, to improve paging and cache locality performance.

This optimization is automatically turned off in the presence of exception handling or unwind tables (on targets using setjump/longjump or target specific scheme), for linkonce sections, for functions with a user−defined section attribute and on any architecture that does not support named sections. When **−fsplit−stack** is used this option is not enabled by default (to avoid linker errors), but may be enabled explicitly (if using a working linker).

Enabled for x86 at levels **−O2**, **−O3**, **−Os**.

−freorder−functions

Reorder functions in the object file in order to improve code locality. Unlike **−fipa−reorder−for−locality** this option prioritises grouping all functions within a category (hot/normal/cold/never−executed) together. This is implemented by using special subsections

.text.hot for most frequently executed functions and .text.unlikely for unlikely executed functions. Reordering is done by the linker so object file format must support named sections and linker must place them in a reasonable way.

This option isn't effective unless you either provide profile feedback (see **–fprofile–arcs** for details) or manually annotate functions with hot or cold attributes.

Enabled at levels **–O2**, **–O3**, **–Os**.

–fstrict–aliasing

Allow 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() {
  union 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 works as expected. However, this code might not:

```
int f() {
  union 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**.

–fipa–strict–aliasing

Controls whether rules of **–fstrict–aliasing** are applied across function boundaries. Note that if multiple functions gets inlined into a single function the memory accesses are no longer considered to be crossing a function boundary.

The **–fipa–strict–aliasing** option is enabled by default and is effective only in combination with **–fstrict–aliasing**.

−falign−functions
−falign−functions=*n*
−falign−functions=*n***:***m*
−falign−functions=*n***:***m***:***n2*
−falign−functions=*n***:***m***:***n2***:***m2*

Align the start of functions to the next power−of−two greater than or equal to *n*, skipping up to *m*−1 bytes. This ensures that at least the first *m* bytes of the function can be fetched by the CPU without crossing an *n*−byte alignment boundary. This is an optimization of code performance and alignment is ignored for functions considered cold. If alignment is required for all functions, use **−fmin−function−alignment**.

If *m* is not specified, it defaults to *n*.

Examples: **−falign−functions=32** aligns functions to the next 32−byte boundary, **−falign−functions=24** aligns to the next 32−byte boundary only if this can be done by skipping 23 bytes or less, **−falign−functions=32:7** aligns to the next 32−byte boundary only if this can be done by skipping 6 bytes or less.

The second pair of *n2*:*m2* values allows you to specify a secondary alignment: **−falign−functions=64:7:32:3** aligns to the next 64−byte boundary if this can be done by skipping 6 bytes or less, otherwise aligns to the next 32−byte boundary if this can be done by skipping 2 bytes or less. If *m2* is not specified, it defaults to *n2*.

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

−fno−align−functions and **−falign−functions=1** are equivalent and mean that functions are not aligned.

If *n* is not specified or is zero, use a machine−dependent default. The maximum allowed *n* option value is 65536.

Enabled at levels **−O2**, **−O3**.

−flimit−function−alignment

If this option is enabled, the compiler tries to avoid unnecessarily overaligning functions. It attempts to instruct the assembler to align by the amount specified by **−falign−functions**, but not to skip more bytes than the size of the function.

−falign−labels
−falign−labels=*n*
−falign−labels=*n***:***m*
−falign−labels=*n***:***m***:***n2*
−falign−labels=*n***:***m***:***n2***:***m2*

Align all branch targets to a power−of−two boundary.

Parameters of this option are analogous to the **−falign−functions** option. **−fno−align−labels** and **−falign−labels=1** are equivalent and mean that labels are not aligned.

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

If *n* is not specified or is zero, use a machine−dependent default which is very likely to be **1**, meaning no alignment. The maximum allowed *n* option value is 65536.

Enabled at levels **−O2**, **−O3**.

−falign−loops
−falign−loops=*n*
−falign−loops=*n***:***m*
−falign−loops=*n***:***m***:***n2*

−falign−loops=*n***:***m***:***n2***:***m2*
>   Align loops to a power−of−two boundary. If the loops are executed many times, this makes up for any
>   execution of the dummy padding instructions. This is an optimization of code performance and
>   alignment is ignored for loops considered cold.
>
>   If **−falign−labels** is greater than this value, then its value is used instead.
>
>   Parameters of this option are analogous to the **−falign−functions** option. **−fno−align−loops** and
>   **−falign−loops=1** are equivalent and mean that loops are not aligned. The maximum allowed *n* option
>   value is 65536.
>
>   If *n* is not specified or is zero, use a machine−dependent default.
>
>   Enabled at levels **−O2**, **−O3**.

−falign−jumps
−falign−jumps=*n*
−falign−jumps=*n***:***m*
−falign−jumps=*n***:***m***:***n2*
−falign−jumps=*n***:***m***:***n2***:***m2*
>   Align branch targets to a power−of−two boundary, for branch targets where the targets can only be
>   reached by jumping. In this case, no dummy operations need be executed. This is an optimization of
>   code performance and alignment is ignored for jumps considered cold.
>
>   If **−falign−labels** is greater than this value, then its value is used instead.
>
>   Parameters of this option are analogous to the **−falign−functions** option. **−fno−align−jumps** and
>   **−falign−jumps=1** are equivalent and mean that loops are not aligned.
>
>   If *n* is not specified or is zero, use a machine−dependent default. The maximum allowed *n* option
>   value is 65536.
>
>   Enabled at levels **−O2**, **−O3**.

−fmin−function−alignment
>   Specify minimal alignment of functions to the next power−of−two greater than or equal to *n*. Unlike
>   **−falign−functions** this alignment is applied also to all functions (even those considered cold). The
>   alignment is also not affected by **−flimit−function−alignment**

−fno−allocation−dce
>   Do not remove unused C++ allocations (using operator `new` and operator `delete`) in dead code
>   elimination.
>
>   See also **−fmalloc−dce**.

−fallow−store−data−races
>   Allow the compiler to perform optimizations that may introduce new data races on stores, without
>   proving that the variable cannot be concurrently accessed by other threads. Does not affect
>   optimization of local data. It is safe to use this option if it is known that global data will not be
>   accessed by multiple threads.
>
>   Examples of optimizations enabled by **−fallow−store−data−races** include hoisting or if−conversions
>   that may cause a value that was already in memory to be re−written with that same value. Such
>   re−writing is safe in a single threaded context but may be unsafe in a multi−threaded context. Note
>   that on some processors, if−conversions may be required in order to enable vectorization.
>
>   Enabled at level **−Ofast**.

−funit−at−a−time
>   This option is left for compatibility reasons. **−funit−at−a−time** has no effect, while
>   **−fno−unit−at−a−time** implies **−fno−toplevel−reorder** and **−fno−section−anchors**.
>
>   Enabled by default.

–fno–toplevel–reorder

Do not reorder top–level functions, variables, and `asm` statements. Output them in the same order that they appear in the input file. When this option is used, unreferenced static variables are not removed. This option is intended to support existing code that relies on a particular ordering. For new code, it is better to use attributes when possible.

–ftoplevel–reorder is the default at **–O1** and higher, and also at **–O0** if **–fsection–anchors** is explicitly requested. Additionally **–fno–toplevel–reorder** implies **–fno–section–anchors**.

–funreachable–traps

With this option, the compiler turns calls to `__builtin_unreachable` into traps, instead of using them for optimization. This also affects any such calls implicitly generated by the compiler.

This option has the same effect as **–fsanitize=unreachable –fsanitize–trap=unreachable**, but does not affect the values of those options. If **–fsanitize=unreachable** is enabled, that option takes priority over this one.

This option is enabled by default at **–O0** and **–Og**.

–fweb

Constructs webs as commonly used for register allocation purposes and assign each web individual pseudo register. This allows the register allocation pass to operate on pseudos directly, but also strengthens several other optimization passes, such as CSE, loop optimizer and trivial dead code remover. It can, however, make debugging impossible, since variables no longer stay in a "home register".

Enabled by default with **–funroll–loops**.

–fwhole–program

Assume that the current compilation unit represents the whole program being compiled. All public functions and variables with the exception of `main` and those merged by attribute `externally_visible` become static functions and in effect are optimized more aggressively by interprocedural optimizers.

With **–flto** this option has a limited use. In most cases the precise list of symbols used or exported from the binary is known the resolution info passed to the link–time optimizer by the linker plugin. It is still useful if no linker plugin is used or during incremental link step when final code is produced (with **–flto –flinker–output=nolto–rel**).

–flto[=*n***]**

This option runs the standard link–time optimizer. When invoked with source code, it generates GIMPLE (one of GCC's internal representations) and writes it to special ELF sections in the object file. When the object files are linked together, all the function bodies are read from these ELF sections and instantiated as if they had been part of the same translation unit.

To use the link–time optimizer, **–flto** and optimization options should be specified at compile time and during the final link. It is recommended that you compile all the files participating in the same link with the same options and also specify those options at link time. For example:

```
gcc -c -O2 -flto foo.c
gcc -c -O2 -flto bar.c
gcc -o myprog -flto -O2 foo.o bar.o
```

The first two invocations to GCC save a bytecode representation of GIMPLE into special ELF sections inside *foo.o* and *bar.o*. The final invocation reads the GIMPLE bytecode from *foo.o* and *bar.o*, merges the two files into a single internal image, and compiles the result as usual. Since both *foo.o* and *bar.o* are merged into a single image, this causes all the interprocedural analyses and optimizations in GCC to work across the two files as if they were a single one. This means, for example, that the inliner is able to inline functions in *bar.o* into functions in *foo.o* and vice–versa.

Another (simpler) way to enable link–time optimization is:

```
gcc −o myprog −flto −O2 foo.c bar.c
```

The above generates bytecode for *foo.c* and *bar.c*, merges them together into a single GIMPLE representation and optimizes them as usual to produce *myprog*.

The important thing to keep in mind is that to enable link−time optimizations you need to use the GCC driver to perform the link step. GCC automatically performs link−time optimization if any of the objects involved were compiled with the **−flto** command−line option. You can always override the automatic decision to do link−time optimization by passing **−fno−lto** to the link command.

To make whole program optimization effective, it is necessary to make certain whole program assumptions. The compiler needs to know what functions and variables can be accessed by libraries and runtime outside of the link−time optimized unit. When supported by the linker, the linker plugin (see **−fuse−linker−plugin**) passes information to the compiler about used and externally visible symbols. When the linker plugin is not available, **−fwhole−program** should be used to allow the compiler to make these assumptions, which leads to more aggressive optimization decisions.

When a file is compiled with **−flto** without **−fuse−linker−plugin**, the generated object file is larger than a regular object file because it contains GIMPLE bytecodes and the usual final code (see **−ffat−lto−objects**). This means that object files with LTO information can be linked as normal object files; if **−fno−lto** is passed to the linker, no interprocedural optimizations are applied. Note that when **−fno−fat−lto−objects** is enabled the compile stage is faster but you cannot perform a regular, non−LTO link on them.

When producing the final binary, GCC only applies link−time optimizations to those files that contain bytecode. Therefore, you can mix and match object files and libraries with GIMPLE bytecodes and final object code. GCC automatically selects which files to optimize in LTO mode and which files to link without further processing.

Generally, options specified at link time override those specified at compile time, although in some cases GCC attempts to infer link−time options from the settings used to compile the input files.

If you do not specify an optimization level option **−O** at link time, then GCC uses the highest optimization level used when compiling the object files. Note that it is generally ineffective to specify an optimization level option only at link time and not at compile time, for two reasons. First, compiling without optimization suppresses compiler passes that gather information needed for effective optimization at link time. Second, some early optimization passes can be performed only at compile time and not at link time.

There are some code generation flags preserved by GCC when generating bytecodes, as they need to be used during the final link. Currently, the following options and their settings are taken from the first object file that explicitly specifies them: **−fcommon**, **−fexceptions**, **−fnon−call−exceptions**, **−fgnu−tm** and all the **−m** target flags.

The following options **−fPIC**, **−fpic**, **−fpie** and **−fPIE** are combined based on the following scheme:

```
B<-fPIC> + B<-fpic> = B<-fpic>
B<-fPIC> + B<-fno-pic> = B<-fno-pic>
B<-fpic/-fPIC> + (no option) = (no option)
B<-fPIC> + B<-fPIE> = B<-fPIE>
B<-fpic> + B<-fPIE> = B<-fpie>
B<-fPIC/-fpic> + B<-fpie> = B<-fpie>
```

Certain ABI−changing flags are required to match in all compilation units, and trying to override this at link time with a conflicting value is ignored. This includes options such as **−freg−struct−return** and **−fpcc−struct−return**.

Other options such as **−ffp−contract**, **−fno−strict−overflow**, **−fwrapv**, **−fno−trapv** or **−fno−strict−aliasing** are passed through to the link stage and merged conservatively for conflicting translation units. Specifically **−fno−strict−overflow**, **−fwrapv** and **−fno−trapv** take precedence; and

for example **−ffp−contract=off** takes precedence over **−ffp−contract=fast**. You can override them at link time.

Diagnostic options such as **−Wstringop−overflow** are passed through to the link stage and their setting matches that of the compile−step at function granularity. Note that this matters only for diagnostics emitted during optimization. Note that code transforms such as inlining can lead to warnings being enabled or disabled for regions if code not consistent with the setting at compile time.

When you need to pass options to the assembler via **−Wa** or **−Xassembler** make sure to either compile such translation units with **−fno−lto** or consistently use the same assembler options on all translation units. You can alternatively also specify assembler options at LTO link time.

To enable debug info generation you need to supply **−g** at compile time. If any of the input files at link time were built with debug info generation enabled the link will enable debug info generation as well. Any elaborate debug info settings like the dwarf level **−gdwarf−5** need to be explicitly repeated at the linker command line and mixing different settings in different translation units is discouraged.

If LTO encounters objects with C linkage declared with incompatible types in separate translation units to be linked together (undefined behavior according to ISO C99 6.2.7), a non−fatal diagnostic may be issued. The behavior is still undefined at run time. Similar diagnostics may be raised for other languages.

Another feature of LTO is that it is possible to apply interprocedural optimizations on files written in different languages:

```
gcc −c −flto foo.c
g++ −c −flto bar.cc
gfortran −c −flto baz.f90
g++ −o myprog −flto −O3 foo.o bar.o baz.o −lgfortran
```

Notice that the final link is done with **g++** to get the C++ runtime libraries and **−lgfortran** is added to get the Fortran runtime libraries. In general, when mixing languages in LTO mode, you should use the same link command options as when mixing languages in a regular (non−LTO) compilation.

If object files containing GIMPLE bytecode are stored in a library archive, say *libfoo.a*, it is possible to extract and use them in an LTO link if you are using a linker with plugin support. To create static libraries suitable for LTO, use **gcc−ar** and **gcc−ranlib** instead of **ar** and **ranlib**; to show the symbols of object files with GIMPLE bytecode, use **gcc−nm**. Those commands require that **ar**, **ranlib** and **nm** have been compiled with plugin support. At link time, use the flag **−fuse−linker−plugin** to ensure that the library participates in the LTO optimization process:

```
gcc −o myprog −O2 −flto −fuse−linker−plugin a.o b.o −lfoo
```

With the linker plugin enabled, the linker extracts the needed GIMPLE files from *libfoo.a* and passes them on to the running GCC to make them part of the aggregated GIMPLE image to be optimized.

If you are not using a linker with plugin support and/or do not enable the linker plugin, then the objects inside *libfoo.a* are extracted and linked as usual, but they do not participate in the LTO optimization process. In order to make a static library suitable for both LTO optimization and usual linkage, compile its object files with **−flto −ffat−lto−objects**.

Link−time optimizations do not require the presence of the whole program to operate. If the program does not require any symbols to be exported, it is possible to combine **−flto** and **−fwhole−program** to allow the interprocedural optimizers to use more aggressive assumptions which may lead to improved optimization opportunities. Use of **−fwhole−program** is not needed when linker plugin is active (see **−fuse−linker−plugin**).

The current implementation of LTO makes no attempt to generate bytecode that is portable between different types of hosts. The bytecode files are versioned and there is a strict version check, so bytecode files generated in one version of GCC do not work with an older or newer version of GCC.

Link−time optimization does not work well with generation of debugging information on systems other than those using a combination of ELF and DWARF.

If you specify the optional *n*, the optimization and code generation done at link time is executed in parallel using *n* parallel jobs by utilizing an installed **make** program. The environment variable **MAKE** may be used to override the program used.

You can also specify **−flto=jobserver** to use GNU make's job server mode to determine the number of parallel jobs. This is useful when the Makefile calling GCC is already executing in parallel. You must prepend a **+** to the command recipe in the parent Makefile for this to work. This option likely only works if **MAKE** is GNU make. Even without the option value, GCC tries to automatically detect a running GNU make's job server.

Use **−flto=auto** to use GNU make's job server, if available, or otherwise fall back to autodetection of the number of CPU threads present in your system.

−flto−partition=*alg*
> Specify the partitioning algorithm used by the link−time optimizer. The value is either **1to1** to specify a partitioning mirroring the original source files or **balanced** to specify partitioning into equally sized chunks (whenever possible) or **max** to create new partition for every symbol where possible or **cache** to balance chunk sizes while keeping related symbols together for better caching in incremental LTO. Specifying **none** as an algorithm disables partitioning and streaming completely. The default value is **balanced**. While **1to1** can be used as an workaround for various code ordering issues, the **max** partitioning is intended for internal testing only. The value **one** specifies that exactly one partition should be used while the value **none** bypasses partitioning and executes the link−time optimization step directly from the WPA phase.

−flto−incremental=*path*
> Enable incremental LTO, with its cache in given existing directory. Can significantly shorten edit−compile cycles with LTO.
>
> When used with LTO (**−flto**), the output of translation units inside LTO is cached. Cached translation units are likely to be encountered again when recompiling with small code changes, leading to recompile time reduction.
>
> Multiple GCC instances can use the same cache in parallel.

−flto−incremental−cache−size=*n*
> Specifies number of cache entries in incremental LTO after which to prune old entries. This is a soft limit, temporarily there may be more entries.

−flto−compression−level=*n*
> This option specifies the level of compression used for intermediate language written to LTO object files, and is only meaningful in conjunction with LTO mode (**−flto**). GCC currently supports two LTO compression algorithms. For zstd, valid values are 0 (no compression) to 19 (maximum compression), while zlib supports values from 0 to 9. Values outside this range are clamped to either minimum or maximum of the supported values. If the option is not given, a default balanced compression setting is used.

−fuse−linker−plugin
> Enables the use of a linker plugin during link−time optimization. This option relies on plugin support in the linker, which is available in gold or in GNU ld 2.21 or newer.
>
> This option enables the extraction of object files with GIMPLE bytecode out of library archives. This improves the quality of optimization by exposing more code to the link−time optimizer. This information specifies what symbols can be accessed externally (by non−LTO object or during dynamic linking). Resulting code quality improvements on binaries (and shared libraries that use hidden visibility) are similar to **−fwhole−program**. See **−flto** for a description of the effect of this flag and how to use it.
>
> This option is enabled by default when LTO support in GCC is enabled and GCC was configured for

use with a linker supporting plugins (GNU ld 2.21 or newer or gold).

−ffat−lto−objects

Fat LTO objects are object files that contain both the intermediate language and the object code. This makes them usable for both LTO linking and normal linking. This option is effective only when compiling with **−flto** and is ignored at link time.

−fno−fat−lto−objects improves compilation time over plain LTO, but requires the complete toolchain to be aware of LTO. It requires a linker with linker plugin support for basic functionality. Additionally, **nm**, **ar** and **ranlib** need to support linker plugins to allow a full−featured build environment (capable of building static libraries etc). GCC provides the **gcc−ar**, **gcc−nm**, **gcc−ranlib** wrappers to pass the right options to these tools. With non fat LTO makefiles need to be modified to use them.

Note that modern binutils provide plugin auto−load mechanism. Installing the linker plugin into *$libdir/bfd−plugins* has the same effect as usage of the command wrappers (**gcc−ar**, **gcc−nm** and **gcc−ranlib**).

The default is **−fno−fat−lto−objects** on targets with linker plugin support.

−fcompare−elim

After register allocation and post−register allocation instruction splitting, identify arithmetic instructions that compute processor flags similar to a comparison operation based on that arithmetic. If possible, eliminate the explicit comparison operation.

This pass only applies to certain targets that cannot explicitly represent the comparison operation before register allocation is complete.

Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**.

−ffold−mem−offsets
−fno−fold−mem−offsets

Try to eliminate add instructions by folding them in memory loads/stores.

Enabled at levels **−O2**, **−O3**.

−fcprop−registers

After register allocation and post−register allocation instruction splitting, perform a copy−propagation pass to try to reduce scheduling dependencies and occasionally eliminate the copy.

Enabled at levels **−O1**, **−O2**, **−O3**, **−Os**.

−fprofile−correction

Profiles collected using an instrumented binary for multi−threaded programs may be inconsistent due to missed counter updates. When this option is specified, GCC uses heuristics to correct or smooth out such inconsistencies. By default, GCC emits an error message when an inconsistent profile is detected.

This option is enabled by **−fauto−profile**.

−fprofile−partial−training

With `−fprofile−use` all portions of programs not executed during train run are optimized aggressively for size rather than speed. In some cases it is not practical to train all possible hot paths in the program. (For example, program may contain functions specific for a given hardware and training may not cover all hardware configurations program is run on.) With `−fprofile−partial−training` profile feedback is ignored for all functions not executed during the train run, leading them to be optimized as if they were compiled without profile feedback. This leads to better performance when train run is not representative but also leads to significantly bigger code.

−fprofile−use
−fprofile−use=*path*

Enable profile feedback−directed optimizations, and the following optimizations, many of which are generally profitable only with profile feedback available:

−fbranch−probabilities   −fprofile−values   −funroll−loops   −fpeel−loops   −ftracer   −fvpt −finline−functions   −fipa−cp   −fipa−cp−clone   −fipa−bit−cp   −fpredictive−commoning −fsplit−loops   −funswitch−loops   −fgcse−after−reload   −ftree−loop−vectorize −ftree−slp−vectorize   −fvect−cost−model=dynamic   −ftree−loop−distribute−patterns −fprofile−reorder−functions

Before you can use this option, you must first generate profiling information.

By default, GCC emits an error message if the feedback profiles do not match the source code. This error can be turned into a warning by using **−Wno−error=coverage−mismatch**. Note this may result in poorly optimized code. Additionally, by default, GCC also emits a warning message if the feedback profiles do not exist (see **−Wmissing−profile**).

If *path* is specified, GCC looks at the *path* to find the profile feedback data files. See **−fprofile−dir**.

−fauto−profile
−fauto−profile=*path*
Enable sampling−based feedback−directed optimizations, and the following optimizations, many of which are generally profitable only with profile feedback available:

−fbranch−probabilities   −fprofile−values   −funroll−loops   −fpeel−loops   −ftracer   −fvpt −finline−functions   −fipa−cp   −fipa−cp−clone   −fipa−bit−cp   −fpredictive−commoning −fsplit−loops   −funswitch−loops   −fgcse−after−reload   −ftree−loop−vectorize −ftree−slp−vectorize   −fvect−cost−model=dynamic   −ftree−loop−distribute−patterns −fprofile−correction

path is the name of a file containing AutoFDO profile information. If omitted, it defaults to *fbdata.afdo* in the current directory.

Producing an AutoFDO profile data file requires running your program with the **perf** utility on a supported GNU/Linux target system. For more information, see <**https://perf.wiki.kernel.org/**>.

E.g.

```
perf record −e br_inst_retired:near_taken −b −o perf.data \
    −− your_program
```

Then use the **create_gcov** tool to convert the raw profile data to a format that can be used by GCC. You must also supply the unstripped binary for your program to this tool. See <**https://github.com/google/autofdo**>.

E.g.

```
create_gcov --binary=your_program.unstripped --profile=perf.data \
    --gcov=profile.afdo
```

The following options control compiler behavior regarding floating−point arithmetic. These options trade off between speed and correctness. All must be specifically enabled.

−fexcess−precision=*style*
This option allows control over excess precision on machines where floating−point operations occur in a format with more precision or range than the IEEE standard and interchange floating−point types. An example of such a target is x87 floating point on x86 processors, which uses an 80−bit representation internally instead of the 64−bit IEEE format. For most programs, the excess precision is harmless, but some programs may rely on the requirements of the C or C++ language standards for handling IEEE values.

By default, **−fexcess−precision=fast** is in effect; this means that operations may be carried out in a wider precision than the types specified in the source if that would result in faster code, and it is unpredictable when rounding to the types specified in the source code takes place. When compiling C or C++, if **−fexcess−precision=standard** is specified then excess precision follows the rules specified in ISO C99 or C++; in particular, both casts and assignments cause values to be rounded to their

semantic types (whereas **−ffloat−store** only affects assignments). This option is enabled by default for C or C++ if a strict conformance option such as **−std=c99** or **−std=c++17** is used. **−ffast−math** enables **−fexcess−precision=fast** by default regardless of whether a strict conformance option is used. If **−fexcess−precision=16** is specified, constants and the results of expressions with types `_Float16` and `__bf16` are computed without excess precision.

−fexcess−precision=standard is not implemented for languages other than C or C++. On the x86, it has no effect if **−mfpmath=sse** or **−mfpmath=sse+387** is specified; in the former case, IEEE semantics apply without excess precision, and in the latter, rounding is unpredictable.

−ffloat−store
> Do not store floating−point variables in registers, and inhibit other options that might change whether a floating−point value is taken from a register or memory. This option has generally been subsumed by **−fexcess−precision=standard**, which is more general. If you do use **−ffloat−store**, you may need to modify your program to explicitly store intermediate computations in temporary variables since **−ffloat−store** handles rounding to IEEE format only on assignments and not casts as **−fexcess−precision=standard** does.

−ffast−math
> Sets the options **−fno−math−errno**, **−funsafe−math−optimizations**, **−ffinite−math−only**, **−fno−rounding−math**, **−fno−signaling−nans**, **−fcx−limited−range** and **−fexcess−precision=fast**.
>
> This option causes the preprocessor macro `__FAST_MATH__` to be defined.
>
> This option is not turned on by any **−O** option besides **−Ofast** since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications.

−fno−math−errno
> Do not set `errno` after calling math functions that are executed with a single instruction, e.g., `sqrt`. A program that relies on IEEE exceptions for math error handling may want to use this flag for speed while maintaining IEEE arithmetic compatibility.
>
> This option is not turned on by any **−O** option besides **−Ofast** since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications.
>
> The default is **−fmath−errno**.
>
> On Darwin systems, the math library never sets `errno`. There is therefore no reason for the compiler to consider the possibility that it might, and **−fno−math−errno** is the default.

−funsafe−math−optimizations
> Allow optimizations for floating−point arithmetic that (a) assume that arguments and results are valid and (b) may violate IEEE or ANSI standards. When used at link time, it may include libraries or startup files that change the default FPU control word or other similar optimizations.
>
> This option is not turned on by any **−O** option besides **−Ofast** since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications. Enables **−fno−signed−zeros**, **−fno−trapping−math**, **−fassociative−math** and **−freciprocal−math**.
>
> The default is **−fno−unsafe−math−optimizations**.

−fassociative−math
> Allow re−association of operands in series of floating−point operations. This violates the ISO C and C++ language standard by possibly changing computation result. NOTE: re−ordering may change the sign of zero as well as ignore NaNs and inhibit or create underflow or overflow (and thus cannot be

used on code that relies on rounding behavior like $(x + 2**52) - 2**52$. May also reorder floating−point comparisons and thus may not be used when ordered comparisons are required. This option requires that both **−fno−signed−zeros** and **−fno−trapping−math** be in effect. Moreover, it doesn't make much sense with **−frounding−math**. For Fortran the option is automatically enabled when both **−fno−signed−zeros** and **−fno−trapping−math** are in effect.

The default is **−fno−associative−math**.

−freciprocal−math

Allow the reciprocal of a value to be used instead of dividing by the value if this enables optimizations. For example x / y can be replaced with $x * (1/y)$, which is useful if $(1/y)$ is subject to common subexpression elimination. Note that this loses precision and increases the number of flops operating on the value.

The default is **−fno−reciprocal−math**.

−ffinite−math−only

Allow optimizations for floating−point arithmetic that assume that arguments and results are not NaNs or +−Infs.

This option is not turned on by any **−O** option besides **−Ofast** since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications.

The default is **−fno−finite−math−only**.

−fno−signed−zeros

Allow optimizations for floating-point arithmetic that ignore the signedness of zero. IEEE arithmetic specifies the behavior of distinct +0.0 and −0.0 values, which then prohibits simplification of expressions such as x+0.0 or 0.0*x (even with **−ffinite−math−only**). This option implies that the sign of a zero result isn't significant.

The default is **−fsigned−zeros**.

−fno−trapping−math

Compile code assuming that floating−point operations cannot generate user−visible traps. These traps include division by zero, overflow, underflow, inexact result and invalid operation. This option requires that **−fno−signaling−nans** be in effect. Setting this option may allow faster code if one relies on "non−stop" IEEE arithmetic, for example.

This option is not turned on by any **−O** option besides **−Ofast** since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions.

The default is **−ftrapping−math**.

Future versions of GCC may provide finer control of this setting using C99's FENV_ACCESS pragma. This command−line option will be used along with **−frounding−math** to specify the default state for FENV_ACCESS.

−frounding−math

Disable transformations and optimizations that assume default floating−point rounding behavior (round−to−nearest). This option should be specified for programs that change the FP rounding mode dynamically, or that may be executed with a non−default rounding mode. This option disables constant folding of floating−point expressions at compile time (which may be affected by rounding mode) and arithmetic transformations that are unsafe in the presence of sign−dependent rounding modes.

The default is **−fno−rounding−math**.

This option is experimental and does not currently guarantee to disable all GCC optimizations that are

affected by rounding mode. Future versions of GCC may provide finer control of this setting using C99's `FENV_ACCESS` pragma. This command−line option will be used along with **−ftrapping−math** to specify the default state for `FENV_ACCESS`.

−fsignaling−nans

Compile code assuming that IEEE signaling NaNs may generate user−visible traps during floating−point operations. Setting this option disables optimizations that may change the number of exceptions visible with signaling NaNs. This option implies **−ftrapping−math**.

This option causes the preprocessor macro `__SUPPORT_SNAN__` to be defined.

The default is **−fno−signaling−nans**.

This option is experimental and does not currently guarantee to disable all GCC optimizations that affect signaling NaN behavior.

−fno−fp−int−builtin−inexact

Do not allow the built−in functions `ceil`, `floor`, `round` and `trunc`, and their `float` and `long double` variants, to generate code that raises the "inexact" floating−point exception for noninteger arguments. ISO C99 and C11 allow these functions to raise the "inexact" exception, but ISO/IEC TS 18661−1:2014, the C bindings to IEEE 754−2008, as integrated into ISO C23, does not allow these functions to do so.

The default is **−ffp−int−builtin−inexact**, allowing the exception to be raised, unless C23 or a later C standard is selected. This option does nothing unless **−ftrapping−math** is in effect.

Even if **−fno−fp−int−builtin−inexact** is used, if the functions generate a call to a library function then the "inexact" exception may be raised if the library implementation does not follow TS 18661.

−fsingle−precision−constant

Treat floating−point constants as single precision instead of implicitly converting them to double−precision constants.

−fcx−limited−range

When enabled, this option states that a range reduction step is not needed when performing complex division. Also, there is no checking whether the result of a complex multiplication or division is `NaN + I*NaN`, with an attempt to rescue the situation in that case. The default is **−fno−cx−limited−range**, but is enabled by **−ffast−math**.

This option controls the default setting of the ISO C99 `CX_LIMITED_RANGE` pragma. Nevertheless, the option applies to all languages.

−fcx−fortran−rules

Complex multiplication and division follow Fortran rules. Range reduction is done as part of complex division, but there is no checking whether the result of a complex multiplication or division is `NaN + I*NaN`, with an attempt to rescue the situation in that case.

The default is **−fno−cx−fortran−rules**.

The following options control optimizations that may improve performance, but are not enabled by any **−O** options. This section includes experimental options that may produce broken code.

−fbranch−probabilities

After running a program compiled with **−fprofile−arcs**, you can compile it a second time using **−fbranch−probabilities**, to improve optimizations based on the number of times each branch was taken. When a program compiled with **−fprofile−arcs** exits, it saves arc execution counts to a file called *sourcename.gcda* for each source file. The information in this data file is very dependent on the structure of the generated code, so you must use the same source code and the same optimization options for both compilations. See details about the file naming in **−fprofile−arcs**.

With **−fbranch−probabilities**, GCC puts a **REG_BR_PROB** note on each **JUMP_INSN** and **CALL_INSN**. These can be used to improve optimization. Currently, they are only used in one place: in *reorg.cc*, instead of guessing which path a branch is most likely to take, the

REG_BR_PROB values are used to exactly determine which path is taken more often.

Enabled by **−fprofile−use** and **−fauto−profile**.

−fprofile−values

If combined with **−fprofile−arcs**, it adds code so that some data about values of expressions in the program is gathered.

With **−fbranch−probabilities**, it reads back the data gathered from profiling values of expressions for usage in optimizations.

Enabled by **−fprofile−generate**, **−fprofile−use**, and **−fauto−profile**.

−fprofile−reorder−functions

Function reordering based on profile instrumentation collects first time of execution of a function and orders these functions in ascending order, aiming to optimize program startup through more efficient loading of text segments.

Enabled with **−fprofile−use**.

−fvpt

If combined with **−fprofile−arcs**, this option instructs the compiler to add code to gather information about values of expressions.

With **−fbranch−probabilities**, it reads back the data gathered and actually performs the optimizations based on them. Currently the optimizations include specialization of division operations using the knowledge about the value of the denominator.

Enabled with **−fprofile−use** and **−fauto−profile**.

−frename−registers

Attempt to avoid false dependencies in scheduled code by making use of registers left over after register allocation. This optimization most benefits processors with lots of registers. Depending on the debug information format adopted by the target, however, it can make debugging impossible, since variables no longer stay in a "home register".

Enabled by default with **−funroll−loops**.

−fschedule−fusion

Performs a target dependent pass over the instruction stream to schedule instructions of same type together because target machine can execute them more efficiently if they are adjacent to each other in the instruction flow.

Enabled at levels **−O2**, **−O3**, **−Os**.

−ftracer

Perform tail duplication to enlarge superblock size. This transformation simplifies the control flow of the function allowing other optimizations to do a better job.

Enabled by **−fprofile−use** and **−fauto−profile**.

−funroll−loops

Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. **−funroll−loops** implies **−frerun−cse−after−loop**, **−fweb** and **−frename−registers**. It also turns on complete loop peeling (i.e. complete removal of loops with a small constant number of iterations). This option makes code larger, and may or may not make it run faster.

Enabled by **−fprofile−use** and **−fauto−profile**.

−funroll−all−loops

Unroll all loops, even if their number of iterations is uncertain when the loop is entered. This usually makes programs run more slowly. **−funroll−all−loops** implies the same options as **−funroll−loops**.

−fpeel−loops

Peels loops for which there is enough information that they do not roll much (from profile feedback or static analysis). It also turns on complete loop peeling (i.e. complete removal of loops with small constant number of iterations).

Enabled by **−O3**, **−fprofile−use**, and **−fauto−profile**.

−fmalloc−dce

Control whether malloc (and its variants such as calloc or strdup), can be optimized away provided its return value is only used as a parameter of free call or compared with NULL. If **−fmalloc−dce=1** is used, only calls to free are allowed while with **−fmalloc−dce=2** also comparisons with NULL pointer are considered safe to remove.

The default is **−fmalloc−dce=2**. See also **−fallocation−dce**.

−fmove−loop−invariants

Enables the loop invariant motion pass in the RTL loop optimizer. Enabled at level **−O1** and higher, except for **−Og**.

−fmove−loop−stores

Enables the loop store motion pass in the GIMPLE loop optimizer. This moves invariant stores to after the end of the loop in exchange for carrying the stored value in a register across the iteration. Note for this option to have an effect **−ftree−loop−im** has to be enabled as well. Enabled at level **−O1** and higher, except for **−Og**.

−fsplit−loops

Split a loop into two if it contains a condition that's always true for one side of the iteration space and false for the other.

Enabled by **−fprofile−use** and **−fauto−profile**.

−funswitch−loops

Move branches with loop invariant conditions out of the loop, with duplicates of the loop on both branches (modified according to result of the condition).

Enabled by **−fprofile−use** and **−fauto−profile**.

−fversion−loops−for−strides

If a loop iterates over an array with a variable stride, create another version of the loop that assumes the stride is always one. For example:

```
for (int i = 0; i < n; ++i)
  x[i * stride] = ...;
```

becomes:

```
if (stride == 1)
  for (int i = 0; i < n; ++i)
    x[i] = ...;
else
  for (int i = 0; i < n; ++i)
    x[i * stride] = ...;
```

This is particularly useful for assumed−shape arrays in Fortran where (for example) it allows better vectorization assuming contiguous accesses. This flag is enabled by default at **−O3**. It is also enabled by **−fprofile−use** and **−fauto−profile**.

−ffunction−sections
−fdata−sections

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

Use these options on systems where the linker can perform optimizations to improve locality of reference in the instruction space. Most systems using the ELF object format have linkers with such optimizations. On AIX, the linker rearranges sections (CSECTs) based on the call graph. The performance impact varies.

Together with a linker garbage collection (linker **−−gc−sections** option) these options may lead to smaller statically−linked executables (after stripping).

On ELF/DWARF systems these options do not degenerate the quality of the debug information. There could be issues with other object files/debug info formats.

Only use these options when there are significant benefits from doing so. When you specify these options, the assembler and linker create larger object and executable files and are also slower. These options affect code generation. They prevent optimizations by the compiler and assembler using relative locations inside a translation unit since the locations are unknown until link time. An example of such an optimization is relaxing calls to short call instructions.

−fstdarg−opt

Optimize the prologue of variadic argument functions with respect to usage of those arguments.

−fsection−anchors

Try to reduce the number of symbolic address calculations by using shared "anchor" symbols to address nearby objects. This transformation can help to reduce the number of GOT entries and GOT accesses on some targets.

For example, the implementation of the following function foo:

```
static int a, b, c;
int foo (void) { return a + b + c; }
```

usually calculates the addresses of all three variables, but if you compile it with **−fsection−anchors**, it accesses the variables from a common anchor point instead. The effect is similar to the following pseudocode (which isn't valid C):

```
int foo (void)
{
  register int *xr = &x;
  return xr[&a − &x] + xr[&b − &x] + xr[&c − &x];
}
```

Not all targets support this option.

−fzero−call−used−regs=*choice*

Zero call−used registers at function return to increase program security by either mitigating Return−Oriented Programming (ROP) attacks or preventing information leakage through registers.

The possible values of *choice* are the same as for the zero_call_used_regs attribute. The default is **skip**.

You can control this behavior for a specific function by using the function attribute zero_call_used_regs.

−−param *name*=*value*

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

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 order to get the minimal, maximal and default values of a parameter, use the **−−help=param −Q** options.

In each case, the *value* is an integer. The following choices of *name* are recognized for all targets:

phiopt−factor−max−stmts−live
> When factoring statements out of if/then/else, this is the max # of statements after the defining statement to be allow to extend the lifetime of a name

predictable−branch−outcome
> When branch is predicted to be taken with probability lower than this threshold (in percent), then it is considered well predictable.

max−rtl−if−conversion−insns
> RTL if−conversion tries to remove conditional branches around a block and replace them with conditionally executed instructions. This parameter gives the maximum number of instructions in a block which should be considered for if−conversion. The compiler will also use other heuristics to decide whether if−conversion is likely to be profitable.

file−cache−files
> Max number of files in the file cache. The file cache is used to print source lines in diagnostics and do some source checks like **−Wmisleading−indentation**.

file−cache−lines
> Max number of lines to index into file cache. When 0 this is automatically sized. The file cache is used to print source lines in diagnostics and do some source checks like **−Wmisleading−indentation**.

max−rtl−if−conversion−predictable−cost
> RTL if−conversion will try to remove conditional branches around a block and replace them with conditionally executed instructions. These parameters give the maximum permissible cost for the sequence that would be generated by if−conversion depending on whether the branch is statically determined to be predictable or not. The units for this parameter are the same as those for the GCC internal seq_cost metric. The compiler will try to provide a reasonable default for this parameter using the BRANCH_COST target macro.

max−crossjump−edges
> The maximum number of incoming edges to consider for cross−jumping. The algorithm used by **−fcrossjumping** is O(N^2) in the number of edges incoming to each block. Increasing values mean more aggressive optimization, making the compilation time increase with probably small improvement in executable size.

min−crossjump−insns
> The minimum number of instructions that must be matched at the end of two blocks before cross−jumping is performed on them. This value is ignored in the case where all instructions in the block being cross−jumped from are matched.

max−grow−copy−bb−insns
> The maximum code size expansion factor when copying basic blocks instead of jumping. The expansion is relative to a jump instruction.

max−goto−duplication−insns
> The maximum number of instructions to duplicate to a block that jumps to a computed goto. To avoid O(N^2) behavior in a number of passes, GCC factors computed gotos early in the compilation process, and unfactors them as late as possible. Only computed jumps at the end of a basic blocks with no more than max−goto−duplication−insns are unfactored.

max−delay−slot−insn−search
> The maximum number of instructions to consider when looking for an instruction to fill a delay slot. If more than this arbitrary number of instructions are searched, the time savings from filling the delay slot are minimal, so stop searching. Increasing values mean more aggressive optimization, making the compilation time increase with probably small improvement in execution time.

max−delay−slot−live−search

When trying to fill delay slots, the maximum number of instructions to consider when searching for a block with valid live register information. Increasing this arbitrarily chosen value means more aggressive optimization, increasing the compilation time. This parameter should be removed when the delay slot code is rewritten to maintain the control−flow graph.

max−gcse−memory

The approximate maximum amount of memory in kB that can be allocated in order to perform the global common subexpression elimination optimization. If more memory than specified is required, the optimization is not done.

max−gcse−insertion−ratio

If the ratio of expression insertions to deletions is larger than this value for any expression, then RTL PRE inserts or removes the expression and thus leaves partially redundant computations in the instruction stream.

max−pending−list−length

The maximum number of pending dependencies scheduling allows before flushing the current state and starting over. Large functions with few branches or calls can create excessively large lists which needlessly consume memory and resources.

max−modulo−backtrack−attempts

The maximum number of backtrack attempts the scheduler should make when modulo scheduling a loop. Larger values can exponentially increase compilation time.

max−inline−functions−called−once−loop−depth

Maximal loop depth of a call considered by inline heuristics that tries to inline all functions called once.

max−inline−functions−called−once−insns

Maximal estimated size of functions produced while inlining functions called once.

max−inline−insns−single

Several parameters control the tree inliner used in GCC. This number sets the maximum number of instructions (counted in GCC's internal representation) in a single function that the tree inliner considers for inlining. This only affects functions declared inline and methods implemented in a class declaration (C++).

max−inline−insns−auto

When you use **−finline−functions** (included in **−O3**), a lot of functions that would otherwise not be considered for inlining by the compiler are investigated. To those functions, a different (more restrictive) limit compared to functions declared inline can be applied (**−−param max−inline−insns−auto**).

max−inline−insns−small

This is the bound applied to calls that are considered relevant with **−finline−small−functions**.

max−inline−insns−size

This is the bound applied to calls that are optimized for size. Small growth may be desirable to anticipate optimization opportunities exposed by inlining.

uninlined−function−insns

Number of instructions accounted by inliner for function overhead such as function prologue and epilogue.

uninlined−function−time

Extra time accounted by inliner for function overhead such as time needed to execute function prologue and epilogue.

inline−heuristics−hint−percent

The scale (in percents) applied to **inline−insns−single**, **inline−insns−single−O2**, **inline−insns−auto** when inline heuristics hints that inlining is very profitable (will enable later

optimizations).

uninlined−thunk−insns
uninlined−thunk−time

Same as **−−param uninlined−function−insns** and **−−param uninlined−function−time** but applied to function thunks.

inline−min−speedup

When estimated performance improvement of caller + callee runtime exceeds this threshold (in percent), the function can be inlined regardless of the limit on **−−param max−inline−insns−single** and **−−param max−inline−insns−auto**.

large−function−insns

The limit specifying really large functions. For functions larger than this limit after inlining, inlining is constrained by **−−param large−function−growth**. This parameter is useful primarily to avoid extreme compilation time caused by non−linear algorithms used by the back end.

large−function−growth

Specifies maximal growth of large functions caused by inlining in percents. For example, parameter value 100 limits large function growth to 2.0 times the original size.

large−unit−insns

The limit specifying large translation unit. Growth caused by inlining of units larger than this limit is limited by **−−param inline−unit−growth**. For small units this might be too tight. For example, consider a unit consisting of function A that is inline and B that just calls A three times. If B is small relative to A, the growth of unit is 300\% and yet such inlining is very sane. For very large units consisting of small inlineable functions, however, the overall unit growth limit is needed to avoid exponential explosion of code size. Thus for smaller units, the size is increased to **−−param large−unit−insns** before applying **−−param inline−unit−growth**.

lazy−modules

Maximum number of concurrently open C++ module files when lazy loading.

inline−unit−growth

Specifies maximal overall growth of the compilation unit caused by inlining. For example, parameter value 20 limits unit growth to 1.2 times the original size. Cold functions (either marked cold via an attribute or by profile feedback) are not accounted into the unit size.

ipa−cp−unit−growth

Specifies maximal overall growth of the compilation unit caused by interprocedural constant propagation. For example, parameter value 10 limits unit growth to 1.1 times the original size.

ipa−cp−large−unit−insns

The size of translation unit that IPA−CP pass considers large.

large−stack−frame

The limit specifying large stack frames. While inlining the algorithm is trying to not grow past this limit too much.

large−stack−frame−growth

Specifies maximal growth of large stack frames caused by inlining in percents. For example, parameter value 1000 limits large stack frame growth to 11 times the original size.

max−inline−insns−recursive
max−inline−insns−recursive−auto

Specifies the maximum number of instructions an out−of−line copy of a self−recursive inline function can grow into by performing recursive inlining.

−−param max−inline−insns−recursive applies to functions declared inline. For functions not declared inline, recursive inlining happens only when **−finline−functions** (included in **−O3**) is enabled; **−−param max−inline−insns−recursive−auto** applies instead.

max−inline−recursive−depth
max−inline−recursive−depth−auto
Specifies the maximum recursion depth used for recursive inlining.

−−param max−inline−recursive−depth applies to functions declared inline. For functions not declared inline, recursive inlining happens only when **−finline−functions** (included in **−O3**) is enabled; **−−param max−inline−recursive−depth−auto** applies instead.

min−inline−recursive−probability
Recursive inlining is profitable only for function having deep recursion in average and can hurt for function having little recursion depth by increasing the prologue size or complexity of function body to other optimizers.

When profile feedback is available (see **−fprofile−generate**) the actual recursion depth can be guessed from the probability that function recurses via a given call expression. This parameter limits inlining only to call expressions whose probability exceeds the given threshold (in percents).

early−inlining−insns
Specify growth that the early inliner can make. In effect it increases the amount of inlining for code having a large abstraction penalty.

max−early−inliner−iterations
Limit of iterations of the early inliner. This basically bounds the number of nested indirect calls the early inliner can resolve. Deeper chains are still handled by late inlining.

comdat−sharing−probability
Probability (in percent) that C++ inline function with comdat visibility are shared across multiple compilation units.

modref−max−bases
modref−max−refs
modref−max−accesses
Specifies the maximal number of base pointers, references and accesses stored for a single function by mod/ref analysis.

modref−max−tests
Specifies the maxmal number of tests alias oracle can perform to disambiguate memory locations using the mod/ref information. This parameter ought to be bigger than **−−param modref−max−bases** and **−−param modref−max−refs**.

modref−max−depth
Specifies the maximum depth of DFS walk used by modref escape analysis. Setting to 0 disables the analysis completely.

modref−max−escape−points
Specifies the maximum number of escape points tracked by modref per SSA−name.

modref−max−adjustments
Specifies the maximum number the access range is enlarged during modref dataflow analysis.

profile−func−internal−id
A parameter to control whether to use function internal id in profile database lookup. If the value is 0, the compiler uses an id that is based on function assembler name and filename, which makes old profile data more tolerant to source changes such as function reordering etc.

min−vect−loop−bound
The minimum number of iterations under which loops are not vectorized when **−ftree−vectorize** is used. The number of iterations after vectorization needs to be greater than the value specified by this option to allow vectorization.

gcse−cost−distance−ratio

Scaling factor in calculation of maximum distance an expression can be moved by GCSE optimizations. This is currently supported only in the code hoisting pass. The bigger the ratio, the more aggressive code hoisting is with simple expressions, i.e., the expressions that have cost less than **gcse−unrestricted−cost**. Specifying 0 disables hoisting of simple expressions.

gcse−unrestricted−cost

Cost, roughly measured as the cost of a single typical machine instruction, at which GCSE optimizations do not constrain the distance an expression can travel. This is currently supported only in the code hoisting pass. The lesser the cost, the more aggressive code hoisting is. Specifying 0 allows all expressions to travel unrestricted distances.

max−hoist−depth

The depth of search in the dominator tree for expressions to hoist. This is used to avoid quadratic behavior in hoisting algorithm. The value of 0 does not limit on the search, but may slow down compilation of huge functions.

max−tail−merge−comparisons

The maximum amount of similar bbs to compare a bb with. This is used to avoid quadratic behavior in tree tail merging.

max−tail−merge−iterations

The maximum amount of iterations of the pass over the function. This is used to limit compilation time in tree tail merging.

store−merging−allow−unaligned

Allow the store merging pass to introduce unaligned stores if it is legal to do so.

max−stores−to−merge

The maximum number of stores to attempt to merge into wider stores in the store merging pass.

max−store−chains−to−track

The maximum number of store chains to track at the same time in the attempt to merge them into wider stores in the store merging pass.

max−stores−to−track

The maximum number of stores to track at the same time in the attemt to to merge them into wider stores in the store merging pass.

max−unrolled−insns

The maximum number of instructions that a loop may have to be unrolled. If a loop is unrolled, this parameter also determines how many times the loop code is unrolled.

max−average−unrolled−insns

The maximum number of instructions biased by probabilities of their execution that a loop may have to be unrolled. If a loop is unrolled, this parameter also determines how many times the loop code is unrolled.

max−unroll−times

The maximum number of unrollings of a single loop.

max−peeled−insns

The maximum number of instructions that a loop may have to be peeled. If a loop is peeled, this parameter also determines how many times the loop code is peeled.

max−peel−times

The maximum number of peelings of a single loop.

max−peel−branches

The maximum number of branches on the hot path through the peeled sequence.

max−completely−peeled−insns

The maximum number of insns of a completely peeled loop.

max−completely−peel−times

The maximum number of iterations of a loop to be suitable for complete peeling.

max−completely−peel−loop−nest−depth

The maximum depth of a loop nest suitable for complete peeling.

max−unswitch−insns

The maximum number of insns of an unswitched loop.

max−unswitch−depth

The maximum depth of a loop nest to be unswitched.

lim−expensive

The minimum cost of an expensive expression in the loop invariant motion.

min−loop−cond−split−prob

When FDO profile information is available, **min−loop−cond−split−prob** specifies minimum threshold for probability of semi−invariant condition statement to trigger loop split.

iv−consider−all−candidates−bound

Bound on number of candidates for induction variables, below which all candidates are considered for each use in induction variable optimizations. If there are more candidates than this, only the most relevant ones are considered to avoid quadratic time complexity.

iv−max−considered−uses

The induction variable optimizations give up on loops that contain more induction variable uses.

iv−always−prune−cand−set−bound

If the number of candidates in the set is smaller than this value, always try to remove unnecessary ivs from the set when adding a new one.

avg−loop−niter

Average number of iterations of a loop.

dse−max−object−size

Maximum size (in bytes) of objects tracked bytewise by dead store elimination. Larger values may result in larger compilation times.

dse−max−alias−queries−per−store

Maximum number of queries into the alias oracle per store. Larger values result in larger compilation times and may result in more removed dead stores.

scev−max−expr−size

Bound on size of expressions used in the scalar evolutions analyzer. Large expressions slow the analyzer.

scev−max−expr−complexity

Bound on the complexity of the expressions in the scalar evolutions analyzer. Complex expressions slow the analyzer.

max−tree−if−conversion−phi−args

Maximum number of arguments in a PHI supported by TREE if conversion unless the loop is marked with simd pragma.

vect−max−layout−candidates

The maximum number of possible vector layouts (such as permutations) to consider when optimizing to−be−vectorized code.

vect−max−version−for−alignment−checks

The maximum number of run−time checks that can be performed when doing loop versioning for alignment in the vectorizer.

vect−max−version−for−alias−checks

The maximum number of run−time checks that can be performed when doing loop versioning for alias in the vectorizer.

vect−max−peeling−for−alignment

The maximum number of loop peels to enhance access alignment for vectorizer. Value −1 means no limit.

max−iterations−to−track

The maximum number of iterations of a loop the brute−force algorithm for analysis of the number of iterations of the loop tries to evaluate.

hot−bb−count−fraction

The denominator n of fraction 1/n of the maximal execution count of a basic block in the entire program that a basic block needs to at least have in order to be considered hot. The default is 10000, which means that a basic block is considered hot if its execution count is greater than 1/10000 of the maximal execution count. 0 means that it is never considered hot. Used in non−LTO mode.

hot−bb−count−ws−permille

The number of most executed permilles, ranging from 0 to 1000, of the profiled execution of the entire program to which the execution count of a basic block must be part of in order to be considered hot. The default is 990, which means that a basic block is considered hot if its execution count contributes to the upper 990 permilles, or 99.0%, of the profiled execution of the entire program. 0 means that it is never considered hot. Used in LTO mode.

hot−bb−frequency−fraction

The denominator n of fraction 1/n of the execution frequency of the entry block of a function that a basic block of this function needs to at least have in order to be considered hot. The default is 1000, which means that a basic block is considered hot in a function if it is executed more frequently than 1/1000 of the frequency of the entry block of the function. 0 means that it is never considered hot.

unlikely−bb−count−fraction

The denominator n of fraction 1/n of the number of profiled runs of the entire program below which the execution count of a basic block must be in order for the basic block to be considered unlikely executed. The default is 20, which means that a basic block is considered unlikely executed if it is executed in fewer than 1/20, or 5%, of the runs of the program. 0 means that it is always considered unlikely executed.

max−predicted−iterations

The maximum number of loop iterations we predict statically. This is useful in cases where a function contains a single loop with known bound and another loop with unknown bound. The known number of iterations is predicted correctly, while the unknown number of iterations average to roughly 10. This means that the loop without bounds appears artificially cold relative to the other one.

builtin−expect−probability

Control the probability of the expression having the specified value. This parameter takes a percentage (i.e. 0 ... 100) as input.

builtin−string−cmp−inline−length

The maximum length of a constant string for a builtin string cmp call eligible for inlining.

align−threshold

Select fraction of the maximal frequency of executions of a basic block in a function to align the basic block.

align−loop−iterations

A loop expected to iterate at least the selected number of iterations is aligned.

tracer−dynamic−coverage
tracer−dynamic−coverage−feedback

> This value is used to limit superblock formation once the given percentage of executed instructions is covered. This limits unnecessary code size expansion.

> The **tracer−dynamic−coverage−feedback** parameter is used only when profile feedback is available. The real profiles (as opposed to statically estimated ones) are much less balanced allowing the threshold to be larger value.

tracer−max−code−growth

> Stop tail duplication once code growth has reached given percentage. This is a rather artificial limit, as most of the duplicates are eliminated later in cross jumping, so it may be set to much higher values than is the desired code growth.

tracer−min−branch−ratio

> Stop reverse growth when the reverse probability of best edge is less than this threshold (in percent).

tracer−min−branch−probability
tracer−min−branch−probability−feedback

> Stop forward growth if the best edge has probability lower than this threshold.

> Similarly to **tracer−dynamic−coverage** two parameters are provided. **tracer−min−branch−probability−feedback** is used for compilation with profile feedback and **tracer−min−branch−probability** compilation without. The value for compilation with profile feedback needs to be more conservative (higher) in order to make tracer effective.

stack−clash−protection−guard−size

> Specify the size of the operating system provided stack guard as 2 raised to *num* bytes. Higher values may reduce the number of explicit probes, but a value larger than the operating system provided guard will leave code vulnerable to stack clash style attacks.

stack−clash−protection−probe−interval

> Stack clash protection involves probing stack space as it is allocated. This param controls the maximum distance between probes into the stack as 2 raised to *num* bytes. Higher values may reduce the number of explicit probes, but a value larger than the operating system provided guard will leave code vulnerable to stack clash style attacks.

max−cse−path−length

> The maximum number of basic blocks on path that CSE considers.

max−cse−insns

> The maximum number of instructions CSE processes before flushing.

ggc−min−expand

> GCC uses a garbage collector to manage its own memory allocation. This parameter specifies the minimum percentage by which the garbage collector's heap should be allowed to expand between collections. Tuning this may improve compilation speed; it has no effect on code generation.

> The default is 30% + 70% * (RAM/1GB) with an upper bound of 100% when RAM >= 1GB. If `getrlimit` is available, the notion of "RAM" is the smallest of actual RAM and `RLIMIT_DATA` or `RLIMIT_AS`. If GCC is not able to calculate RAM on a particular platform, the lower bound of 30% is used. Setting this parameter and **ggc−min−heapsize** to zero causes a full collection to occur at every opportunity. This is extremely slow, but can be useful for debugging.

ggc−min−heapsize

> Minimum size of the garbage collector's heap before it begins bothering to collect garbage. The first collection occurs after the heap expands by **ggc−min−expand**% beyond **ggc−min−heapsize**. Again, tuning this may improve compilation speed, and has no effect on code generation.

> The default is the smaller of RAM/8, RLIMIT_RSS, or a limit that tries to ensure that

RLIMIT_DATA or RLIMIT_AS are not exceeded, but with a lower bound of 4096 (four megabytes) and an upper bound of 131072 (128 megabytes). If GCC is not able to calculate RAM on a particular platform, the lower bound is used. Setting this parameter very large effectively disables garbage collection. Setting this parameter and **ggc−min−expand** to zero causes a full collection to occur at every opportunity.

max−reload−search−insns
   The maximum number of instruction reload should look backward for equivalent register. Increasing values mean more aggressive optimization, making the compilation time increase with probably slightly better performance.

max−cselib−memory−locations
   The maximum number of memory locations cselib should take into account. Increasing values mean more aggressive optimization, making the compilation time increase with probably slightly better performance.

max−sched−ready−insns
   The maximum number of instructions ready to be issued the scheduler should consider at any given time during the first scheduling pass. Increasing values mean more thorough searches, making the compilation time increase with probably little benefit.

max−sched−region−blocks
   The maximum number of blocks in a region to be considered for interblock scheduling.

max−pipeline−region−blocks
   The maximum number of blocks in a region to be considered for pipelining in the selective scheduler.

max−sched−region−insns
   The maximum number of insns in a region to be considered for interblock scheduling.

max−pipeline−region−insns
   The maximum number of insns in a region to be considered for pipelining in the selective scheduler.

min−spec−prob
   The minimum probability (in percents) of reaching a source block for interblock speculative scheduling.

max−sched−extend−regions−iters
   The maximum number of iterations through CFG to extend regions. A value of 0 disables region extensions.

max−sched−insn−conflict−delay
   The maximum conflict delay for an insn to be considered for speculative motion.

sched−spec−prob−cutoff
   The minimal probability of speculation success (in percents), so that speculative insns are scheduled.

sched−state−edge−prob−cutoff
   The minimum probability an edge must have for the scheduler to save its state across it.

sched−mem−true−dep−cost
   Minimal distance (in CPU cycles) between store and load targeting same memory locations.

selsched−max−lookahead
   The maximum size of the lookahead window of selective scheduling. It is a depth of search for available instructions.

selsched−max−sched−times
   The maximum number of times that an instruction is scheduled during selective scheduling. This is the limit on the number of iterations through which the instruction may be pipelined.

selsched−insns−to−rename
> The maximum number of best instructions in the ready list that are considered for renaming in the selective scheduler.

sms−min−sc
> The minimum value of stage count that swing modulo scheduler generates.

max−last−value−rtl
> The maximum size measured as number of RTLs that can be recorded in an expression in combiner for a pseudo register as last known value of that register.

max−combine−insns
> The maximum number of instructions the RTL combiner tries to combine.

max−combine−search−insns
> The maximum number of instructions that the RTL combiner searches in order to find the next use of a given register definition. If this limit is reached without finding such a use, the combiner will stop trying to optimize the definition.
>
> Currently this limit only applies after certain successful combination attempts, but it could be extended to other cases in future.

integer−share−limit
> Small integer constants can use a shared data structure, reducing the compiler's memory usage and increasing its speed. This sets the maximum value of a shared integer constant.

ssp−buffer−size
> The minimum size of buffers (i.e. arrays) that receive stack smashing protection when **−fstack−protector** is used.

min−size−for−stack−sharing
> The minimum size of variables taking part in stack slot sharing when not optimizing.

max−jump−thread−duplication−stmts
> Maximum number of statements allowed in a block that needs to be duplicated when threading jumps.

max−jump−thread−paths
> The maximum number of paths to consider when searching for jump threading opportunities. When arriving at a block, incoming edges are only considered if the number of paths to be searched so far multiplied by the number of incoming edges does not exhaust the specified maximum number of paths to consider.

max−fields−for−field−sensitive
> Maximum number of fields in a structure treated in a field sensitive manner during pointer analysis.

prefetch−latency
> Estimate on average number of instructions that are executed before prefetch finishes. The distance prefetched ahead is proportional to this constant. Increasing this number may also lead to less streams being prefetched (see **simultaneous−prefetches**).

simultaneous−prefetches
> Maximum number of prefetches that can run at the same time.

l1−cache−line−size
> The size of cache line in L1 data cache, in bytes.

l1−cache−size
> The size of L1 data cache, in kilobytes.

l2−cache−size
> The size of L2 data cache, in kilobytes.

prefetch−dynamic−strides

Whether the loop array prefetch pass should issue software prefetch hints for strides that are non−constant. In some cases this may be beneficial, though the fact the stride is non−constant may make it hard to predict when there is clear benefit to issuing these hints.

Set to 1 if the prefetch hints should be issued for non−constant strides. Set to 0 if prefetch hints should be issued only for strides that are known to be constant and below **prefetch−minimum−stride**.

prefetch−minimum−stride

Minimum constant stride, in bytes, to start using prefetch hints for. If the stride is less than this threshold, prefetch hints will not be issued.

This setting is useful for processors that have hardware prefetchers, in which case there may be conflicts between the hardware prefetchers and the software prefetchers. If the hardware prefetchers have a maximum stride they can handle, it should be used here to improve the use of software prefetchers.

A value of −1 means we don't have a threshold and therefore prefetch hints can be issued for any constant stride.

This setting is only useful for strides that are known and constant.

destructive−interference−size
constructive−interference−size

The values for the C++17 variables `std::hardware_destructive_interference_size` and `std::hardware_constructive_interference_size`. The destructive interference size is the minimum recommended offset between two independent concurrently−accessed objects; the constructive interference size is the maximum recommended size of contiguous memory accessed together. Typically both will be the size of an L1 cache line for the target, in bytes. For a generic target covering a range of L1 cache line sizes, typically the constructive interference size will be the small end of the range and the destructive size will be the large end.

The destructive interference size is intended to be used for layout, and thus has ABI impact. The default value is not expected to be stable, and on some targets varies with **−mtune**, so use of this variable in a context where ABI stability is important, such as the public interface of a library, is strongly discouraged; if it is used in that context, users can stabilize the value using this option.

The constructive interference size is less sensitive, as it is typically only used in a **static_assert** to make sure that a type fits within a cache line.

See also **−Winterference−size**.

loop−interchange−max−num−stmts

The maximum number of stmts in a loop to be interchanged.

loop−interchange−stride−ratio

The minimum ratio between stride of two loops for interchange to be profitable.

min−insn−to−prefetch−ratio

The minimum ratio between the number of instructions and the number of prefetches to enable prefetching in a loop.

prefetch−min−insn−to−mem−ratio

The minimum ratio between the number of instructions and the number of memory references to enable prefetching in a loop.

use−canonical−types

Whether the compiler should use the "canonical" type system. Should always be 1, which uses a more efficient internal mechanism for comparing types in C++ and Objective−C++. However, if bugs in the canonical type system are causing compilation failures, set this value to 0 to disable

canonical types.

switch−conversion−max−branch−ratio
Switch initialization conversion refuses to create arrays that are bigger than **switch−conversion−max−branch−ratio** times the number of branches in the switch.

switch−lower−slow−alg−max−cases
Maximum number of cases for slow switch lowering algorithms to be used.

max−partial−antic−length
Maximum length of the partial antic set computed during the tree partial redundancy elimination optimization (**−ftree−pre**) when optimizing at **−O3** and above. For some sorts of source code the enhanced partial redundancy elimination optimization can run away, consuming all of the memory available on the host machine. This parameter sets a limit on the length of the sets that are computed, which prevents the runaway behavior. Setting a value of 0 for this parameter allows an unlimited set length.

rpo−vn−max−loop−depth
Maximum loop depth that is value−numbered optimistically. When the limit hits the innermost *rpo−vn−max−loop−depth* loops and the outermost loop in the loop nest are value−numbered optimistically and the remaining ones not.

sccvn−max−alias−queries−per−access
Maximum number of alias−oracle queries we perform when looking for redundancies for loads and stores. If this limit is hit the search is aborted and the load or store is not considered redundant. The number of queries is algorithmically limited to the number of stores on all paths from the load to the function entry.

ira−max−loops−num
IRA uses regional register allocation by default. If a function contains more loops than the number given by this parameter, only at most the given number of the most frequently−executed loops form regions for regional register allocation.

ira−max−conflict−table−size
Although IRA uses a sophisticated algorithm to compress the conflict table, the table can still require excessive amounts of memory for huge functions. If the conflict table for a function could be more than the size in MB given by this parameter, the register allocator instead uses a faster, simpler, and lower−quality algorithm that does not require building a pseudo−register conflict table.

ira−loop−reserved−regs
IRA can be used to evaluate more accurate register pressure in loops for decisions to move loop invariants (see **−O3**). The number of available registers reserved for some other purposes is given by this parameter. Default of the parameter is the best found from numerous experiments.

ira−consider−dup−in−all−alts
Make IRA to consider matching constraint (duplicated operand number) heavily in all available alternatives for preferred register class. If it is set as zero, it means IRA only respects the matching constraint when it's in the only available alternative with an appropriate register class. Otherwise, it means IRA will check all available alternatives for preferred register class even if it has found some choice with an appropriate register class and respect the found qualified matching constraint.

ira−simple−lra−insn−threshold
Approximate function insn number in 1K units triggering simple local RA.

lra−inheritance−ebb−probability−cutoff
LRA tries to reuse values reloaded in registers in subsequent insns. This optimization is called inheritance. EBB is used as a region to do this optimization. The parameter defines a minimal fall−through edge probability in percentage used to add BB to inheritance EBB in LRA. The default value was chosen from numerous runs of SPEC2000 on x86−64.

loop−invariant−max−bbs−in−loop

Loop invariant motion can be very expensive, both in compilation time and in amount of needed compile−time memory, with very large loops. Loops with more basic blocks than this parameter won't have loop invariant motion optimization performed on them.

loop−max−datarefs−for−datadeps

Building data dependencies is expensive for very large loops. This parameter limits the number of data references in loops that are considered for data dependence analysis. These large loops are no handled by the optimizations using loop data dependencies.

max−vartrack−size

Sets a maximum number of hash table slots to use during variable tracking dataflow analysis of any function. If this limit is exceeded with variable tracking at assignments enabled, analysis for that function is retried without it, after removing all debug insns from the function. If the limit is exceeded even without debug insns, var tracking analysis is completely disabled for the function. Setting the parameter to zero makes it unlimited.

max−vartrack−expr−depth

Sets a maximum number of recursion levels when attempting to map variable names or debug temporaries to value expressions. This trades compilation time for more complete debug information. If this is set too low, value expressions that are available and could be represented in debug information may end up not being used; setting this higher may enable the compiler to find more complex debug expressions, but compile time and memory use may grow.

max−debug−marker−count

Sets a threshold on the number of debug markers (e.g. begin stmt markers) to avoid complexity explosion at inlining or expanding to RTL. If a function has more such gimple stmts than the set limit, such stmts will be dropped from the inlined copy of a function, and from its RTL expansion.

min−nondebug−insn−uid

Use uids starting at this parameter for nondebug insns. The range below the parameter is reserved exclusively for debug insns created by **−fvar−tracking−assignments**, but debug insns may get (non−overlapping) uids above it if the reserved range is exhausted.

ipa−sra−deref−prob−threshold

IPA−SRA replaces a pointer which is known not be NULL with one or more new parameters only when the probability (in percent, relative to function entry) of it being dereferenced is higher than this parameter.

ipa−sra−ptr−growth−factor

IPA−SRA replaces a pointer to an aggregate with one or more new parameters only when their cumulative size is less or equal to **ipa−sra−ptr−growth−factor** times the size of the original pointer parameter.

ipa−sra−ptrwrap−growth−factor

Additional maximum allowed growth of total size of new parameters that ipa−sra replaces a pointer to an aggregate with, if it points to a local variable that the caller only writes to and passes it as an argument to other functions.

ipa−sra−max−replacements

Maximum pieces of an aggregate that IPA−SRA tracks. As a consequence, it is also the maximum number of replacements of a formal parameter.

sra−max−scalarization−size−Ospeed
sra−max−scalarization−size−Osize

The two Scalar Reduction of Aggregates passes (SRA and IPA−SRA) aim to replace scalar parts of aggregates with uses of independent scalar variables. These parameters control the maximum size, in storage units, of aggregate which is considered for replacement when compiling for speed (**sra−max−scalarization−size−Ospeed**)   or   size   (**sra−max−scalarization−size−Osize**)

respectively.

sra−max−propagations

The maximum number of artificial accesses that Scalar Replacement of Aggregates (SRA) will track, per one local variable, in order to facilitate copy propagation.

tm−max−aggregate−size

When making copies of thread−local variables in a transaction, this parameter specifies the size in bytes after which variables are saved with the logging functions as opposed to save/restore code sequence pairs. This option only applies when using **−fgnu−tm**.

graphite−max−nb−scop−params

To avoid exponential effects in the Graphite loop transforms, the number of parameters in a Static Control Part (SCoP) is bounded. A value of zero can be used to lift the bound. A variable whose value is unknown at compilation time and defined outside a SCoP is a parameter of the SCoP.

hardcfr−max−blocks

Disable **−fharden−control−flow−redundancy** for functions with a larger number of blocks than the specified value. Zero removes any limit.

hardcfr−max−inline−blocks

Force **−fharden−control−flow−redundancy** to use out−of−line checking for functions with a larger number of basic blocks than the specified value.

loop−block−tile−size

Loop blocking or strip mining transforms, enabled with **−floop−block** or **−floop−strip−mine**, strip mine each loop in the loop nest by a given number of iterations. The strip length can be changed using the **loop−block−tile−size** parameter.

ipa−jump−function−lookups

Specifies number of statements visited during jump function offset discovery.

ipa−cp−value−list−size

IPA−CP attempts to track all possible values and types passed to a function's parameter in order to propagate them and perform devirtualization. **ipa−cp−value−list−size** is the maximum number of values and types it stores per one formal parameter of a function.

ipa−cp−eval−threshold

IPA−CP calculates its own score of cloning profitability heuristics and performs those cloning opportunities with scores that exceed **ipa−cp−eval−threshold**.

ipa−cp−max−recursive−depth

Maximum depth of recursive cloning for self−recursive function.

ipa−cp−min−recursive−probability

Recursive cloning only when the probability of call being executed exceeds the parameter.

ipa−cp−profile−count−base

When using **−fprofile−use** option, IPA−CP will consider the measured execution count of a call graph edge at this percentage position in their histogram as the basis for its heuristics calculation.

ipa−cp−recursive−freq−factor

The number of times interprocedural copy propagation expects recursive functions to call themselves.

ipa−cp−recursion−penalty

Percentage penalty the recursive functions will receive when they are evaluated for cloning.

ipa−cp−single−call−penalty

Percentage penalty functions containing a single call to another function will receive when they are evaluated for cloning.

ipa−max−agg−items

> IPA−CP is also capable to propagate a number of scalar values passed in an aggregate. **ipa−max−agg−items** controls the maximum number of such values per one parameter.

ipa−cp−loop−hint−bonus

> When IPA−CP determines that a cloning candidate would make the number of iterations of a loop known, it adds a bonus of **ipa−cp−loop−hint−bonus** to the profitability score of the candidate.

ipa−max−loop−predicates

> The maximum number of different predicates IPA will use to describe when loops in a function have known properties.

ipa−max−aa−steps

> During its analysis of function bodies, IPA−CP employs alias analysis in order to track values pointed to by function parameters. In order not spend too much time analyzing huge functions, it gives up and consider all memory clobbered after examining **ipa−max−aa−steps** statements modifying memory.

ipa−max−switch−predicate−bounds

> Maximal number of boundary endpoints of case ranges of switch statement. For switch exceeding this limit, IPA−CP will not construct cloning cost predicate, which is used to estimate cloning benefit, for default case of the switch statement.

ipa−max−param−expr−ops

> IPA−CP will analyze conditional statement that references some function parameter to estimate benefit for cloning upon certain constant value. But if number of operations in a parameter expression exceeds **ipa−max−param−expr−ops**, the expression is treated as complicated one, and is not handled by IPA analysis.

lto−partitions

> Specify desired number of partitions produced during WHOPR compilation. The number of partitions should exceed the number of CPUs used for compilation.

lto−min−partition

> Size of minimal partition for WHOPR (in estimated instructions). This prevents expenses of splitting very small programs into too many partitions.

lto−max−partition

> Size of max partition for WHOPR (in estimated instructions). to provide an upper bound for individual size of partition. Meant to be used only with balanced partitioning.

lto−partition−locality−frequency−cutoff

> The denominator n of fraction 1/n of the execution frequency of callee to be cloned for a particular caller. Special value of 0 dictates to always clone without a cut−off.

lto−partition−locality−size−cutoff

> Size cut−off for callee including inlined calls to be cloned for a particular caller.

lto−max−locality−partition

> Maximal size of a locality partition for LTO (in estimated instructions). Value of 0 results in default value being used.

lto−max−streaming−parallelism

> Maximal number of parallel processes used for LTO streaming.

cxx−max−namespaces−for−diagnostic−help

> The maximum number of namespaces to consult for suggestions when C++ name lookup fails for an identifier.

sink−frequency−threshold

> The maximum relative execution frequency (in percents) of the target block relative to a statement's original block to allow statement sinking of a statement. Larger numbers result in more aggressive statement sinking. A small positive adjustment is applied for statements with

memory operands as those are even more profitable so sink.

max−stores−to−sink

The maximum number of conditional store pairs that can be sunk. Set to 0 if either vectorization (**−ftree−vectorize**) or if−conversion (**−ftree−loop−if−convert**) is disabled.

case−values−threshold

The smallest number of different values for which it is best to use a jump−table instead of a tree of conditional branches. If the value is 0, use the default for the machine.

jump−table−max−growth−ratio−for−size

The maximum code size growth ratio when expanding into a jump table (in percent). The parameter is used when optimizing for size.

jump−table−max−growth−ratio−for−speed

The maximum code size growth ratio when expanding into a jump table (in percent). The parameter is used when optimizing for speed.

tree−reassoc−width

Set the maximum number of instructions executed in parallel in reassociated tree. This parameter overrides target dependent heuristics used by default if has non zero value.

sched−pressure−algorithm

Choose between the two available implementations of **−fsched−pressure**. Algorithm 1 is the original implementation and is the more likely to prevent instructions from being reordered. Algorithm 2 was designed to be a compromise between the relatively conservative approach taken by algorithm 1 and the rather aggressive approach taken by the default scheduler. It relies more heavily on having a regular register file and accurate register pressure classes. See *haifa−sched.cc* in the GCC sources for more details.

The default choice depends on the target.

max−slsr−cand−scan

Set the maximum number of existing candidates that are considered when seeking a basis for a new straight−line strength reduction candidate.

asan−globals

Enable buffer overflow detection for global objects. This kind of protection is enabled by default if you are using **−fsanitize=address** option. To disable global objects protection use **−−param asan−globals=0**.

asan−stack

Enable buffer overflow detection for stack objects. This kind of protection is enabled by default when using **−fsanitize=address**. To disable stack protection use **−−param asan−stack=0** option.

asan−instrument−reads

Enable buffer overflow detection for memory reads. This kind of protection is enabled by default when using **−fsanitize=address**. To disable memory reads protection use **−−param asan−instrument−reads=0**.

asan−instrument−writes

Enable buffer overflow detection for memory writes. This kind of protection is enabled by default when using **−fsanitize=address**. To disable memory writes protection use **−−param asan−instrument−writes=0** option.

asan−memintrin

Enable detection for built−in functions. This kind of protection is enabled by default when using **−fsanitize=address**. To disable built−in functions protection use **−−param asan−memintrin=0**.

asan−use−after−return

Enable detection of use−after−return. This kind of protection is enabled by default when using the **−fsanitize=address** option. To disable it use **−−param asan−use−after−return=0**.

Note: By default the check is disabled at run time. To enable it, add `detect_stack_use_after_return=1` to the environment variable **ASAN_OPTIONS**.

asan−instrumentation−with−call−threshold
If number of memory accesses in function being instrumented is greater or equal to this number, use callbacks instead of inline checks. E.g. to disable inline code use **−−param asan−instrumentation−with−call−threshold=0**.

asan−kernel−mem−intrinsic−prefix
If nonzero, prefix calls to `memcpy`, `memset` and `memmove` with **__asan_** or **__hwasan_** for **−fsanitize=kernel−address** or **−fsanitize=kernel−hwaddress**, respectively.

hwasan−instrument−stack
Enable hwasan instrumentation of statically sized stack−allocated variables. This kind of instrumentation is enabled by default when using **−fsanitize=hwaddress** and disabled by default when using **−fsanitize=kernel−hwaddress**. To disable stack instrumentation use **−−param hwasan−instrument−stack=0**, and to enable it use **−−param hwasan−instrument−stack=1**.

hwasan−random−frame−tag
When using stack instrumentation, decide tags for stack variables using a deterministic sequence beginning at a random tag for each frame. With this parameter unset tags are chosen using the same sequence but beginning from 1. This is enabled by default for **−fsanitize=hwaddress** and unavailable for **−fsanitize=kernel−hwaddress**. To disable it use **−−param hwasan−random−frame−tag=0**.

hwasan−instrument−allocas
Enable hwasan instrumentation of dynamically sized stack−allocated variables. This kind of instrumentation is enabled by default when using **−fsanitize=hwaddress** and disabled by default when using **−fsanitize=kernel−hwaddress**. To disable instrumentation of such variables use **−−param hwasan−instrument−allocas=0**, and to enable it use **−−param hwasan−instrument−allocas=1**.

hwasan−instrument−reads
Enable hwasan checks on memory reads. Instrumentation of reads is enabled by default for both **−fsanitize=hwaddress** and **−fsanitize=kernel−hwaddress**. To disable checking memory reads use **−−param hwasan−instrument−reads=0**.

hwasan−instrument−writes
Enable hwasan checks on memory writes. Instrumentation of writes is enabled by default for both **−fsanitize=hwaddress** and **−fsanitize=kernel−hwaddress**. To disable checking memory writes use **−−param hwasan−instrument−writes=0**.

hwasan−instrument−mem−intrinsics
Enable hwasan instrumentation of builtin functions. Instrumentation of these builtin functions is enabled by default for both **−fsanitize=hwaddress** and **−fsanitize=kernel−hwaddress**. To disable instrumentation of builtin functions use **−−param hwasan−instrument−mem−intrinsics=0**.

use−after−scope−direct−emission−threshold
If the size of a local variable in bytes is smaller or equal to this number, directly poison (or unpoison) shadow memory instead of using run−time callbacks.

tsan−distinguish−volatile
Emit special instrumentation for accesses to volatiles.

tsan−instrument−func−entry−exit
Emit instrumentation calls to **__tsan_func_entry()** and **__tsan_func_exit()**.

max−fsm−thread−path−insns
Maximum number of instructions to copy when duplicating blocks on a finite state automaton jump thread path.

threader−debug

threader−debug=[none|all] Enables verbose dumping of the threader solver.

parloops−chunk−size

Chunk size of omp schedule for loops parallelized by parloops.

parloops−schedule

Schedule type of omp schedule for loops parallelized by parloops (static, dynamic, guided, auto, runtime).

parloops−min−per−thread

The minimum number of iterations per thread of an innermost parallelized loop for which the parallelized variant is preferred over the single threaded one. Note that for a parallelized loop nest the minimum number of iterations of the outermost loop per thread is two.

max−ssa−name−query−depth

Maximum depth of recursion when querying properties of SSA names in things like fold routines. One level of recursion corresponds to following a use−def chain.

max−speculative−devirt−maydefs

The maximum number of may−defs we analyze when looking for a must−def specifying the dynamic type of an object that invokes a virtual call we may be able to devirtualize speculatively.

ranger−debug

Specifies the type of debug output to be issued for ranges.

unroll−jam−min−percent

The minimum percentage of memory references that must be optimized away for the unroll−and−jam transformation to be considered profitable.

unroll−jam−max−unroll

The maximum number of times the outer loop should be unrolled by the unroll−and−jam transformation.

max−rtl−if−conversion−unpredictable−cost

Maximum permissible cost for the sequence that would be generated by the RTL if−conversion pass for a branch that is considered unpredictable.

max−variable−expansions−in−unroller

If **−fvariable−expansion−in−unroller** is used, the maximum number of times that an individual variable will be expanded during loop unrolling.

partial−inlining−entry−probability

Maximum probability of the entry BB of split region (in percent relative to entry BB of the function) to make partial inlining happen.

max−tracked−strlens

Maximum number of strings for which strlen optimization pass will track string lengths.

gcse−after−reload−partial−fraction

The threshold ratio for performing partial redundancy elimination after reload.

gcse−after−reload−critical−fraction

The threshold ratio of critical edges execution count that permit performing redundancy elimination after reload.

max−loop−header−insns

The maximum number of insns in loop header duplicated by the copy loop headers pass.

vect−epilogues−nomask

Enable loop epilogue vectorization using smaller vector size.

vect−partial−vector−usage

Controls when the loop vectorizer considers using partial vector loads and stores as an alternative to falling back to scalar code. 0 stops the vectorizer from ever using partial vector loads and

stores. 1 allows partial vector loads and stores if vectorization removes the need for the code to iterate. 2 allows partial vector loads and stores in all loops. The parameter only has an effect on targets that support partial vector loads and stores.

vect−inner−loop−cost−factor
> The maximum factor which the loop vectorizer applies to the cost of statements in an inner loop relative to the loop being vectorized. The factor applied is the maximum of the estimated number of iterations of the inner loop and this parameter. The default value of this parameter is 50.

vect−induction−float
> Enable loop vectorization of floating point inductions.

vect−force−slp
> Force the use of SLP when vectorizing, fail if not possible.

vrp−block−limit
> Maximum number of basic blocks before VRP switches to a lower memory algorithm.

vrp−sparse−threshold
> Maximum number of basic blocks before VRP uses a sparse bitmap cache.

vrp−switch−limit
> Maximum number of outgoing edges in a switch before VRP will not process it.

vrp−vector−threshold
> Maximum number of basic blocks for VRP to use a basic cache vector.

avoid−fma−max−bits
> Maximum number of bits for which we avoid creating FMAs.

fully−pipelined−fma
> Whether the target fully pipelines FMA instructions. If non−zero, reassociation considers the benefit of parallelizing FMA's multiplication part and addition part, assuming FMUL and FMA use the same units that can also do FADD.

sms−loop−average−count−threshold
> A threshold on the average loop count considered by the swing modulo scheduler.

sms−dfa−history
> The number of cycles the swing modulo scheduler considers when checking conflicts using DFA.

graphite−allow−codegen−errors
> Whether codegen errors should be ICEs when **−fchecking**.

sms−max−ii−factor
> A factor for tuning the upper bound that swing modulo scheduler uses for scheduling a loop.

lra−max−considered−reload−pseudos
> The max number of reload pseudos which are considered during spilling a non−reload pseudo.

max−pow−sqrt−depth
> Maximum depth of sqrt chains to use when synthesizing exponentiation by a real constant.

max−dse−active−local−stores
> Maximum number of active local stores in RTL dead store elimination.

asan−instrument−allocas
> Enable asan allocas/VLAs protection.

max−iterations−computation−cost
> Bound on the cost of an expression to compute the number of iterations.

max−isl−operations
> Maximum number of isl operations, 0 means unlimited.

graphite−max−arrays−per−scop

Maximum number of arrays per scop.

max−vartrack−reverse−op−size

Max. size of loc list for which reverse ops should be added.

fsm−scale−path−stmts

Scale factor to apply to the number of statements in a threading path crossing a loop backedge when comparing to **−−param=max−jump−thread−duplication−stmts**.

uninit−control−dep−attempts

Maximum number of nested calls to search for control dependencies during uninitialized variable analysis.

uninit−max−chain−len

Maximum number of predicates anded for each predicate ored in the normalized predicate chain.

uninit−max−num−chains

Maximum number of predicates ored in the normalized predicate chain.

sched−autopref−queue−depth

Hardware autoprefetcher scheduler model control flag. Number of lookahead cycles the model looks into; at ' ' only enable instruction sorting heuristic.

loop−versioning−max−inner−insns

The maximum number of instructions that an inner loop can have before the loop versioning pass considers it too big to copy.

loop−versioning−max−outer−insns

The maximum number of instructions that an outer loop can have before the loop versioning pass considers it too big to copy, discounting any instructions in inner loops that directly benefit from versioning.

ssa−name−def−chain−limit

The maximum number of SSA_NAME assignments to follow in determining a property of a variable such as its value. This limits the number of iterations or recursive calls GCC performs when optimizing certain statements or when determining their validity prior to issuing diagnostics.

store−merging−max−size

Maximum size of a single store merging region in bytes.

store−forwarding−max−distance

Maximum number of instruction distance that a small store forwarded to a larger load may stall. Value '0' disables the cost checks for the avoid−store−forwarding pass.

hash−table−verification−limit

The number of elements for which hash table verification is done for each searched element.

max−find−base−term−values

Maximum number of VALUEs handled during a single find_base_term call.

analyzer−max−enodes−per−program−point

The maximum number of exploded nodes per program point within the analyzer, before terminating analysis of that point.

analyzer−max−constraints

The maximum number of constraints per state.

analyzer−min−snodes−for−call−summary

The minimum number of supernodes within a function for the analyzer to consider summarizing its effects at call sites.

analyzer−max−enodes−for−full−dump

The maximum depth of exploded nodes that should appear in a dot dump before switching to a less verbose format.

analyzer−max−recursion−depth

The maximum number of times a callsite can appear in a call stack within the analyzer, before terminating analysis of a call that would recurse deeper.

analyzer−max−svalue−depth

The maximum depth of a symbolic value, before approximating the value as unknown.

analyzer−max−infeasible−edges

The maximum number of infeasible edges to reject before declaring a diagnostic as infeasible.

gimple−fe−computed−hot−bb−threshold

The number of executions of a basic block which is considered hot. The parameter is used only in GIMPLE FE.

analyzer−bb−explosion−factor

The maximum number of 'after supernode' exploded nodes within the analyzer per supernode, before terminating analysis.

analyzer−text−art−string−ellipsis−threshold

The number of bytes at which to ellipsize string literals in analyzer text art diagrams.

analyzer−text−art−ideal−canvas−width

The ideal width in characters of text art diagrams generated by the analyzer.

analyzer−text−art−string−ellipsis−head−len

The number of literal bytes to show at the head of a string literal in text art when ellipsizing it.

analyzer−text−art−string−ellipsis−tail−len

The number of literal bytes to show at the tail of a string literal in text art when ellipsizing it.

ranger−logical−depth

Maximum depth of logical expression evaluation ranger will look through when evaluating outgoing edge ranges.

ranger−recompute−depth

Maximum depth of instruction chains to consider for recomputation in the outgoing range calculator.

relation−block−limit

Maximum number of relations the oracle will register in a basic block.

transitive−relations−work−bound

Work bound when discovering transitive relations from existing relations.

min−pagesize

Minimum page size for warning and early break vectorization purposes.

openacc−kernels

Specify mode of OpenACC `kernels' constructs handling. With **−−param=openacc−kernels=decompose**, OpenACC `kernels' constructs are decomposed into parts, a sequence of compute constructs, each then handled individually. This is work in progress. With **−−param=openacc−kernels=parloops**, OpenACC `kernels' constructs are handled by the **parloops** pass, en bloc. This is the current default.

openacc−privatization

Control whether the **−fopt−info−omp−note** and applicable **−fdump−tree−*−details** options emit OpenACC privatization diagnostics. With **−−param=openacc−privatization=quiet**, don't diagnose. This is the current default. With **−−param=openacc−privatization=noisy**, do diagnose.

cycle−accurate−model

Specifies whether GCC should assume that the scheduling description is mostly a cycle−accurate model of the target processor the code is intended to run on, in the absence of cache misses. Nonzero means that the selected scheduling model is accurate and likely describes an in−order processor, and that scheduling should aggressively spill to try and fill any pipeline bubbles. This is the current default. Zero means the scheduling description might not be available/accurate or perhaps not applicable at all, such as for modern out−of−order processors.

The following choices of *name* are available on AArch64 targets:

aarch64−vect−compare−costs

When vectorizing, consider using multiple different approaches and use the cost model to choose the cheapest one. This includes:

* Trying both SVE and Advanced SIMD, when SVE is available.

* Trying to use 64−bit Advanced SIMD vectors for the smallest data elements, rather than using 128−bit vectors for everything.

* Trying to use "unpacked" SVE vectors for smaller elements. This includes storing smaller elements in larger containers and accessing elements with extending loads and truncating stores.

aarch64−float−recp−precision

The number of Newton iterations for calculating the reciprocal for float type. The precision of division is proportional to this param when division approximation is enabled. The default value is 1.

aarch64−double−recp−precision

The number of Newton iterations for calculating the reciprocal for double type. The precision of division is proportional to this param when division approximation is enabled. The default value is 2.

aarch64−autovec−preference

Force an ISA selection strategy for auto−vectorization.

default

Use the default heuristics.

asimd−only

Use only Advanced SIMD for auto−vectorization.

sve−only

Use only SVE for auto−vectorization.

prefer−asimd

Use both Advanced SIMD and SVE. Prefer Advanced SIMD when the costs are deemed equal.

prefer−sve

Use both Advanced SIMD and SVE. Prefer SVE when the costs are deemed equal.

aarch64−ldp−policy

Fine−grained policy for load pairs. With **−−param=aarch64−ldp−policy=default**, use the policy of the tuning structure. This is the current default. With **−−param=aarch64−ldp−policy=always**, emit ldp regardless of alignment. With **−−param=aarch64−ldp−policy=never**, do not emit ldp. With **−−param=aarch64−ldp−policy=aligned**, emit ldp only if the source pointer is aligned to at least double the alignment of the type.

aarch64−stp−policy

Fine−grained policy for store pairs. With **−−param=aarch64−stp−policy=default**, use the policy of the tuning structure. This is the current default. With

−−param=aarch64−stp−policy=always, emit stp regardless of alignment. With **−−param=aarch64−stp−policy=never**, do not emit stp. With **−−param=aarch64−stp−policy=aligned**, emit stp only if the source pointer is aligned to at least double the alignment of the type.

aarch64−ldp−alias−check−limit
> Limit on the number of alias checks performed by the AArch64 load/store pair fusion pass when attempting to form an ldp/stp. Higher values make the pass more aggressive at re−ordering loads over stores, at the expense of increased compile time.

aarch64−ldp−writeback
> Param to control which writeback opportunities we try to handle in the AArch64 load/store pair fusion pass. A value of zero disables writeback handling. One means we try to form pairs involving one or more existing individual writeback accesses where possible. A value of two means we also try to opportunistically form writeback opportunities by folding in trailing destructive updates of the base register used by a pair.

aarch64−loop−vect−issue−rate−niters
> The tuning for some AArch64 CPUs tries to take both latencies and issue rates into account when deciding whether a loop should be vectorized using SVE, vectorized using Advanced SIMD, or not vectorized at all. If this parameter is set to *n*, GCC will not use this heuristic for loops that are known to execute in fewer than *n* Advanced SIMD iterations.

aarch64−vect−unroll−limit
> The vectorizer will use available tuning information to determine whether it would be beneficial to unroll the main vectorized loop and by how much. This parameter set's the upper bound of how much the vectorizer will unroll the main loop. The default value is four.

The following choices of *name* are available on GCN targets:

gcn−preferred−vectorization−factor
> Preferred vectorization factor: **default**, **32**, **64**.

The following choices of *name* are available on i386 and x86_64 targets:

x86−stlf−window−ninsns
> Instructions number above which STFL stall penalty can be compensated.

x86−stv−max−visits
> The maximum number of use and def visits when discovering a STV chain before the discovery is aborted.

Program Instrumentation Options

GCC supports a number of command−line options that control adding run−time instrumentation to the code it normally generates. For example, one purpose of instrumentation is collect profiling statistics for use in finding program hot spots, code coverage analysis, or profile−guided optimizations. Another class of program instrumentation is adding run−time checking to detect programming errors like invalid pointer dereferences or out−of−bounds array accesses, as well as deliberately hostile attacks such as stack smashing or C++ vtable hijacking. There is also a general hook which can be used to implement other forms of tracing or function−level instrumentation for debug or program analysis purposes.

−p
−pg
> Generate extra code to write profile information suitable for the analysis program **prof** (for **−p**) or **gprof** (for **−pg**). You must use this option when compiling the source files you want data about, and you must also use it when linking.

> You can use the function attribute `no_instrument_function` to suppress profiling of individual functions when compiling with these options.

−fprofile−arcs

Add code so that program flow *arcs* are instrumented. During execution the program records how many times each branch and call is executed and how many times it is taken or returns. On targets that support constructors with priority support, profiling properly handles constructors, destructors and C++ constructors (and destructors) of classes which are used as a type of a global variable.

When the compiled program exits it saves this data to a file called *auxname.gcda* for each source file. The data may be used for profile−directed optimizations (**−fbranch−probabilities**), or for test coverage analysis (**−ftest−coverage**). Each object file's *auxname* is generated from the name of the output file, if explicitly specified and it is not the final executable, otherwise it is the basename of the source file. In both cases any suffix is removed (e.g. *foo.gcda* for input file *dir/foo.c*, or *dir/foo.gcda* for output file specified as **−o dir/foo.o**).

Note that if a command line directly links source files, the corresponding *.gcda* files will be prefixed with the unsuffixed name of the output file. E.g. `gcc a.c b.c -o binary` would generate *binary−a.gcda* and *binary−b.gcda* files.

−fcondition−coverage

Add code so that program conditions are instrumented. During execution the program records what terms in a conditional contributes to a decision, which can be used to verify that all terms in a Boolean function are tested and have an independent effect on the outcome of a decision. The result can be read with `gcov --conditions`.

−fpath−coverage

Add code so that the paths taken are tracked. During execution the program records the prime paths taken. The number of paths grows very fast with complexity, and to avoid exploding compile times GCC will give up instrumentation if the approximate number of paths exceeds the limit controlled by **−fpath−coverage−limit**. The result can be read with `gcov --prime-paths --prime-paths-lines --prime-paths-source`,

−fpath−coverage−limit=*limit*

The threshold at which point **−fpath−coverage** gives up on instrumenting a function. This limit is approximate and conservative, as GCC uses a pessimistic heuristic which slightly overcounts the running number of paths, and gives up if the threshold is reached before finding all the paths. This option is not for fine grained control over which functions to instrument − rather it is intended to limit the effect of path explosion and keep compile times reasonable. The default is *250000*.

−−coverage

This option is used to compile and link code instrumented for coverage analysis. The option is a synonym for **−fprofile−arcs −ftest−coverage** (when compiling) and **−lgcov** (when linking). See the documentation for those options for more details.

*   Compile the source files with **−fprofile−arcs** plus optimization and code generation options. For test coverage analysis, use the additional **−ftest−coverage** option. You do not need to profile every source file in a program.

*   Compile the source files additionally with **−fprofile−abs−path** to create absolute path names in the *.gcno* files. This allows **gcov** to find the correct sources in projects where compilations occur with different working directories.

*   Link your object files with **−lgcov** or **−fprofile−arcs** (the latter implies the former).

*   Run the program on a representative workload to generate the arc profile information. This may be repeated any number of times. You can run concurrent instances of your program, and provided that the file system supports locking, the data files will be correctly updated. Unless a strict ISO C dialect option is in effect, `fork` calls are detected and correctly handled without double counting.

    Moreover, an object file can be recompiled multiple times and the corresponding *.gcda* file merges as long as the source file and the compiler options are unchanged.

   * For profile−directed optimizations, compile the source files again with the same optimization and code generation options plus **−fbranch−probabilities**.

   * For test coverage analysis, use **gcov** to produce human readable information from the *.gcno* and *.gcda* files. Refer to the **gcov** documentation for further information.

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

With **−fcondition−coverage**, for each conditional in your program GCC creates a bitset and records the exercised boolean values that have an independent effect on the outcome of that expression.

With **−fpath−coverage**, GCC finds and enumerates and records the taken prime paths of each function, unless the number of paths would exceed the limit controlled by **−fpath−coverage−limit**. If the limit is exceeded the function is not instrumented as if **−fpath−coverage** was not used. A prime path is the longest sequence of unique blocks, except possibly the first and last, which is not a subpath of any other path.

−ftest−coverage
> Produce a notes file that the **gcov** code−coverage utility can use to show program coverage. Each source file's note file is called *auxname.gcno*. Refer to the **−fprofile−arcs** option above for a description of *auxname* and instructions on how to generate test coverage data. Coverage data matches the source files more closely if you do not optimize.

−fprofile−abs−path
> Automatically convert relative source file names to absolute path names in the *.gcno* files. This allows **gcov** to find the correct sources in projects where compilations occur with different working directories.

−fprofile−dir=*path*
> Set the directory to search for the profile data files in to *path*. This option affects only the profile data generated by **−fprofile−generate**, **−ftest−coverage**, **−fprofile−arcs** and used by **−fprofile−use** and **−fbranch−probabilities** and its related options. Both absolute and relative paths can be used. By default, GCC uses the current directory as *path*, thus the profile data file appears in the same directory as the object file. In order to prevent the file name clashing, if the object file name is not an absolute path, we mangle the absolute path of the *sourcename.gcda* file and use it as the file name of a *.gcda* file. See details about the file naming in **−fprofile−arcs**. See similar option **−fprofile−note**.

> When an executable is run in a massive parallel environment, it is recommended to save profile to different folders. That can be done with variables in *path* that are exported during run−time:

> **%p** process ID.

> **%q{VAR}**
>> value of environment variable *VAR*

−fprofile−generate
−fprofile−generate=*path*
> Enable options usually used for instrumenting application to produce profile useful for later recompilation with profile feedback based optimization. You must use **−fprofile−generate** both when compiling and when linking your program.

> The following options are enabled: **−fprofile−arcs**, **−fprofile−values**, **−finline−functions**, and **−fipa−bit−cp**.

> If *path* is specified, GCC looks at the *path* to find the profile feedback data files. See **−fprofile−dir**.

> To optimize the program based on the collected profile information, use **−fprofile−use**.

−fprofile−info−section
−fprofile−info−section=*name*

Register the profile information in the specified section instead of using a constructor/destructor. The section name is *name* if it is specified, otherwise the section name defaults to `.gcov_info`. A pointer to the profile information generated by **−fprofile−arcs** is placed in the specified section for each translation unit. This option disables the profile information registration through a constructor and it disables the profile information processing through a destructor. This option is not intended to be used in hosted environments such as GNU/Linux. It targets freestanding environments (for example embedded systems) with limited resources which do not support constructors/destructors or the C library file I/O.

The linker could collect the input sections in a continuous memory block and define start and end symbols. A GNU linker script example which defines a linker output section follows:

```
.gcov_info      :
{
  PROVIDE (__gcov_info_start = .);
  KEEP (*(.gcov_info))
  PROVIDE (__gcov_info_end = .);
}
```

The program could dump the profiling information registered in this linker set for example like this:

```
#include <gcov.h>
#include <stdio.h>
#include <stdlib.h>

extern const struct gcov_info *const __gcov_info_start[];
extern const struct gcov_info *const __gcov_info_end[];

static void
dump (const void *d, unsigned n, void *arg)
{
  const unsigned char *c = d;

  for (unsigned i = 0; i < n; ++i)
    printf ("%02x", c[i]);
}

static void
filename (const char *f, void *arg)
{
  __gcov_filename_to_gcfn (f, dump, arg );
}

static void *
allocate (unsigned length, void *arg)
{
  return malloc (length);
}

static void
dump_gcov_info (void)
{
  const struct gcov_info *const *info = __gcov_info_start;
  const struct gcov_info *const *end = __gcov_info_end;
```

```
                          /* Obfuscate variable to prevent compiler optimizations.  */
                          __asm__ ("" : "+r" (info));

                          while (info != end)
                          {
                            void *arg = NULL;
                            __gcov_info_to_gcda (*info, filename, dump, allocate, arg);
                            putchar ('\n');
                            ++info;
                          }
                      }

                      int
                      main (void)
                      {
                        dump_gcov_info ();
                        return 0;
                      }
```

The **merge−stream** subcommand of **gcov−tool** may be used to deserialize the data stream generated by the `__gcov_filename_to_gcfn` and `__gcov_info_to_gcda` functions and merge the profile information into *.gcda* files on the host filesystem.

−fprofile−note=*path*

If *path* is specified, GCC saves *.gcno* file into *path* location. If you combine the option with multiple source files, the *.gcno* file will be overwritten.

−fprofile−prefix−path=*path*

This option can be used in combination with **profile−generate=***profile_dir* and **profile−use=***profile_dir* to inform GCC where is the base directory of built source tree. By default *profile_dir* will contain files with mangled absolute paths of all object files in the built project. This is not desirable when directory used to build the instrumented binary differs from the directory used to build the binary optimized with profile feedback because the profile data will not be found during the optimized build. In such setups **−fprofile−prefix−path=***path* with *path* pointing to the base directory of the build can be used to strip the irrelevant part of the path and keep all file names relative to the main build directory.

−fprofile−prefix−map=*old*=*new*

When compiling files residing in directory *old*, record profiling information (with **−−coverage**) describing them as if the files resided in directory *new* instead. See also **−ffile−prefix−map** and **−fcanon−prefix−map**.

−fprofile−update=*method*

Alter the update method for an application instrumented for profile feedback based optimization. The *method* argument should be one of **single**, **atomic** or **prefer−atomic**. The first one is useful for single−threaded applications, while the second one prevents profile corruption by emitting thread−safe code.

Warning: When an application does not properly join all threads (or creates an detached thread), a profile file can be still corrupted.

Using **prefer−atomic** would be transformed either to **atomic**, when supported by a target, or to **single** otherwise. The GCC driver automatically selects **prefer−atomic** when **−pthread** is present in the command line, otherwise the default method is **single**.

If **atomic** is selected, then the profile information is updated using atomic operations on a best−effort basis. Ideally, the profile information is updated through atomic operations in hardware. If the target platform does not support the required atomic operations in hardware, however, *libatomic* is available,

then the profile information is updated through calls to *libatomic*. If the target platform neither supports the required atomic operations in hardware nor *libatomic*, then the profile information is not atomically updated and a warning is issued. In this case, the obtained profiling information may be corrupt for multi−threaded applications.

For performance reasons, if 64−bit counters are used for the profiling information and the target platform only supports 32−bit atomic operations in hardware, then the performance critical profiling updates are done using two 32−bit atomic operations for each counter update. If a signal interrupts these two operations updating a counter, then the profiling information may be in an inconsistent state.

−fprofile−filter−files=*regex*
Instrument only functions from files whose name matches any of the regular expressions (separated by semi−colons).

For example, **−fprofile−filter−files=main\.c;module.*\.c** will instrument only *main.c* and all C files starting with 'module'.

−fprofile−exclude−files=*regex*
Instrument only functions from files whose name does not match any of the regular expressions (separated by semi−colons).

For example, **−fprofile−exclude−files=/usr/.*** will prevent instrumentation of all files that are located in the */usr/* folder.

−fprofile−reproducible=[multithreaded|parallel−runs|serial]
Control level of reproducibility of profile gathered by −fprofile−generate. This makes it possible to rebuild program with same outcome which is useful, for example, for distribution packages.

With **−fprofile−reproducible=serial** the profile gathered by **−fprofile−generate** is reproducible provided the trained program behaves the same at each invocation of the train run, it is not multi−threaded and profile data streaming is always done in the same order. Note that profile streaming happens at the end of program run but also before fork function is invoked.

Note that it is quite common that execution counts of some part of programs depends, for example, on length of temporary file names or memory space randomization (that may affect hash−table collision rate). Such non−reproducible part of programs may be annotated by no_instrument_function function attribute. **gcov−dump** with **−l** can be used to dump gathered data and verify that they are indeed reproducible.

With **−fprofile−reproducible=parallel−runs** collected profile stays reproducible regardless the order of streaming of the data into gcda files. This setting makes it possible to run multiple instances of instrumented program in parallel (such as with make −j). This reduces quality of gathered data, in particular of indirect call profiling.

−fsanitize=address
Enable AddressSanitizer, a fast memory error detector. Memory access instructions are instrumented to detect out−of−bounds and use−after−free bugs. The option enables **−fsanitize−address−use−after−scope**. See **<https://github.com/google/sanitizers/wiki/AddressSanitizer>** for more details. The run−time behavior can be influenced using the **ASAN_OPTIONS** environment variable. When set to help=1, the available options are shown at startup of the instrumented program. See **<https://github.com/google/sanitizers/wiki/AddressSanitizerFlags#run−time−flags>** for a list of supported options. The option cannot be combined with **−fsanitize=thread** or **−fsanitize=hwaddress**. Note that the only targets **−fsanitize=hwaddress** is currently supported on are x86−64 (only with −mlam=u48 or −mlam=u57 options) and AArch64, in both cases only in ABIs with 64−bit pointers.

When compiling with **−fsanitize=address**, you should also use **−g** to produce more meaningful output. To get more accurate stack traces, it is possible to use options such as **−O0**, **−O1**, or **−Og** (which, for instance, prevent most function inlining), **−fno−optimize−sibling−calls** (which prevents

optimizing sibling and tail recursive calls; this option is implicit for **−O0**, **−O1**, or **−Og**), or **−fno−ipa−icf** (which disables Identical Code Folding for functions). Using **−fno−omit−frame−pointer** also improves stack traces. Since multiple runs of the program may yield backtraces with different addresses due to ASLR (Address Space Layout Randomization), it may be desirable to turn ASLR off. On Linux, this can be achieved with **setarch `uname −m` −R ./prog**.

−fsanitize=kernel−address
> Enable AddressSanitizer for Linux kernel. See **<https://github.com/google/kernel−sanitizers>** for more details.

−fsanitize=hwaddress
> Enable Hardware−assisted AddressSanitizer, which uses a hardware ability to ignore the top byte of a pointer to allow the detection of memory errors with a low memory overhead. Memory access instructions are instrumented to detect out−of−bounds and use−after−free bugs. The option enables **−fsanitize−address−use−after−scope**. See **<https://clang.llvm.org/docs/HardwareAssistedAddressSanitizerDesign.html>** for more details. The run−time behavior can be influenced using the **HWASAN_OPTIONS** environment variable. When set to `help=1`, the available options are shown at startup of the instrumented program. The option cannot be combined with **−fsanitize=thread** or **−fsanitize=address**, and is currently only available on AArch64.

−fsanitize=kernel−hwaddress
> Enable Hardware−assisted AddressSanitizer for compilation of the Linux kernel. Similar to **−fsanitize=kernel−address** but using an alternate instrumentation method, and similar to **−fsanitize=hwaddress** but with instrumentation differences necessary for compiling the Linux kernel. These differences are to avoid hwasan library initialization calls and to account for the stack pointer having a different value in its top byte.

> *Note:* This option has different defaults to the **−fsanitize=hwaddress**. Instrumenting the stack and alloca calls are not on by default but are still possible by specifying the command−line options **−−param hwasan−instrument−stack=1** and **−−param hwasan−instrument−allocas=1** respectively. Using a random frame tag is not implemented for kernel instrumentation.

−fsanitize=pointer−compare
> Instrument comparison operation (<, <=, >, >=) with pointer operands. The option must be combined with either **−fsanitize=kernel−address** or **−fsanitize=address** The option cannot be combined with **−fsanitize=thread**. Note: By default the check is disabled at run time. To enable it, add `detect_invalid_pointer_pairs=2` to the environment variable **ASAN_OPTIONS**. Using `detect_invalid_pointer_pairs=1` detects invalid operation only when both pointers are non−null.

−fsanitize=pointer−subtract
> Instrument subtraction with pointer operands. The option must be combined with either **−fsanitize=kernel−address** or **−fsanitize=address** The option cannot be combined with **−fsanitize=thread**. Note: By default the check is disabled at run time. To enable it, add `detect_invalid_pointer_pairs=2` to the environment variable **ASAN_OPTIONS**. Using `detect_invalid_pointer_pairs=1` detects invalid operation only when both pointers are non−null.

−fsanitize=shadow−call−stack
> Enable ShadowCallStack, a security enhancement mechanism used to protect programs against return address overwrites (e.g. stack buffer overflows.) It works by saving a function's return address to a separately allocated shadow call stack in the function prologue and restoring the return address from the shadow call stack in the function epilogue. Instrumentation only occurs in functions that need to save the return address to the stack.

> Currently it only supports the aarch64 platform. It is specifically designed for linux kernels that enable the CONFIG_SHADOW_CALL_STACK option. For the user space programs, runtime support is not currently provided in libc and libgcc. Users who want to use this feature in user space

need to provide their own support for the runtime. It should be noted that this may cause the ABI rules to be broken.

On aarch64, the instrumentation makes use of the platform register x18. This generally means that any code that may run on the same thread as code compiled with ShadowCallStack must be compiled with the flag **−ffixed−x18**, otherwise functions compiled without **−ffixed−x18** might clobber x18 and so corrupt the shadow stack pointer.

Also, because there is no userspace runtime support, code compiled with ShadowCallStack cannot use exception handling. Use **−fno−exceptions** to turn off exceptions.

See <**https://clang.llvm.org/docs/ShadowCallStack.html**> for more details.

−fsanitize=thread
> Enable ThreadSanitizer, a fast data race detector. Memory access instructions are instrumented to detect data race bugs. See <**https://github.com/google/sanitizers/wiki#threadsanitizer**> for more details. The run−time behavior can be influenced using the **TSAN_OPTIONS** environment variable; see <**https://github.com/google/sanitizers/wiki/ThreadSanitizerFlags**> for a list of supported options. The option cannot be combined with **−fsanitize=address**, **−fsanitize=leak**.

> When compiling with **−fsanitize=thread**, you should also use **−g** to produce more meaningful output.

> Note that sanitized atomic builtins cannot throw exceptions when operating on invalid memory addresses with non−call exceptions (**−fnon−call−exceptions**).

−fsanitize=leak
> Enable LeakSanitizer, a memory leak detector. This option only matters for linking of executables. The executable is linked against a library that overrides malloc and other allocator functions. See <**https://github.com/google/sanitizers/wiki/AddressSanitizerLeakSanitizer**> for more details. The run−time behavior can be influenced using the **LSAN_OPTIONS** environment variable. The option cannot be combined with **−fsanitize=thread**.

−fsanitize=undefined
> Enable UndefinedBehaviorSanitizer, a fast undefined behavior detector. Various computations are instrumented to detect undefined behavior at runtime. See <**https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html**> for more details. The run−time behavior can be influenced using the **UBSAN_OPTIONS** environment variable. Current suboptions are:

> **−fsanitize=shift**
>> This option enables checking that the result of a shift operation is not undefined. Note that what exactly is considered undefined differs slightly between C and C++, as well as between ISO C90 and C99, etc. This option has two suboptions, **−fsanitize=shift−base** and **−fsanitize=shift−exponent**.

> **−fsanitize=shift−exponent**
>> This option enables checking that the second argument of a shift operation is not negative and is smaller than the precision of the promoted first argument.

> **−fsanitize=shift−base**
>> If the second argument of a shift operation is within range, check that the result of a shift operation is not undefined. Note that what exactly is considered undefined differs slightly between C and C++, as well as between ISO C90 and C99, etc.

> **−fsanitize=integer−divide−by−zero**
>> Detect integer division by zero.

> **−fsanitize=unreachable**
>> With this option, the compiler turns the __builtin_unreachable call into a diagnostics message call instead. When reaching the __builtin_unreachable call, the behavior is undefined.

−fsanitize=vla−bound

This option instructs the compiler to check that the size of a variable length array is positive.

−fsanitize=null

This option enables pointer checking. Particularly, the application built with this option turned on will issue an error message when it tries to dereference a NULL pointer, or if a reference (possibly an rvalue reference) is bound to a NULL pointer, or if a method is invoked on an object pointed by a NULL pointer.

−fsanitize=return

This option enables return statement checking. Programs built with this option turned on will issue an error message when the end of a non−void function is reached without actually returning a value. This option works in C++ only.

−fsanitize=signed−integer−overflow

This option enables signed integer overflow checking. We check that the result of +, *, and both unary and binary − does not overflow in the signed arithmetics. This also detects `INT_MIN / −1` signed division. Note, integer promotion rules must be taken into account. That is, the following is not an overflow:

```
signed char a = SCHAR_MAX;
a++;
```

−fsanitize=bounds

This option enables instrumentation of array bounds. Various out of bounds accesses are detected. Flexible array members, flexible array member−like arrays, and initializers of variables with static storage are not instrumented, with the exception of flexible array member−like arrays for which `−fstrict−flex−arrays` or `−fstrict−flex−arrays=` options or `strict_flex_array` attributes say they shouldn't be treated like flexible array member−like arrays.

−fsanitize=bounds−strict

This option enables strict instrumentation of array bounds. Most out of bounds accesses are detected, including flexible array member−like arrays. Initializers of variables with static storage are not instrumented.

−fsanitize=alignment

This option enables checking of alignment of pointers when they are dereferenced, or when a reference is bound to insufficiently aligned target, or when a method or constructor is invoked on insufficiently aligned object.

−fsanitize=object−size

This option enables instrumentation of memory references using the `__builtin_dynamic_object_size` function. Various out of bounds pointer accesses are detected.

−fsanitize=float−divide−by−zero

Detect floating−point division by zero. Unlike other similar options, **−fsanitize=float−divide−by−zero** is not enabled by **−fsanitize=undefined**, since floating−point division by zero can be a legitimate way of obtaining infinities and NaNs.

−fsanitize=float−cast−overflow

This option enables floating−point type to integer conversion checking. We check that the result of the conversion does not overflow. Unlike other similar options, **−fsanitize=float−cast−overflow** is not enabled by **−fsanitize=undefined**. This option does not work well with `FE_INVALID` exceptions enabled.

−fsanitize=nonnull−attribute

This option enables instrumentation of calls, checking whether null values are not passed to arguments marked as requiring a non−null value by the `nonnull` function attribute.

−fsanitize=returns−nonnull−attribute

   This option enables instrumentation of return statements in functions marked with `returns_nonnull` function attribute, to detect returning of null values from such functions.

−fsanitize=bool

   This option enables instrumentation of loads from bool. If a value other than 0/1 is loaded, a run−time error is issued.

−fsanitize=enum

   This option enables instrumentation of loads from an enum type. If a value outside the range of values for the enum type is loaded, a run−time error is issued.

−fsanitize=vptr

   This option enables instrumentation of C++ member function calls, member accesses and some conversions between pointers to base and derived classes, to verify the referenced object has the correct dynamic type.

−fsanitize=pointer−overflow

   This option enables instrumentation of pointer arithmetics. If the pointer arithmetics overflows, a run−time error is issued.

−fsanitize=builtin

   This option enables instrumentation of arguments to selected builtin functions. If an invalid value is passed to such arguments, a run−time error is issued. E.g. passing 0 as the argument to `__builtin_ctz` or `__builtin_clz` invokes undefined behavior and is diagnosed by this option.

Note that sanitizers tend to increase the rate of false positive warnings, most notably those around **−Wmaybe−uninitialized**. We recommend against combining **−Werror** and [the use of] sanitizers.

While **−ftrapv** causes traps for signed overflows to be emitted, **−fsanitize=undefined** gives a diagnostic message. This currently works only for the C family of languages.

−fno−sanitize=all

   This option disables all previously enabled sanitizers. **−fsanitize=all** is not allowed, as some sanitizers cannot be used together.

−fasan−shadow−offset=_number_

   This option forces GCC to use custom shadow offset in AddressSanitizer checks. It is useful for experimenting with different shadow memory layouts in Kernel AddressSanitizer.

−fsanitize−sections=_s1_,_s2_,_..._

   Sanitize global variables in selected user−defined sections. _si_ may contain wildcards.

−fsanitize−recover[**=**_opts_]

   −fsanitize−recover= controls error recovery mode for sanitizers mentioned in comma−separated list of _opts_. Enabling this option for a sanitizer component causes it to attempt to continue running the program as if no error happened. This means multiple runtime errors can be reported in a single program run, and the exit code of the program may indicate success even when errors have been reported. The **−fno−sanitize−recover=** option can be used to alter this behavior: only the first detected error is reported and program then exits with a non−zero exit code.

   Currently this feature only works for **−fsanitize=undefined** (and its suboptions except for **−fsanitize=unreachable** and **−fsanitize=return**), **−fsanitize=float−cast−overflow**, **−fsanitize=float−divide−by−zero**, **−fsanitize=bounds−strict**, **−fsanitize=kernel−address** and **−fsanitize=address**. For these sanitizers error recovery is turned on by default, except **−fsanitize=address**, for which this feature is experimental. **−fsanitize−recover=all** and **−fno−sanitize−recover=all** is also accepted, the former enables recovery for all sanitizers that support it, the latter disables recovery for all sanitizers that support it.

   Even if a recovery mode is turned on the compiler side, it needs to be also enabled on the runtime library side, otherwise the failures are still fatal. The runtime library defaults to `halt_on_error=0`

for ThreadSanitizer and UndefinedBehaviorSanitizer, while default value for AddressSanitizer is `halt_on_error=1`. This can be overridden through setting the `halt_on_error` flag in the corresponding environment variable.

Syntax without an explicit *opts* parameter is deprecated. It is equivalent to specifying an *opts* list of:

> `undefined,float-cast-overflow,float-divide-by-zero,bounds-strict`

−fsanitize−address−use−after−scope
> Enable sanitization of local variables to detect use−after−scope bugs. The option sets **−fstack−reuse** to **none**.

−fsanitize−trap[**=***opts*]
> The **−fsanitize−trap=** option instructs the compiler to report for sanitizers mentioned in comma−separated list of *opts* undefined behavior using `__builtin_trap` rather than a `libubsan` library routine. If this option is enabled for certain sanitizer, it takes precedence over the **−fsanitizer−recover=** for that sanitizer, `__builtin_trap` will be emitted and be fatal regardless of whether recovery is enabled or disabled using **−fsanitize−recover=**.

> The advantage of this is that the `libubsan` library is not needed and is not linked in, so this is usable even in freestanding environments.

> Currently this feature works with **−fsanitize=undefined** (and its suboptions except for **−fsanitize=vptr**), **−fsanitize=float−cast−overflow**, **−fsanitize=float−divide−by−zero** and **−fsanitize=bounds−strict**. `-fsanitize-trap=all` can be also specified, which enables it for `undefined` suboptions, **−fsanitize=float−cast−overflow**, **−fsanitize=float−divide−by−zero** and **−fsanitize=bounds−strict**. If `-fsanitize-trap=undefined` or `-fsanitize-trap=all` is used and `-fsanitize=vptr` is enabled on the command line, the instrumentation is silently ignored as the instrumentation always needs `libubsan` support, **−fsanitize−trap=vptr** is not allowed.

−fsanitize−undefined−trap−on−error
> The **−fsanitize−undefined−trap−on−error** option is deprecated equivalent of **−fsanitize−trap=all**.

−fsanitize−coverage=trace−pc
> Enable coverage−guided fuzzing code instrumentation. Inserts a call to `__sanitizer_cov_trace_pc` into every basic block.

−fsanitize−coverage=trace−cmp
> Enable dataflow guided fuzzing code instrumentation. Inserts a call to `__sanitizer_cov_trace_cmp1`, `__sanitizer_cov_trace_cmp2`, `__sanitizer_cov_trace_cmp4` or `__sanitizer_cov_trace_cmp8` for integral comparison with both operands variable or `__sanitizer_cov_trace_const_cmp1`, `__sanitizer_cov_trace_const_cmp2`, `__sanitizer_cov_trace_const_cmp4` or `__sanitizer_cov_trace_const_cmp8` for integral comparison with one operand constant, `__sanitizer_cov_trace_cmpf` or `__sanitizer_cov_trace_cmpd` for float or double comparisons and `__sanitizer_cov_trace_switch` for switch statements.

−fcf−protection=[full|branch|return|none|check]
−fcf−protection
> Enable code instrumentation to increase program security by checking that target addresses of control−flow transfer instructions (such as indirect function call, function return, indirect jump) are valid. This prevents diverting the flow of control to an unexpected target. This is intended to protect against such threats as Return−oriented Programming (ROP), and similarly call/jmp−oriented programming (COP/JOP).

> The **−fcf−protection=** keywords are interpreted as follows.

> The value `branch` tells the compiler to implement checking of validity of control−flow transfer at the point of indirect branch instructions, i.e. call/jmp instructions.

> The value `return` implements checking of validity at the point of returning from a function.

The value `full` is an alias for specifying both `branch` and `return`.

The value `check` is used for the final link with link−time optimization (LTO). An error is issued if LTO object files are compiled with different **−fcf−protection** values. The value `check` is ignored at the compile time.

The value `none` turns off instrumentation.

−fcf−protection is an alias for **−fcf−protection=full**. To override a previous **−fcf−protection** option on the command line, add **−fcf−protection=none** and then **−fcf−protection=***kind*.

The macro `__CET__` is defined when **−fcf−protection** is used. The first bit of `__CET__` is set to 1 for the value `branch` and the second bit of `__CET__` is set to 1 for the `return`.

You can also use the `nocf_check` attribute to identify which functions and calls should be skipped from instrumentation.

Currently the x86 GNU/Linux target provides an implementation based on Intel Control−flow Enforcement Technology (CET) which works for i686 processor or newer.

−fharden−compares

For every logical test that survives gimple optimizations and is *not* the condition in a conditional branch (for example, conditions tested for conditional moves, or to store in boolean variables), emit extra code to compute and verify the reversed condition, and to call `__builtin_trap` if the results do not match. Use with **−fharden−conditional−branches** to cover all conditionals.

−fharden−conditional−branches

For every non−vectorized conditional branch that survives gimple optimizations, emit extra code to compute and verify the reversed condition, and to call `__builtin_trap` if the result is unexpected. Use with **−fharden−compares** to cover all conditionals.

−fharden−control−flow−redundancy

Emit extra code to set booleans when entering basic blocks, and to verify and trap, at function exits, when the booleans do not form an execution path that is compatible with the control flow graph.

Verification takes place before returns, before mandatory tail calls (see below) and, optionally, before escaping exceptions with **−fhardcfr−check−exceptions**, before returning calls with **−fhardcfr−check−returning−calls**, and before noreturn calls with **−fhardcfr−check−noreturn−calls**). Tuning options **−−param hardcfr−max−blocks** and **−−param hardcfr−max−inline−blocks** are available.

Tail call optimization takes place too late to affect control flow redundancy, but calls annotated as mandatory tail calls by language front−ends, and any calls marked early enough as potential tail calls would also have verification issued before the call, but these possibilities are merely theoretical, as these conditions can only be met when using custom compiler plugins.

−fhardcfr−skip−leaf

Disable **−fharden−control−flow−redundancy** in leaf functions.

−fhardcfr−check−exceptions

When **−fharden−control−flow−redundancy** is active, check the recorded execution path against the control flow graph at exception escape points, as if the function body was wrapped with a cleanup handler that performed the check and reraised. This option is enabled by default; use **−fno−hardcfr−check−exceptions** to disable it.

−fhardcfr−check−returning−calls

When **−fharden−control−flow−redundancy** is active, check the recorded execution path against the control flow graph before any function call immediately followed by a return of its result, if any, so as to not prevent tail−call optimization, whether or not it is ultimately optimized to a tail call.

This option is enabled by default whenever sibling call optimizations are enabled (see **−foptimize−sibling−calls**), but it can be enabled (or disabled, using its negated form) explicitly, regardless of the optimizations.

−fhardcfr−check−noreturn−calls=[always|no−xthrow|nothrow|never]

When **−fharden−control−flow−redundancy** is active, check the recorded execution path against the control flow graph before `noreturn` calls, either all of them (**always**), those that aren't expected to return control to the caller through an exception (**no−xthrow**, the default), those that may not return control to the caller through an exception either (**nothrow**), or none of them (**never**).

Checking before a `noreturn` function that may return control to the caller through an exception may cause checking to be performed more than once, if the exception is caught in the caller, whether by a handler or a cleanup. When **−fhardcfr−check−exceptions** is also enabled, the compiler will avoid associating a `noreturn` call with the implicitly−added cleanup handler, since it would be redundant with the check performed before the call, but other handlers or cleanups in the function, if activated, will modify the recorded execution path and check it again when another checkpoint is hit. The checkpoint may even be another `noreturn` call, so checking may end up performed multiple times.

Various optimizers may cause calls to be marked as `noreturn` and/or `nothrow`, even in the absence of the corresponding attributes, which may affect the placement of checks before calls, as well as the addition of implicit cleanup handlers for them. This unpredictability, and the fact that raising and reraising exceptions frequently amounts to implicitly calling `noreturn` functions, have made **no−xthrow** the default setting for this option: it excludes from the `noreturn` treatment only internal functions used to (re)raise exceptions, that are not affected by these optimizations.

−fhardened

Enable a set of flags for C and C++ that improve the security of the generated code without affecting its ABI. The precise flags enabled may change between major releases of GCC, but are currently:

−D_FORTIFY_SOURCE=3 −D_GLIBCXX_ASSERTIONS −ftrivial−auto−var−init=zero −fPIE −pie −Wl,−z,relro,−z,now −fstack−protector−strong −fstack−clash−protection −fcf−protection=full (x86 GNU/Linux only)

The list of options enabled by **−fhardened** can be generated using the **−−help=hardened** option.

When the system glibc is older than 2.35, **−D_FORTIFY_SOURCE=2** is used instead.

This option is intended to be used in production builds, not merely in debug builds.

Currently, **−fhardened** is only supported on GNU/Linux targets.

−fhardened only enables a particular option if it wasn't already specified anywhere on the command line. For instance, **−fhardened −fstack−protector** will only enable **−fstack−protector**, but not **−fstack−protector−strong**.

−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 or equal to 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. Only variables that are actually allocated on the stack are considered, optimized away variables or variables allocated in registers don't count.

−fstack−protector−all

Like **−fstack−protector** except that all functions are protected.

−fstack−protector−strong

Like **−fstack−protector** but includes additional functions to be protected −−− those that have local array definitions, or have references to local frame addresses. Only variables that are actually allocated on the stack are considered, optimized away variables or variables allocated in registers don't count.

−fstack−protector−explicit

Like **−fstack−protector** but only protects those functions which have the `stack_protect` attribute.

−fstack−check

Generate code to verify that you do not go beyond the boundary of the stack. You should specify this flag if you are running in an environment with multiple threads, but you only rarely need to specify it in a single−threaded environment since stack overflow is automatically detected on nearly all systems if there is only one stack.

Note that this switch does not actually cause checking to be done; the operating system or the language runtime must do that. The switch causes generation of code to ensure that they see the stack being extended.

You can additionally specify a string parameter: **no** means no checking, **generic** means force the use of old−style checking, **specific** means use the best checking method and is equivalent to bare **−fstack−check**.

Old−style checking is a generic mechanism that requires no specific target support in the compiler but comes with the following drawbacks:

1. Modified allocation strategy for large objects: they are always allocated dynamically if their size exceeds a fixed threshold. Note this may change the semantics of some code.

2. Fixed limit on the size of the static frame of functions: when it is topped by a particular function, stack checking is not reliable and a warning is issued by the compiler.

3. Inefficiency: because of both the modified allocation strategy and the generic implementation, code performance is hampered.

Note that old−style stack checking is also the fallback method for **specific** if no target support has been added in the compiler.

−fstack−check= is designed for Ada's needs to detect infinite recursion and stack overflows. **specific** is an excellent choice when compiling Ada code. It is not generally sufficient to protect against stack−clash attacks. To protect against those you want **−fstack−clash−protection**.

−fstack−clash−protection

Generate code to prevent stack clash style attacks. When this option is enabled, the compiler will only allocate one page of stack space at a time and each page is accessed immediately after allocation. Thus, it prevents allocations from jumping over any stack guard page provided by the operating system.

Most targets do not fully support stack clash protection. However, on those targets **−fstack−clash−protection** will protect dynamic stack allocations. **−fstack−clash−protection** may also provide limited protection for static stack allocations if the target supports **−fstack−check=specific**.

−fstack−limit−register=_reg_
−fstack−limit−symbol=_sym_
−fno−stack−limit

Generate code to ensure that the stack does not grow beyond a certain value, either the value of a register or the address of a symbol. If a larger stack is required, a signal is raised at run time. For most targets, the signal is raised before the stack overruns the boundary, so it is possible to catch the signal without taking special precautions.

For instance, if the stack starts at absolute address **0x80000000** and grows downwards, you can use the flags **−fstack−limit−symbol=__stack_limit** and **−Wl,−−defsym,__stack_limit=0x7ffe0000** to enforce a stack limit of 128KB. Note that this may only work with the GNU linker.

You can locally override stack limit checking by using the `no_stack_limit` function attribute.

−fsplit−stack

Generate code to automatically split the stack before it overflows. The resulting program has a discontiguous stack which can only overflow if the program is unable to allocate any more memory. This is most useful when running threaded programs, as it is no longer necessary to calculate a good

stack size to use for each thread. This is currently only implemented for the x86 targets running GNU/Linux.

When code compiled with **−fsplit−stack** calls code compiled without **−fsplit−stack**, there may not be much stack space available for the latter code to run. If compiling all code, including library code, with **−fsplit−stack** is not an option, then the linker can fix up these calls so that the code compiled without **−fsplit−stack** always has a large stack. Support for this is implemented in the gold linker in GNU binutils release 2.21 and later.

−fstrub=disable
> Disable stack scrubbing entirely, ignoring any `strub` attributes. See

−fstrub=strict
> Functions default to `strub` mode `disabled`, and apply **strict**ly the restriction that only functions associated with `strub-callable` modes (`at-calls`, `callable` and `always_inline internal`) are `callable` by functions with `strub−enabled` modes (`at-calls` and `internal`).

−fstrub=relaxed
> Restore the default stack scrub (`strub`) setting, namely, `strub` is only enabled as required by `strub` attributes associated with function and data types. `Relaxed` means that strub contexts are only prevented from calling functions explicitly associated with `strub` mode `disabled`. This option is only useful to override other **−fstrub=*** options that precede it in the command line.

−fstrub=at−calls
> Enable `at-calls strub` mode where viable. The primary use of this option is for testing. It exercises the `strub` machinery in scenarios strictly local to a translation unit. This `strub` mode modifies function interfaces, so any function that is visible to other translation units, or that has its address taken, will *not* be affected by this option. Optimization options may also affect viability. See the `strub` attribute documentation for details on viability and eligibility requirements.

−fstrub=internal
> Enable `internal strub` mode where viable. The primary use of this option is for testing. This option is intended to exercise thoroughly parts of the `strub` machinery that implement the less efficient, but interface−preserving `strub` mode. Functions that would not be affected by this option are quite uncommon.

−fstrub=all
> Enable some `strub` mode where viable. When both strub modes are viable, `at-calls` is preferred. **−fdump−ipa−strubm** adds function attributes that tell which mode was selected for each function. The primary use of this option is for testing, to exercise thoroughly the `strub` machinery.

−fvtable−verify=[std|preinit|none]
> This option is only available when compiling C++ code. It turns on (or off, if using **−fvtable−verify=none**) the security feature that verifies at run time, for every virtual call, that the vtable pointer through which the call is made is valid for the type of the object, and has not been corrupted or overwritten. If an invalid vtable pointer is detected at run time, an error is reported and execution of the program is immediately halted.
>
> This option causes run−time data structures to be built at program startup, which are used for verifying the vtable pointers. The options **std** and **preinit** control the timing of when these data structures are built. In both cases the data structures are built before execution reaches `main`. Using **−fvtable−verify=std** causes the data structures to be built after shared libraries have been loaded and initialized. **−fvtable−verify=preinit** causes them to be built before shared libraries have been loaded and initialized.
>
> If this option appears multiple times in the command line with different values specified, **none** takes highest priority over both **std** and **preinit**; **preinit** takes priority over **std**.

−fvtv−debug

When used in conjunction with **−fvtable−verify=std** or **−fvtable−verify=preinit**, causes debug versions of the runtime functions for the vtable verification feature to be called. This flag also causes the compiler to log information about which vtable pointers it finds for each class. This information is written to a file named *vtv_set_ptr_data.log* in the directory named by the environment variable **VTV_LOGS_DIR** if that is defined or the current working directory otherwise.

Note: This feature *appends* data to the log file. If you want a fresh log file, be sure to delete any existing one.

−fvtv−counts

This is a debugging flag. When used in conjunction with **−fvtable−verify=std** or **−fvtable−verify=preinit**, this causes the compiler to keep track of the total number of virtual calls it encounters and the number of verifications it inserts. It also counts the number of calls to certain run−time library functions that it inserts and logs this information for each compilation unit. The compiler writes this information to a file named *vtv_count_data.log* in the directory named by the environment variable **VTV_LOGS_DIR** if that is defined or the current working directory otherwise. It also counts the size of the vtable pointer sets for each class, and writes this information to *vtv_class_set_sizes.log* in the same directory.

Note: This feature *appends* data to the log files. To get fresh log files, be sure to delete any existing ones.

−finstrument−functions

Generate instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions are called with the address of the current function and its call site. (On some platforms, `__builtin_return_address` does not work beyond the current function, so the call site information may not be available to the profiling functions otherwise.)

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

The first argument is the address of the start of the current function, which may be looked up exactly in the symbol table.

This instrumentation is also done for functions expanded inline in other functions. The profiling calls indicate where, conceptually, the inline function is entered and exited. This means that addressable versions of such functions must be available. If all your uses of a function are expanded inline, this may mean an additional expansion of code size. If you use `extern inline` in your C code, an addressable version of such functions must be provided. (This is normally the case anyway, but if you get lucky and the optimizer always expands the functions inline, you might have gotten away without providing static copies.)

A function may be given the attribute `no_instrument_function`, in which case this instrumentation is not done. This can be used, for example, for the profiling functions listed above, high−priority interrupt routines, and any functions from which the profiling functions cannot safely be called (perhaps signal handlers, if the profiling routines generate output or allocate memory).

−finstrument−functions−once

This is similar to **−finstrument−functions**, but the profiling functions are called only once per instrumented function, i.e. the first profiling function is called after the first entry into the instrumented function and the second profiling function is called before the exit corresponding to this first entry.

The definition of `once` for the purpose of this option is a little vague because the implementation is not protected against data races. As a result, the implementation only guarantees that the profiling functions are called at *least* once per process and at *most* once per thread, but the calls are always paired, that is to say, if a thread calls the first function, then it will call the second function, unless it never reaches the exit of the instrumented function.

−finstrument−functions−exclude−file−list=*file***,***file***,...**
> Set the list of functions that are excluded from instrumentation (see the description of **−finstrument−functions**). If the file that contains a function definition matches with one of *file*, then that function is not instrumented. The match is done on substrings: if the *file* parameter is a substring of the file name, it is considered to be a match.
>
> For example:
>
> > −finstrument−functions−exclude−file−list=/bits/stl,include/sys
>
> excludes any inline function defined in files whose pathnames contain */bits/stl* or *include/sys*.
>
> If, for some reason, you want to include letter **,** in one of *sym*, write **,**. For example, **−finstrument−functions−exclude−file−list=',,tmp'** (note the single quote surrounding the option).

−finstrument−functions−exclude−function−list=*sym***,***sym***,...**
> This is similar to **−finstrument−functions−exclude−file−list**, but this option sets the list of function names to be excluded from instrumentation. The function name to be matched is its user−visible name, such as `vector<int> blah(const vector<int> &)`, not the internal mangled name (e.g., `_Z4blahRSt6vectorIiSaIiEE`). The match is done on substrings: if the *sym* parameter is a substring of the function name, it is considered to be a match. For C99 and C++ extended identifiers, the function name must be given in UTF−8, not using universal character names.

−fpatchable−function−entry=*N***[,***M***]**
> Generate *N* NOPs right at the beginning of each function, with the function entry point before the *M*th NOP. If *M* is omitted, it defaults to `0` so the function entry points to the address just at the first NOP. The NOP instructions reserve extra space which can be used to patch in any desired instrumentation at run time, provided that the code segment is writable. The amount of space is controllable indirectly via the number of NOPs; the NOP instruction used corresponds to the instruction emitted by the internal GCC back−end interface `gen_nop`. This behavior is target−specific and may also depend on the architecture variant and/or other compilation options.
>
> For run−time identification, the starting addresses of these areas, which correspond to their respective function entries minus *M*, are additionally collected in the `__patchable_function_entries` section of the resulting binary.
>
> Note that the value of `__attribute__ ((patchable_function_entry (N,M)))` takes precedence over command−line option **−fpatchable−function−entry=N,M**. This can be used to increase the area size or to remove it completely on a single function. If `N=0`, no pad location is recorded.
>
> The NOP instructions are inserted at−−−and maybe before, depending on *M*−−−the function entry address, even before the prologue. On PowerPC with the ELFv2 ABI, for a function with dual entry points, the local entry point is this function entry address by default. See the **−msplit−patch−nops** option to change this.
>
> The maximum value of *N* and *M* is 65535. On PowerPC with the ELFv2 ABI, for a function with dual entry points, the supported values for *M* are 0, 2, 6 and 14 when not using **−msplit−patch−nops**.

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.

In addition to the options listed here, there are a number of options to control search paths for include files documented in **Directory Options**. Options to control preprocessor diagnostics are listed in **Warning Options**.

−D *name*
> Predefine *name* as a macro, with definition `1`.

−D *name*=*definition*

The contents of *definition* are tokenized and processed as if they appeared during translation phase three in a **#define** directive. In particular, the definition is truncated by embedded newline characters.

If you are invoking the preprocessor from a shell or shell−like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax.

If you wish to define a function−like macro on the command line, write its argument list with surrounding parentheses before the equals sign (if any). Parentheses are meaningful to most shells, so you should quote the option. With **sh** and **csh**, **−D'**name(*args...*)=*definition*' works.

−D and **−U** options are processed in the order they are given on the command line. All **−imacros** *file* and **−include** *file* options are processed after all **−D** and **−U** options.

−U *name*

Cancel any previous definition of *name*, either built in or provided with a **−D** option.

−include *file*

Process *file* as if #include "file" appeared as the first line of the primary source file. However, the first directory searched for *file* is the preprocessor's working directory *instead of* the directory containing the main source file. If not found there, it is searched for in the remainder of the #include "..." search chain as normal.

If multiple **−include** options are given, the files are included in the order they appear on the command line.

−imacros *file*

Exactly like **−include**, except that any output produced by scanning *file* is thrown away. Macros it defines remain defined. This allows you to acquire all the macros from a header without also processing its declarations.

All files specified by **−imacros** are processed before all files specified by **−include**.

−undef

Do not predefine any system−specific or GCC−specific macros. The standard predefined macros remain defined.

−pthread

Define additional macros required for using the POSIX threads library. You should use this option consistently for both compilation and linking. This option is supported on GNU/Linux targets, most other Unix derivatives, and also on x86 Cygwin and MinGW targets.

−M

Instead of outputting the result of preprocessing, output a rule suitable for **make** describing the dependencies of the main source file. The preprocessor outputs one **make** rule containing the object file name for that source file, a colon, and the names of all the included files, including those coming from **−include** or **−imacros** command−line options.

Unless specified explicitly (with **−MT** or **−MQ**), the object file name consists of the name of the source file with any suffix replaced with object file suffix and with any leading directory parts removed. If there are many included files then the rule is split into several lines using \−newline. The rule has no commands.

This option does not suppress the preprocessor's debug output, such as **−dM**. To avoid mixing such debug output with the dependency rules you should explicitly specify the dependency output file with **−MF**, or use an environment variable like **DEPENDENCIES_OUTPUT**. Debug output is still sent to the regular output stream as normal.

Passing **−M** to the driver implies **−E**, and suppresses warnings with an implicit **−w**.

−MM

Like **−M** but do not mention header files that are found in system header directories, nor header files that are included, directly or indirectly, from such a header.

This implies that the choice of angle brackets or double quotes in an **#include** directive does not in itself determine whether that header appears in **−MM** dependency output.

−MF *file*

When used with **−M** or **−MM**, specifies a file to write the dependencies to. If no **−MF** switch is given the preprocessor sends the rules to the same place it would send preprocessed output.

When used with the driver options **−MD** or **−MMD**, **−MF** overrides the default dependency output file.

If *file* is −, then the dependencies are written to *stdout*.

−MG

In conjunction with an option such as **−M** requesting dependency generation, **−MG** assumes missing header files are generated files and adds them to the dependency list without raising an error. The dependency filename is taken directly from the #include directive without prepending any path. **−MG** also suppresses preprocessed output, as a missing header file renders this useless.

This feature is used in automatic updating of makefiles.

−Mno−modules

Disable dependency generation for compiled module interfaces.

−MP

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

This is typical output:

```
test.o: test.c test.h

test.h:
```

−MT *target*

Change the target of the rule emitted by dependency generation. By default CPP takes the name of the main input file, deletes any directory components and any file suffix such as **.c**, and appends the platform's usual object suffix. The result is the target.

An **−MT** option sets the target to be exactly the string you specify. If you want multiple targets, you can specify them as a single argument to **−MT**, or use multiple **−MT** options.

For example, **−MT '$(objpfx)foo.o'** might give

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

−MQ *target*

Same as **−MT**, but it quotes any characters which are special to Make. **−MQ '$(objpfx)foo.o'** gives

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

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

−MD

−MD is equivalent to **−M −MF** *file*, except that **−E** is not implied. The driver determines *file* based on whether an **−o** option is given. If it is, the driver uses its argument but with a suffix of *.d*, otherwise it takes the name of the input file, removes any directory components and suffix, and applies a *.d* suffix.

If **−MD** is used in conjunction with **−E**, any **−o** switch is understood to specify the dependency output file, but if used without **−E**, each **−o** is understood to specify a target object file.

Since **−E** is not implied, **−MD** can be used to generate a dependency output file as a side effect of the compilation process.

−MMD

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

−fpreprocessed

>   Indicate to the preprocessor that the input file has already been preprocessed. This suppresses things like macro expansion, trigraph conversion, escaped newline splicing, and processing of most directives. The preprocessor still recognizes and removes comments, so that you can pass a file preprocessed with **−C** to the compiler without problems. In this mode the integrated preprocessor is little more than a tokenizer for the front ends.
>
>   **−fpreprocessed** is implicit if the input file has one of the extensions **.i**, **.ii** or **.mi**. These are the extensions that GCC uses for preprocessed files created by **−save−temps**.

−fdirectives−only

>   When preprocessing, handle directives, but do not expand macros.
>
>   The option's behavior depends on the **−E** and **−fpreprocessed** options.
>
>   With **−E**, preprocessing is limited to the handling of directives such as #define, #ifdef, and #error. Other preprocessor operations, such as macro expansion and trigraph conversion are not performed. In addition, the **−dD** option is implicitly enabled.
>
>   With **−fpreprocessed**, predefinition of command line and most builtin macros is disabled. Macros such as __LINE__, which are contextually dependent, are handled normally. This enables compilation of files previously preprocessed with -E -fdirectives-only.
>
>   With both **−E** and **−fpreprocessed**, the rules for **−fpreprocessed** take precedence. This enables full preprocessing of files previously preprocessed with -E -fdirectives-only.

−fdollars−in−identifiers

>   Accept **$** in identifiers.

−fextended−identifiers

>   Accept universal character names and extended characters in identifiers. This option is enabled by default for C99 (and later C standard versions) and C++.

−fno−canonical−system−headers

>   When preprocessing, do not shorten system header paths with canonicalization.

−fmax−include−depth=depth

>   Set the maximum depth of the nested #include. The default is 200.

−fsearch−include−path[=*kind*]

>   Look for input files on the #include path, not just the current directory. This is particularly useful with C++20 modules, for which both header units and module interface units need to be compiled directly:
>
>   g++ −c −std=c++20 −fmodules −fsearch−include−path bits/stdc++.h bits/std.
>
>   *kind* defaults to **user**, which looks on the #include "..." search path; you can also explicitly specify **system** for the #include <...> search path.

−ftabstop=width

>   Set the distance between tab stops. This helps the preprocessor report correct column numbers in warnings or errors, even if tabs appear on the line. If the value is less than 1 or greater than 100, the option is ignored. The default is 8.

−ftrack−macro−expansion[=*level*]

>   Track locations of tokens across macro expansions. This allows the compiler to emit diagnostic about the current macro expansion stack when a compilation error occurs in a macro expansion. Using this option makes the preprocessor and the compiler consume more memory. The *level* parameter can be used to choose the level of precision of token location tracking thus decreasing the memory consumption if necessary. Value **0** of *level* de−activates this option. Value **1** tracks tokens locations in a degraded mode for the sake of minimal memory overhead. In this mode all tokens resulting from the

expansion of an argument of a function−like macro have the same location. Value **2** tracks tokens locations completely. This value is the most memory hungry. When this option is given no argument, the default parameter value is **2**.

Note that −ftrack-macro-expansion=2 is activated by default.

−fmacro−prefix−map=*old***=***new*
>   When preprocessing files residing in directory *old*, expand the __FILE__ and __BASE_FILE__ macros as if the files resided in directory *new* instead. This can be used to change an absolute path to a relative path by using . for *new* which can result in more reproducible builds that are location independent. This option also affects __builtin_FILE() during compilation. See also **−ffile−prefix−map** and **−fcanon−prefix−map**.

−fexec−charset=*charset*
>   Set the execution character set, used for string and character constants. The default is UTF−8. *charset* can be any encoding supported by the system's iconv library routine.

−fwide−exec−charset=*charset*
>   Set the wide execution character set, used for wide string and character constants. The default is one of UTF−32BE, UTF−32LE, UTF−16BE, or UTF−16LE, whichever corresponds to the width of wchar_t and the big−endian or little−endian byte order being used for code generation. As with **−fexec−charset**, *charset* can be any encoding supported by the system's iconv library routine; however, you will have problems with encodings that do not fit exactly in wchar_t.

−finput−charset=*charset*
>   Set the input character set, used for translation from the character set of the input file to the source character set used by GCC. If the locale does not specify, or GCC cannot get this information from the locale, the default is UTF−8. This can be overridden by either the locale or this command−line option. Currently the command−line option takes precedence if there's a conflict. *charset* can be any encoding supported by the system's iconv library routine.

−fpch−deps
>   When using precompiled headers, this flag causes the dependency−output flags to also list the files from the precompiled header's dependencies. If not specified, only the precompiled header are listed and not the files that were used to create it, because those files are not consulted when a precompiled header is used.

−fpch−preprocess
>   This option allows use of a precompiled header together with **−E**. It inserts a special #pragma, #pragma GCC pch_preprocess "*filename*" in the output to mark the place where the precompiled header was found, and its *filename*. When **−fpreprocessed** is in use, GCC recognizes this #pragma and loads the PCH.

>   This option is off by default, because the resulting preprocessed output is only really suitable as input to GCC. It is switched on by **−save−temps**.

>   You should not write this #pragma in your own code, but it is safe to edit the filename if the PCH file is available in a different location. The filename may be absolute or it may be relative to GCC's current directory.

−fworking−directory
>   Enable generation of linemarkers in the preprocessor output that let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor emits, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC uses this directory, when it's present in the preprocessed input, as the directory emitted as the current working directory in some debugging information formats. This option is implicitly enabled if debugging information is enabled, but this can be inhibited with the negated form **−fno−working−directory**. If the **−P** flag is present in the command line, this option has no effect, since no #line directives are emitted whatsoever.

−A *predicate=answer*

>   Make an assertion with the predicate *predicate* and answer *answer*. This form is preferred to the older form **−A** *predicate*(*answer*), which is still supported, because it does not use shell special characters.

−A −*predicate=answer*

>   Cancel an assertion with the predicate *predicate* and answer *answer*.

−C   Do not discard comments. All comments are passed through to the output file, except for comments in processed directives, which are deleted along with the directive.

>   You should be prepared for side effects when using **−C**; it causes the preprocessor to treat comments as tokens in their own right. For example, comments appearing at the start of what would be a directive line have the effect of turning that line into an ordinary source line, since the first token on the line is no longer a **#**.

−CC

>   Do not discard comments, including during macro expansion. This is like **−C**, except that comments contained within macros are also passed through to the output file where the macro is expanded.

>   In addition to the side effects of the **−C** option, the **−CC** option causes all C++−style comments inside a macro to be converted to C−style comments. This is to prevent later use of that macro from inadvertently commenting out the remainder of the source line.

>   The **−CC** option is generally used to support lint comments.

−P   Inhibit generation of linemarkers in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code, and will be sent to a program which might be confused by the linemarkers.

−traditional
−traditional−cpp

>   Try to imitate the behavior of pre−standard C preprocessors, as opposed to ISO C preprocessors. See the GNU CPP manual for details.

>   Note that GCC does not otherwise attempt to emulate a pre−standard C compiler, and these options are only supported with the **−E** switch, or when invoking CPP explicitly.

−trigraphs

>   Support ISO C trigraphs. These are three−character sequences, all starting with **??**, that are defined by ISO C to stand for single characters. For example, **??/** stands for ****, so **'??/n'** is a character constant for a newline.

>   The nine trigraphs and their replacements are

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

>   By default, GCC ignores trigraphs, but in standard−conforming modes it converts them. See the **−std** and **−ansi** options.

−remap

>   Enable special code to work around file systems which only permit very short file names, such as MS−DOS.

−H   Print the name of each header file used, in addition to other normal activities. Each name is indented to show how deep in the **#include** stack it is. Precompiled header files are also printed, even if they are found to be invalid; an invalid precompiled header file is printed with **...x** and a valid one with **...!** .

−d*letters*

>   Says to make debugging dumps during compilation as specified by *letters*. The flags documented here are those relevant to the preprocessor. Other *letters* are interpreted by the compiler proper, or reserved for future versions of GCC, and so are silently ignored. If you specify *letters* whose behavior conflicts, the result is undefined.

−dM

Instead of the normal output, generate a list of **#define** directives for all the macros defined during the execution of the preprocessor, including predefined macros. This gives you a way of finding out what is predefined in your version of the preprocessor. Assuming you have no file *foo.h*, the command

```
touch foo.h; cpp -dM foo.h
```

shows all the predefined macros.

If you use **−dM** without the **−E** option, **−dM** is interpreted as a synonym for **−fdump−rtl−mach**.

−dD

Like **−dM** except that it outputs *both* the **#define** directives and the result of preprocessing. Both kinds of output go to the standard output file.

−dN

Like **−dD**, but emit only the macro names, not their expansions.

−dI

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

−dU

Like **−dD** except that only macros that are expanded, or whose definedness is tested in preprocessor directives, are output; the output is delayed until the use or test of the macro; and **#undef** directives are also output for macros tested but undefined at the time.

−fdebug−cpp

This option is only useful for debugging GCC. When used from CPP or with **−E**, it dumps debugging information about location maps. Every token in the output is preceded by the dump of the map its location belongs to.

When used from GCC without **−E**, this option has no effect.

−Wp,*option*

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.

−Xpreprocessor *option*

Pass *option* as an option to the preprocessor. You can use this to supply system−specific preprocessor options that GCC does not recognize.

If you want to pass an option that takes an argument, you must use **−Xpreprocessor** twice, once for the option and once for the argument.

−no−integrated−cpp

Perform preprocessing as a separate pass before compilation. By default, GCC performs preprocessing as an integrated part of input tokenization and parsing. If this option is provided, the appropriate language front end (**cc1**, **cc1plus**, or **cc1obj** for C, C++, and Objective−C, respectively) is instead invoked twice, once for preprocessing only and once for actual compilation of the preprocessed input. This option may be useful in conjunction with the **−B** or **−wrapper** options to specify an alternate preprocessor or perform additional processing of the program source between normal preprocessing and compilation.

Passing Options to the Assembler

You can pass options to the assembler.

–Wa,*option*

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

–Xassembler *option*

Pass *option* as an option to the assembler. You can use this to supply system–specific assembler options that GCC does not recognize.

If you want to pass an option that takes an argument, you must use **–Xassembler** twice, once for the option and once for the argument.

Options for Linking

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

object–file–name

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

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

–flinker–output=*type*

This option controls code generation of the link–time optimizer. By default the linker output is automatically determined by the linker plugin. For debugging the compiler and if incremental linking with a non–LTO object file is desired, it may be useful to control the type manually.

If *type* is **exec**, code generation produces a static binary. In this case **–fpic** and **–fpie** are both disabled.

If *type* is **dyn**, code generation produces a shared library. In this case **–fpic** or **–fPIC** is preserved, but not enabled automatically. This allows to build shared libraries without position–independent code on architectures where this is possible, i.e. on x86.

If *type* is **pie**, code generation produces an **–fpie** executable. This results in similar optimizations as **exec** except that **–fpie** is not disabled if specified at compilation time.

If *type* is **rel**, the compiler assumes that incremental linking is done. The sections containing intermediate code for link–time optimization are merged, pre–optimized, and output to the resulting object file. In addition, if **–ffat–lto–objects** is specified, binary code is produced for future non–LTO linking. The object file produced by incremental linking is smaller than a static library produced from the same object files. At link time the result of incremental linking also loads faster than a static library assuming that the majority of objects in the library are used.

Finally **nolto–rel** configures the compiler for incremental linking where code generation is forced, a final binary is produced, and the intermediate code for later link–time optimization is stripped. When multiple object files are linked together the resulting code is better optimized than with link–time optimizations disabled (for example, cross–module inlining happens), but most of benefits of whole program optimizations are lost.

During the incremental link (by **–r**) the linker plugin defaults to **rel**. With current interfaces to GNU Binutils it is however not possible to incrementally link LTO objects and non–LTO objects into a single mixed object file. If any of object files in incremental link cannot be used for link–time optimization, the linker plugin issues a warning and uses **nolto–rel**. To maintain whole program optimization, it is recommended to link such objects into static library instead. Alternatively it is possible to use H.J. Lu's binutils with support for mixed objects.

–fuse–ld=bfd

Use the **bfd** linker instead of the default linker.

−fuse−ld=gold

Use the **gold** linker instead of the default linker.

−fuse−ld=lld

Use the LLVM **lld** linker instead of the default linker.

−fuse−ld=mold

Use the Modern Linker (**mold**) instead of the default linker.

−l*library*
−l *library*

Search the library named *library* when linking. (The second alternative with the library as a separate argument is only for POSIX compliance and is not recommended.)

The **−l** option is passed directly to the linker by GCC. Refer to your linker documentation for exact details. The general description below applies to the GNU linker.

The linker searches a standard list of directories for the library. The directories searched include several standard system directories plus any that you specify with **−L**.

Static libraries are archives of object files, and have file names like *liblibrary.a*. Some targets also support shared libraries, which typically have names like *liblibrary.so*. If both static and shared libraries are found, the linker gives preference to linking with the shared library unless the **−static** option is used.

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

−lobjc

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

−nostartfiles

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

−nodefaultlibs

Do not use the standard system libraries when linking. Only the libraries you specify are passed to the linker, and options specifying linkage of the system libraries, such as **−static−libgcc** or **−shared−libgcc**, are ignored. The standard startup files are used normally, unless **−nostartfiles** is used.

The compiler may generate calls to `memcmp`, `memset`, `memcpy` and `memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

−nolibc

Do not use the C library or system libraries tightly coupled with it when linking. Still link with the startup files, *libgcc* or toolchain provided language support libraries such as *libgnat*, *libgfortran* or *libstdc++* unless options preventing their inclusion are used as well. This typically removes **−lc** from the link command line, as well as system libraries that normally go with it and become meaningless when absence of a C library is assumed, for example **−lpthread** or **−lm** in some configurations. This is intended for bare−board targets when there is indeed no C library available.

−nostdlib

Do not use the standard system startup files or libraries when linking. No startup files and only the libraries you specify are passed to the linker, and options specifying linkage of the system libraries, such as **−static−libgcc** or **−shared−libgcc**, are ignored.

The compiler may generate calls to `memcmp`, `memset`, `memcpy` and `memmove`. These entries are usually resolved by entries in libc. These entry points should be supplied through some other mechanism when this option is specified.

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

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

−nostdlib++

Do not implicitly link with standard C++ libraries.

−e *entry*

−−entry=*entry*

Specify that the program entry point is *entry*. The argument is interpreted by the linker; the GNU linker accepts either a symbol name or an address.

−pie

Produce a dynamically linked position independent executable on targets that support it. For predictable results, you must also specify the same set of options used for compilation (**−fpie**, **−fPIE**, or model suboptions) when you specify this linker option.

−no−pie

Don't produce a dynamically linked position independent executable.

−static−pie

Produce a static position independent executable on targets that support it. A static position independent executable is similar to a static executable, but can be loaded at any address without a dynamic linker. For predictable results, you must also specify the same set of options used for compilation (**−fpie**, **−fPIE**, or model suboptions) when you specify this linker option.

−pthread

Link with the POSIX threads library. This option is supported on GNU/Linux targets, most other Unix derivatives, and also on x86 Cygwin and MinGW targets. On some targets this option also sets flags for the preprocessor, so it should be used consistently for both compilation and linking.

−r   Produce a relocatable object as output. This is also known as partial linking.

−rdynamic

Pass the flag **−export−dynamic** to the ELF linker, on targets that support it. This instructs the linker to add all symbols, not only used ones, to the dynamic symbol table. This option is needed for some uses of `dlopen` or to allow obtaining backtraces from within a program.

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

−static

On systems that support dynamic linking, this overrides **−pie** and prevents linking with the shared libraries. On other systems, this option has no effect.

−shared

Produce a shared object which can then be linked with other objects to form an executable. Not all systems support this option. For predictable results, you must also specify the same set of options used for compilation (**−fpic**, **−fPIC**, or model suboptions) when you specify this linker option.[1]

−shared−libgcc

−static−libgcc

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

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

should use the shared *libgcc*.

Therefore, the G++ driver automatically adds **−shared−libgcc** whenever you build a shared library or a main executable, because C++ programs typically use exceptions, so this is the right thing to do.

If, instead, you use the GCC driver to create shared libraries, you may find that they are not always linked with the shared *libgcc*. If GCC finds, at its configuration time, that you have a non−GNU linker or a GNU linker that does not support option **−−eh−frame−hdr**, it links the shared version of *libgcc* into shared libraries by default. Otherwise, it takes advantage of the linker and optimizes away the linking with the shared version of *libgcc*, linking with the static version of libgcc by default. This allows exceptions to propagate through such shared libraries, without incurring relocation costs at library load time.

However, if a library or main executable is supposed to throw or catch exceptions, you must link it using the G++ driver, or using the option **−shared−libgcc**, such that it is linked with the shared *libgcc*.

−static−libasan

When the **−fsanitize=address** option is used to link a program, the GCC driver automatically links against **libasan**. If *libasan* is available as a shared library, and the **−static** option is not used, then this links against the shared version of *libasan*. The **−static−libasan** option directs the GCC driver to link *libasan* statically, without necessarily linking other libraries statically.

−static−libtsan

When the **−fsanitize=thread** option is used to link a program, the GCC driver automatically links against **libtsan**. If *libtsan* is available as a shared library, and the **−static** option is not used, then this links against the shared version of *libtsan*. The **−static−libtsan** option directs the GCC driver to link *libtsan* statically, without necessarily linking other libraries statically.

−static−liblsan

When the **−fsanitize=leak** option is used to link a program, the GCC driver automatically links against **liblsan**. If *liblsan* is available as a shared library, and the **−static** option is not used, then this links against the shared version of *liblsan*. The **−static−liblsan** option directs the GCC driver to link *liblsan* statically, without necessarily linking other libraries statically.

−static−libubsan

When the **−fsanitize=undefined** option is used to link a program, the GCC driver automatically links against **libubsan**. If *libubsan* is available as a shared library, and the **−static** option is not used, then this links against the shared version of *libubsan*. The **−static−libubsan** option directs the GCC driver to link *libubsan* statically, without necessarily linking other libraries statically.

−static−libstdc++

When the **g++** program is used to link a C++ program, it normally automatically links against **libstdc++**. If *libstdc++* is available as a shared library, and the **−static** option is not used, then this links against the shared version of *libstdc++*. That is normally fine. However, it is sometimes useful to freeze the version of *libstdc++* used by the program without going all the way to a fully static link. The **−static−libstdc++** option directs the **g++** driver to link *libstdc++* statically, without necessarily linking other libraries statically.

−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.

−T *script*

Use *script* as the linker script. This option is supported by most systems using the GNU linker. On some targets, such as bare−board targets without an operating system, the **−T** option may be required when linking to avoid references to undefined symbols.

−Xlinker *option*

Pass *option* as an option to the linker. You can use this to supply system−specific linker options that GCC does not recognize.

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

When using the GNU linker, it is usually more convenient to pass arguments to linker options using the *option*=*value* syntax than as separate arguments. For example, you can specify **−Xlinker −Map=output.map** rather than **−Xlinker −Map −Xlinker output.map**. Other linkers may not support this syntax for command−line options.

−Wl,*option*

Pass *option* as an option to the linker. If *option* contains commas, it is split into multiple options at the commas. You can use this syntax to pass an argument to the option. For example, **−Wl,−Map,output.map** passes **−Map output.map** to the linker. When using the GNU linker, you can also get the same effect with **−Wl,−Map=output.map**.

−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.

−z *keyword*

−z is passed directly on to the linker along with the keyword *keyword*. See the section in the documentation of your linker for permitted values and their meanings.

Options for Directory Search

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

−I *dir*
−iquote *dir*
−isystem *dir*
−idirafter *dir*

Add the directory *dir* to the list of directories to be searched for header files during preprocessing. If *dir* begins with **=** or `$SYSROOT`, then the **=** or `$SYSROOT` is replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

Directories specified with **−iquote** apply only to the quote form of the directive, `#include "file"`. Directories specified with **−I**, **−isystem**, or **−idirafter** apply to lookup for both the `#include "file"` and `#include <file>` directives.

You can specify any number or combination of these options on the command line to search for header files in several directories. The lookup order is as follows:

1. For the quote form of the include directive, the directory of the current file is searched first.

2. For the quote form of the include directive, the directories specified by **−iquote** options are searched in left−to−right order, as they appear on the command line.

3. Directories specified with **−I** options are scanned in left−to−right order.

4. Directories specified with **−isystem** options are scanned in left−to−right order.

5. Standard system directories are scanned.

6. Directories specified with **−idirafter** options are scanned in left−to−right order.

You can use **−I** to override a system header file, substituting your own version, since these directories are searched before the standard system header file directories. However, you should not use this option to add directories that contain vendor−supplied system header files; use **−isystem** for that.

The **−isystem** and **−idirafter** options also mark the directory as a system directory, so that it gets the same special treatment that is applied to the standard system directories.

If a standard system include directory, or a directory specified with **−isystem**, is also specified with **−I**, the **−I** option is ignored. The directory is still searched but as a system directory at its normal position

in the system include chain. This is to ensure that GCC's procedure to fix buggy system headers and the ordering for the #include_next directive are not inadvertently changed. If you really need to change the search order for system directories, use the **−nostdinc** and/or **−isystem** options.

−I−

Split the include path. This option has been deprecated. Please use **−iquote** instead for **−I** directories before the **−I−** and remove the **−I−** option.

Any directories specified with **−I** options before **−I−** are searched only for headers requested with #include "*file*"; they are not searched for #include <*file*>. If additional directories are specified with **−I** options after the **−I−**, those directories are searched for all **#include** directives.

In addition, **−I−** inhibits the use of the directory of the current file directory as the first search directory for #include "*file*". There is no way to override this effect of **−I−**.

−iprefix *prefix*

Specify *prefix* as the prefix for subsequent **−iwithprefix** options. If the prefix represents a directory, you should include the final **/**.

−iwithprefix *dir*

−iwithprefixbefore *dir*

Append *dir* to the prefix specified previously with **−iprefix**, and add the resulting directory to the include search path. **−iwithprefixbefore** puts it in the same place **−I** would; **−iwithprefix** puts it where **−idirafter** would.

−isysroot *dir*

This option is like the **−−sysroot** option, but applies only to header files (except for Darwin targets, where it applies to both header files and libraries). See the **−−sysroot** option for more information.

−imultilib *dir*

Use *dir* as a subdirectory of the directory containing target−specific C++ headers.

−nostdinc

Do not search the standard system directories for header files. Only the directories explicitly specified with **−I**, **−iquote**, **−isystem**, and/or **−idirafter** options (and the directory of the current file, if appropriate) are searched.

−nostdinc++

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

−−embed−dir=*dir*

Append *dir* directory to the list of searched directories for #embed preprocessing directive or __has_embed macro. There are no default directories for #embed.

If *dir* begins with **=** or $SYSROOT, then the **=** or $SYSROOT is replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−iplugindir=*dir*

Set the directory to search for plugins that are passed by **−fplugin=***name* instead of **−fplugin=***path*/*name***.so**. This option is not meant to be used by the user, but only passed by the driver.

−L*dir*

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

−B*prefix*

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

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

For each subprogram to be run, the compiler driver first tries the **−B** prefix, if any. If that name is not

found, or if **−B** is not specified, the driver tries two standard prefixes, */usr/lib/gcc/* and */usr/local/lib/gcc/*. If neither of those results in a file name that is found, the unmodified program name is searched for using the directories specified in your **PATH** environment variable.

The compiler checks to see if the path provided by **−B** refers to a directory, and if necessary it adds a directory separator character at the end of the path.

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

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

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

As a special kludge, if the path provided by **−B** is *[dir/]stageN/*, where *N* is a number in the range 0 to 9, then it is replaced by *[dir/]include*. This is to help with boot−strapping the compiler.

−no−canonical−prefixes
    Do not expand any symbolic links, resolve references to **/../** or **/./**, or make the path absolute when generating a relative prefix.

−−sysroot=*dir*
    Use *dir* as the logical root directory for headers and libraries. For example, if the compiler normally searches for headers in */usr/include* and libraries in */usr/lib*, it instead searches *dir/usr/include* and *dir/usr/lib*.

    If you use both this option and the **−isysroot** option, then the **−−sysroot** option applies to libraries, but the **−isysroot** option applies to header files.

    The GNU linker (beginning with version 2.16) has the necessary support for this option. If your linker does not support this option, the header file aspect of **−−sysroot** still works, but the library aspect does not.

−−no−sysroot−suffix
    For some targets, a suffix is added to the root directory specified with **−−sysroot**, depending on the other options used, so that headers may for example be found in *dir/suffix/usr/include* instead of *dir/usr/include*. This option disables the addition of such a suffix.

Options for Code Generation Conventions
    These machine−independent options control the interface conventions used in code generation.

    Most of them have both positive and negative forms; the negative form of **−ffoo** is **−fno−foo**. In the table below, only one of the forms is listed−−−the one that is not the default. You can figure out the other form by either removing **no−** or adding it.

−fstack−reuse=*reuse−level*
    This option controls stack space reuse for user declared local/auto variables and compiler generated temporaries. *reuse_level* can be **all**, **named_vars**, or **none**. **all** enables stack reuse for all local variables and temporaries, **named_vars** enables the reuse only for user defined local variables with names, and **none** disables stack reuse completely. The default value is **all**. The option is needed when the program extends the lifetime of a scoped local variable or a compiler generated temporary beyond the end point defined by the language. When a lifetime of a variable ends, and if the variable lives in memory, the optimizing compiler has the freedom to reuse its stack space with other temporaries or scoped local variables whose live range does not overlap with it. Legacy code extending local lifetime is likely to break with the stack reuse optimization.

    For example,

```
int *p;
{
  int local1;

  p = &local1;
  local1 = 10;
  ....
}
{
   int local2;
   local2 = 20;
   ...
}

if (*p == 10)  // out of scope use of local1
  {

  }
```

Another example:

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

A *ap;

void foo(const A& ar)
{
    ap = &ar;
}

void bar()
{
    foo(A(10)); // temp object's lifetime ends when foo returns

    {
      A a(20);
      ....
    }
    ap->i+= 10;  // ap references out of scope temp whose space
                 // is reused with a. What is the value of ap->i?
}
```

The lifetime of a compiler generated temporary is well defined by the C++ standard. When a lifetime of a temporary ends, and if the temporary lives in memory, the optimizing compiler has the freedom to reuse its stack space with other temporaries or scoped local variables whose live range does not overlap with it. However some of the legacy code relies on the behavior of older compilers in which temporaries' stack space is not reused, the aggressive stack reuse can lead to runtime errors. This option is used to control the temporary stack reuse optimization.

−ftrapv
This option generates traps for signed overflow on addition, subtraction, multiplication operations. The options **−ftrapv** and **−fwrapv** override each other, so using **−ftrapv −fwrapv** on the

command−line results in **−fwrapv** being effective. Note that only active options override, so using **−ftrapv −fwrapv −fno−wrapv** on the command−line results in **−ftrapv** being effective.

−fwrapv

This option instructs the compiler to assume that signed arithmetic overflow of addition, subtraction and multiplication wraps around using twos−complement representation. This flag enables some optimizations and disables others. The options **−ftrapv** and **−fwrapv** override each other, so using **−ftrapv −fwrapv** on the command−line results in **−fwrapv** being effective. Note that only active options override, so using **−ftrapv −fwrapv −fno−wrapv** on the command−line results in **−ftrapv** being effective.

−fwrapv−pointer

This option instructs the compiler to assume that pointer arithmetic overflow on addition and subtraction wraps around using twos−complement representation. This flag disables some optimizations which assume pointer overflow is invalid.

−fstrict−overflow

This option implies **−fno−wrapv −fno−wrapv−pointer** and when negated implies **−fwrapv −fwrapv−pointer**.

−fexceptions

Enable exception handling. Generates extra code needed to propagate exceptions. For some targets, this implies GCC generates 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 enables it by default for languages like C++ that normally require exception handling, and disables 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.

−fnon−call−exceptions

Generate code that allows trapping instructions to throw exceptions. Note that this requires platform−specific runtime support that does not exist everywhere. Moreover, it only allows *trapping* instructions to throw exceptions, i.e. memory references or floating−point instructions. It does not allow exceptions to be thrown from arbitrary signal handlers such as `SIGALRM`. This enables **−fexceptions**.

−fdelete−dead−exceptions

Consider that instructions that may throw exceptions but don't otherwise contribute to the execution of the program can be optimized away. This does not affect calls to functions except those with the `pure` or `const` attributes. This option is enabled by default for the Ada and C++ compilers, as permitted by the language specifications. Optimization passes that cause dead exceptions to be removed are enabled independently at different optimization levels.

−funwind−tables

Similar to **−fexceptions**, except that it just generates any needed static data, but does not affect the generated code in any other way. You normally do not need to enable this option; instead, a language processor that needs this handling enables it on your behalf.

−fasynchronous−unwind−tables

Generate unwind table in DWARF 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).

−fno−gnu−unique

On systems with recent GNU assembler and C library, the C++ compiler uses the `STB_GNU_UNIQUE` binding to make sure that definitions of template static data members and static local variables in inline functions are unique even in the presence of `RTLD_LOCAL`; this is necessary to avoid problems with a library used by two different `RTLD_LOCAL` plugins depending on a definition in one of them and therefore disagreeing with the other one about the binding of the symbol. But this causes

dlclose to be ignored for affected DSOs; if your program relies on reinitialization of a DSO via dlclose and dlopen, you can use **–fno–gnu–unique**.

–fpcc–struct–return

Return "short" struct and union values in memory like longer ones, rather than in registers. This convention is less efficient, but it has the advantage of allowing intercallability between GCC–compiled files and files compiled with other compilers, particularly the Portable C Compiler (pcc).

The precise convention for returning structures in memory depends on the target configuration macros.

Short structures and unions are those whose size and alignment match that of some integer type.

Warning: code compiled with the **–fpcc–struct–return** switch is not binary compatible with code compiled with the **–freg–struct–return** switch. Use it to conform to a non–default application binary interface.

–freg–struct–return

Return struct and union values 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**, GCC defaults to whichever convention is standard for the target. If there is no standard convention, GCC defaults to **–fpcc–struct–return**, except on targets where GCC is the principal compiler. In those cases, we can choose the standard, and we chose the more efficient register return alternative.

Warning: code compiled with the **–freg–struct–return** switch is not binary compatible with code compiled with the **–fpcc–struct–return** switch. Use it to conform to a non–default application binary interface.

–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 is equivalent to the smallest integer type that has enough room. This option has no effect for an enumeration type with a fixed underlying type.

Warning: the **–fshort–enums** switch causes GCC to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non–default application binary interface.

–fshort–wchar

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

Warning: the **–fshort–wchar** switch causes GCC to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non–default application binary interface.

–fcommon

In C code, this option controls the placement of global variables defined without an initializer, known as *tentative definitions* in the C standard. Tentative definitions are distinct from declarations of a variable with the extern keyword, which do not allocate storage.

The default is **–fno–common**, which specifies that the compiler places uninitialized global variables in the BSS section of the object file. This inhibits the merging of tentative definitions by the linker so you get a multiple–definition error if the same variable is accidentally defined in more than one compilation unit.

The **–fcommon** places uninitialized global variables in a common block. This allows the linker to resolve all tentative definitions of the same variable in different compilation units to the same object, or to a non–tentative definition. This behavior is inconsistent with C++, and on many targets implies a speed and code size penalty on global variable references. It is mainly useful to enable legacy code to link without errors.

−fno−ident

Ignore the #ident directive.

−finhibit−size−directive

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

−fverbose−asm

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

−fno−verbose−asm, the default, causes the extra information to be omitted and is useful when comparing two assembler files.

The added comments include:

*   information on the compiler version and command−line options,

*   the source code lines associated with the assembly instructions, in the form FILENAME:LINENUMBER:CONTENT OF LINE,

*   hints on which high−level expressions correspond to the various assembly instruction operands.

For example, given this C source file:

```
int test (int n)
{
  int i;
  int total = 0;

  for (i = 0; i < n; i++)
    total += i * i;

  return total;
}
```

compiling to (x86_64) assembly via **−S** and emitting the result direct to stdout via **−o −**

```
gcc −S test.c −fverbose-asm −Os −o −
```

gives output similar to this:

```
        .file   "test.c"
# GNU C11 (GCC) version 7.0.0 20160809 (experimental) (x86_64-pc-linux-gn
  [...snip...]
# options passed:
  [...snip...]

        .text
        .globl  test
        .type   test, @function
test:
.LFB0:
        .cfi_startproc
# test.c:4:   int total = 0;
        xorl    %eax, %eax      # <retval>
# test.c:6:   for (i = 0; i < n; i++)
        xorl    %edx, %edx      # i
.L2:
```

```
                        # test.c:6:    for (i = 0; i < n; i++)
                                cmpl    %edi, %edx       # n, i
                                jge     .L5      #,
                        # test.c:7:    total += i * i;
                                movl    %edx, %ecx       # i, tmp92
                                imull   %edx, %ecx       # i, tmp92
                        # test.c:6:    for (i = 0; i < n; i++)
                                incl    %edx     # i
                        # test.c:7:    total += i * i;
                                addl    %ecx, %eax       # tmp92, <retval>
                                jmp     .L2      #
                        .L5:
                        # test.c:10: }
                                ret
                                .cfi_endproc
                        .LFE0:
                                .size   test, .-test
                                .ident  "GCC: (GNU) 7.0.0 20160809 (experimental)"
                                .section        .note.GNU-stack,"",@progbits
```

The comments are intended for humans rather than machines and hence the precise format of the comments is subject to change.

−frecord−gcc−switches

This switch causes the command line used to invoke the compiler to be recorded into the object file that is being created. This switch is only implemented on some targets and the exact format of the recording is target and binary file format dependent, but it usually takes the form of a section containing ASCII text. This switch is related to the **−fverbose−asm** switch, but that switch only records information in the assembler output file as comments, so it never reaches the object file. See also **−grecord−gcc−switches** for another way of storing compiler options into the object file.

−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, 28k on AArch64 and 32k on the m68k and RS/6000. The x86 has no such limit.)

Position−independent code requires special support, and therefore works only on certain machines. For the x86, 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 AArch64, 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.

−fpie
−fPIE

These options are similar to **−fpic** and **−fPIC**, but the generated position−independent code can be only linked into executables. Usually these options are used to compile code that will be linked using

the **−pie** GCC option.

−fpie and **−fPIE** both define the macros `__pie__` and `__PIE__`. The macros have the value 1 for **−fpie** and 2 for **−fPIE**.

−fno−plt

Do not use the PLT for external function calls in position−independent code. Instead, load the callee address at call sites from the GOT and branch to it. This leads to more efficient code by eliminating PLT stubs and exposing GOT loads to optimizations. On architectures such as 32−bit x86 where PLT stubs expect the GOT pointer in a specific register, this gives more register allocation freedom to the compiler. Lazy binding requires use of the PLT; with **−fno−plt** all external symbols are resolved at load time.

Alternatively, the function attribute `noplt` can be used to avoid calls through the PLT for specific external functions.

In position−dependent code, a few targets also convert calls to functions that are marked to not use the PLT to use the GOT instead.

−fno−jump−tables

Do not use jump tables for switch statements even where it would be more efficient than other code generation strategies. This option is of use in conjunction with **−fpic** or **−fPIC** for building code that forms part of a dynamic linker and cannot reference the address of a jump table. On some targets, jump tables do not require a GOT and this option is not needed.

−fno−bit−tests

Do not use bit tests for switch statements even where it would be more efficient than other code generation strategies.

−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 do not save and restore the register *reg*.

It is an error to use this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model produces 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 save and restore the register *reg* if they use it.

It is an error to use this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model produces disastrous results.

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

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

−fpack−struct[=*n***]**

Without a value specified, pack all structure members together without holes. When a value is specified (which must be a small power of two), pack structure members according to this value,

representing the maximum alignment (that is, objects with default alignment requirements larger than this are output potentially unaligned at the next fitting location.

Warning: the **−fpack−struct** switch causes GCC to generate code that is not binary compatible with code generated without that switch. Additionally, it makes the code suboptimal. Use it to conform to a non−default application binary interface.

−fleading−underscore

This option and its counterpart, **−fno−leading−underscore**, forcibly change the way C symbols are represented in the object file. One use is to help link with legacy assembly code.

Warning: the **−fleading−underscore** switch causes GCC to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non−default application binary interface. Not all targets provide complete support for this switch.

−ftls−model=*model*

Alter the thread−local storage model to be used. The *model* argument should be one of **global−dynamic**, **local−dynamic**, **initial−exec** or **local−exec**. Note that the choice is subject to optimization: the compiler may use a more efficient model for symbols not visible outside of the translation unit, or if **−fpic** is not given on the command line.

The default without **−fpic** is **initial−exec**; with **−fpic** the default is **global−dynamic**.

−ftrampolines

For targets that normally need trampolines for nested functions, always generate them instead of using descriptors. Otherwise, for targets that do not need them, like for example HP−PA or IA−64, do nothing.

A trampoline is a small piece of code that is created at run time on the stack when the address of a nested function is taken, and is used to call the nested function indirectly. Therefore, it requires the stack to be made executable in order for the program to work properly.

−fno−trampolines is enabled by default on a language by language basis to let the compiler avoid generating them, if it computes that this is safe, and replace them with descriptors. Descriptors are made up of data only, but the generated code must be prepared to deal with them. As of this writing, **−fno−trampolines** is enabled by default only for Ada.

Moreover, code compiled with **−ftrampolines** and code compiled with **−fno−trampolines** are not binary compatible if nested functions are present. This option must therefore be used on a program−wide basis and be manipulated with extreme care.

For languages other than Ada, the `-ftrampolines` and `-fno-trampolines` options currently have no effect, and trampolines are always generated on platforms that need them for nested functions.

−ftrampoline−impl=[stack|heap]

By default, trampolines are generated on stack. However, certain platforms (such as the Apple M1) do not permit an executable stack. Compiling with **−ftrampoline−impl=heap** generate calls to `__gcc_nested_func_ptr_created` and `__gcc_nested_func_ptr_deleted` in order to allocate and deallocate trampoline space on the executable heap. These functions are implemented in libgcc, and will only be provided on specific targets: x86_64 Darwin, x86_64 and aarch64 Linux. *PLEASE NOTE*: Heap trampolines are *not* guaranteed to be correctly deallocated if you `setjmp`, instantiate nested functions, and then `longjmp` back to a state prior to having allocated those nested functions.

−fvisibility=[default|internal|hidden|protected]

Set the default ELF image symbol visibility to the specified option−−−all symbols are marked with this unless overridden within the code. Using this feature can very substantially improve linking and load times of shared object libraries, produce more optimized code, provide near−perfect API export and prevent symbol clashes. It is **strongly** recommended that you use this in any shared objects you distribute.

Despite the nomenclature, **default** always means public; i.e., available to be linked against from outside the shared object. **protected** and **internal** are pretty useless in real−world usage so the only other commonly used option is **hidden**. The default if **−fvisibility** isn't specified is **default**, i.e., make every symbol public.

A good explanation of the benefits offered by ensuring ELF symbols have the correct visibility is given by "How To Write Shared Libraries" by Ulrich Drepper (which can be found at **<https://www.akkadia.org/drepper/>**)−−−however a superior solution made possible by this option to marking things hidden when the default is public is to make the default hidden and mark things public. This is the norm with DLLs on Windows and with **−fvisibility=hidden** and `__attribute__ ((visibility("default")))` instead of `__declspec(dllexport)` you get almost identical semantics with identical syntax. This is a great boon to those working with cross−platform projects.

For those adding visibility support to existing code, you may find `#pragma GCC visibility` of use. This works by you enclosing the declarations you wish to set visibility for with (for example) `#pragma GCC visibility push(hidden)` and `#pragma GCC visibility pop`. Bear in mind that symbol visibility should be viewed **as part of the API interface contract** and thus all new code should always specify visibility when it is not the default; i.e., declarations only for use within the local DSO should **always** be marked explicitly as hidden as so to avoid PLT indirection overheads−−−making this abundantly clear also aids readability and self−documentation of the code. Note that due to ISO C++ specification requirements, `operator new` and `operator delete` must always be of default visibility.

Be aware that headers from outside your project, in particular system headers and headers from any other library you use, may not be expecting to be compiled with visibility other than the default. You may need to explicitly say `#pragma GCC visibility push(default)` before including any such headers.

`extern` declarations are not affected by **−fvisibility**, so a lot of code can be recompiled with **−fvisibility=hidden** with no modifications. However, this means that calls to `extern` functions with no explicit visibility use the PLT, so it is more effective to use `__attribute ((visibility))` and/or `#pragma GCC visibility` to tell the compiler which `extern` declarations should be treated as hidden.

Note that **−fvisibility** does affect C++ vague linkage entities. This means that, for instance, an exception class that is be thrown between DSOs must be explicitly marked with default visibility so that the **type_info** nodes are unified between the DSOs.

An overview of these techniques, their benefits and how to use them is at **<https://gcc.gnu.org/wiki/Visibility>**.

−fstrict−volatile−bitfields

> This option should be used if accesses to volatile bit−fields (or other structure fields, although the compiler usually honors those types anyway) should use a single access of the width of the field's type, aligned to a natural alignment if possible. For example, targets with memory−mapped peripheral registers might require all such accesses to be 16 bits wide; with this flag you can declare all peripheral bit−fields as `unsigned short` (assuming short is 16 bits on these targets) to force GCC to use 16−bit accesses instead of, perhaps, a more efficient 32−bit access.
>
> If this option is disabled, the compiler uses the most efficient instruction. In the previous example, that might be a 32−bit load instruction, even though that accesses bytes that do not contain any portion of the bit−field, or memory−mapped registers unrelated to the one being updated.
>
> In some cases, such as when the `packed` attribute is applied to a structure field, it may not be possible to access the field with a single read or write that is correctly aligned for the target machine. In this case GCC falls back to generating multiple accesses rather than code that will fault or truncate the result at run time.
>
> Note: Due to restrictions of the C/C++11 memory model, write accesses are not allowed to touch non

bit−field members. It is therefore recommended to define all bits of the field's type as bit−field members.

The default value of this option is determined by the application binary interface for the target processor.

−fsync−libcalls

This option controls whether any out−of−line instance of the `__sync` family of functions may be used to implement the C++11 `__atomic` family of functions.

The default value of this option is enabled, thus the only useful form of the option is **−fno−sync−libcalls**. This option is used in the implementation of the *libatomic* runtime library.

−fzero−init−padding−bits=value

Guarantee zero initialization of padding bits in automatic variable initializers. Certain languages guarantee zero initialization of padding bits in certain cases, e.g. C23 when using empty initializers (`{}`), or C++ when using zero−initialization or C guarantees that fields not specified in an initializer have their padding bits zero initialized. This option allows to change when padding bits in initializers are guaranteed to be zero initialized. The default is `−fzero-init-padding-bits=standard`, which makes no further guarantees than the corresponding standard. E.g.

```
struct A { char a; unsigned long long b; char c; };
union B { char a; unsigned long long b; };
struct A a = {}; // C23 guarantees padding bits are zero.
struct A b = { 1, 2, 3 }; // No guarantees.
union B c = {}; // C23 guarantees padding bits are zero.
union B d = { 1 }; // No guarantees.
```

`−fzero-init-padding-bits=unions` guarantees zero initialization of padding bits in unions on top of what the standards guarantee, if the initializer of an union is empty (then all bits of the union are zero initialized) or if the initialized member of the union is smaller than the size of the union (in that case guarantees padding bits outside of the initialized member to be zero initialized). This was the GCC behavior before GCC 15 and in the above example guarantees zero initialization of last `sizeof (unsigned long long) − 1` bytes in the union.

`−fzero-init-padding-bits=all` guarantees additionally zero initialization of padding bits of other aggregates, so the padding in between `b.a` and `b.b` (if any) and tail padding in the structure (if any).

GCC Developer Options

This section describes command−line options that are primarily of interest to GCC developers, including options to support compiler testing and investigation of compiler bugs and compile−time performance problems. This includes options that produce debug dumps at various points in the compilation; that print statistics such as memory use and execution time; and that print information about GCC's configuration, such as where it searches for libraries. You should rarely need to use any of these options for ordinary compilation and linking tasks.

Many developer options that cause GCC to dump output to a file take an optional **=**filename suffix. You can specify **stdout** or **−** to dump to standard output, and **stderr** for standard error.

If **=**filename is omitted, a default dump file name is constructed by concatenating the base dump file name, a pass number, phase letter, and pass name. The base dump file name is the name of output file produced by the compiler if explicitly specified and not an executable; otherwise it is the source file name. The pass number is determined by the order passes are registered with the compiler's pass manager. This is generally the same as the order of execution, but passes registered by plugins, target−specific passes, or passes that are otherwise registered late are numbered higher than the pass named **final**, even if they are executed earlier. The phase letter is one of **i** (inter−procedural analysis), **l** (language−specific), **r** (RTL), or **t** (tree). The files are created in the directory of the output file.

−fcallgraph−info
−fcallgraph−info=*MARKERS*
>   Makes the compiler output callgraph information for the program, on a per−object−file basis. The
>   information is generated in the common VCG format. It can be decorated with additional, per−node
>   and/or per−edge information, if a list of comma−separated markers is additionally specified. When
>   the su marker is specified, the callgraph is decorated with stack usage information; it is equivalent to
>   **−fstack−usage**. When the da marker is specified, the callgraph is decorated with information about
>   dynamically allocated objects.
>
>   When compiling with **−flto**, no callgraph information is output along with the object file. At LTO link
>   time, **−fcallgraph−info** may generate multiple callgraph information files next to intermediate LTO
>   output files.

−d*letters*
−fdump−rtl−*pass*
−fdump−rtl−*pass*=*filename*
>   Says to make debugging dumps during compilation at times specified by *letters*. This is used for
>   debugging the RTL−based passes of the compiler.
>
>   Some **−d***letters* switches have different meaning when **−E** is used for preprocessing.
>
>   Debug dumps can be enabled with a **−fdump−rtl** switch or some **−d** option *letters*. Here are the
>   possible letters for use in *pass* and *letters*, and their meanings:

>   **−fdump−rtl−alignments**
>   >   Dump after branch alignments have been computed.
>
>   **−fdump−rtl−asmcons**
>   >   Dump after fixing rtl statements that have unsatisfied in/out constraints.
>
>   **−fdump−rtl−auto_inc_dec**
>   >   Dump after auto−inc−dec discovery. This pass is only run on architectures that have auto inc or
>   >   auto dec instructions.
>
>   **−fdump−rtl−barriers**
>   >   Dump after cleaning up the barrier instructions.
>
>   **−fdump−rtl−bbpart**
>   >   Dump after partitioning hot and cold basic blocks.
>
>   **−fdump−rtl−bbro**
>   >   Dump after block reordering.
>
>   **−fdump−rtl−btl1**
>   **−fdump−rtl−btl2**
>   >   **−fdump−rtl−btl1** and **−fdump−rtl−btl2** enable dumping after the two branch target load
>   >   optimization passes.
>
>   **−fdump−rtl−bypass**
>   >   Dump after jump bypassing and control flow optimizations.
>
>   **−fdump−rtl−combine**
>   >   Dump after the RTL instruction combination pass.
>
>   **−fdump−rtl−compgotos**
>   >   Dump after duplicating the computed gotos.
>
>   **−fdump−rtl−ce1**
>   **−fdump−rtl−ce2**
>   **−fdump−rtl−ce3**
>   >   **−fdump−rtl−ce1**, **−fdump−rtl−ce2**, and **−fdump−rtl−ce3** enable dumping after the three if
>   >   conversion passes.

–fdump–rtl–cprop_hardreg
    Dump after hard register copy propagation.

–fdump–rtl–csa
    Dump after combining stack adjustments.

–fdump–rtl–cse1
–fdump–rtl–cse2
    –fdump–rtl–cse1 and **–fdump–rtl–cse2** enable dumping after the two common subexpression elimination passes.

–fdump–rtl–dce
    Dump after the standalone dead code elimination passes.

–fdump–rtl–dbr
    Dump after delayed branch scheduling.

–fdump–rtl–dce1
–fdump–rtl–dce2
    –fdump–rtl–dce1 and **–fdump–rtl–dce2** enable dumping after the two dead store elimination passes.

–fdump–rtl–eh
    Dump after finalization of EH handling code.

–fdump–rtl–eh_ranges
    Dump after conversion of EH handling range regions.

–fdump–rtl–expand
    Dump after RTL generation.

–fdump–rtl–fwprop1
–fdump–rtl–fwprop2
    –fdump–rtl–fwprop1 and **–fdump–rtl–fwprop2** enable dumping after the two forward propagation passes.

–fdump–rtl–gcse1
–fdump–rtl–gcse2
    –fdump–rtl–gcse1 and **–fdump–rtl–gcse2** enable dumping after global common subexpression elimination.

–fdump–rtl–init–regs
    Dump after the initialization of the registers.

–fdump–rtl–initvals
    Dump after the computation of the initial value sets.

–fdump–rtl–into_cfglayout
    Dump after converting to cfglayout mode.

–fdump–rtl–ira
    Dump after iterated register allocation.

–fdump–rtl–jump
    Dump after the second jump optimization.

–fdump–rtl–loop2
    –fdump–rtl–loop2 enables dumping after the rtl loop optimization passes.

–fdump–rtl–mach
    Dump after performing the machine dependent reorganization pass, if that pass exists.

–fdump–rtl–mode_sw
    Dump after removing redundant mode switches.

−fdump−rtl−rnreg
  Dump after register renumbering.

−fdump−rtl−outof_cfglayout
  Dump after converting from cfglayout mode.

−fdump−rtl−peephole2
  Dump after the peephole pass.

−fdump−rtl−postreload
  Dump after post−reload optimizations.

−fdump−rtl−pro_and_epilogue
  Dump after generating the function prologues and epilogues.

−fdump−rtl−sched1
−fdump−rtl−sched2
  −fdump−rtl−sched1 and **−fdump−rtl−sched2** enable dumping after the basic block scheduling passes.

−fdump−rtl−ree
  Dump after sign/zero extension elimination.

−fdump−rtl−seqabstr
  Dump after common sequence discovery.

−fdump−rtl−shorten
  Dump after shortening branches.

−fdump−rtl−split1
−fdump−rtl−split2
−fdump−rtl−split3
−fdump−rtl−split4
−fdump−rtl−split5
  These options enable dumping after five rounds of instruction splitting.

−fdump−rtl−sms
  Dump after modulo scheduling. This pass is only run on some architectures.

−fdump−rtl−stack
  Dump after conversion from GCC's "flat register file" registers to the x87's stack−like registers. This pass is only run on x86 variants.

−fdump−rtl−subreg1
−fdump−rtl−subreg2
  −fdump−rtl−subreg1 and **−fdump−rtl−subreg2** enable dumping after the two subreg expansion passes.

−fdump−rtl−vartrack
  Dump after variable tracking.

−fdump−rtl−vregs
  Dump after converting virtual registers to hard registers.

−fdump−rtl−web
  Dump after live range splitting.

−fdump−rtl−regclass
−fdump−rtl−subregs_of_mode_init
−fdump−rtl−subregs_of_mode_finish
−fdump−rtl−dfinit
−fdump−rtl−dfinish
  These dumps are defined but always produce empty files.

−da

−fdump−rtl−all

> Produce all the dumps listed above.

−dA

> Annotate the assembler output with miscellaneous debugging information.

−dD

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

−dH

> Produce a core dump whenever an error occurs.

−dp

> Annotate the assembler output with a comment indicating which pattern and alternative is used.
> The length and cost of each instruction are also printed.

−dP

> Dump the RTL in the assembler output as a comment before each instruction. Also turns on **−dp**
> annotation.

−dx

> Just generate RTL for a function instead of compiling it. Usually used with
> **−fdump−rtl−expand**.

−fdump−debug

> Dump debugging information generated during the debug generation phase.

−fdump−earlydebug

> Dump debugging information generated during the early debug generation phase.

−fdump−noaddr

> When doing debugging dumps, suppress address output. This makes it more feasible to use diff on
> debugging dumps for compiler invocations with different compiler binaries and/or different text / bss /
> data / heap / stack / dso start locations.

−freport−bug

> Collect and dump debug information into a temporary file if an internal compiler error (ICE) occurs.

−fdump−unnumbered

> When doing debugging dumps, suppress instruction numbers and address output. This makes it more
> feasible to use diff on debugging dumps for compiler invocations with different options, in particular
> with and without **−g**.

−fdump−unnumbered−links

> When doing debugging dumps (see **−d** option above), suppress instruction numbers for the links to the
> previous and next instructions in a sequence.

−fdump−ipa−*switch*

−fdump−ipa−*switch−options*

> Control the dumping at various stages of inter−procedural analysis language tree to a file. The file
> name is generated by appending a switch specific suffix to the source file name, and the file is created
> in the same directory as the output file. The following dumps are possible:

> **all**   Enables all inter−procedural analysis dumps.

> **cgraph**

>> Dumps information about call−graph optimization, unused function removal, and inlining
>> decisions.

> **inline**

>> Dump after function inlining.

strubm

Dump after selecting `strub` modes, and recording the selections as function attributes.

strub

Dump `strub` transformations: interface changes, function wrapping, and insertion of builtin calls for stack scrubbing and watermarking.

Additionally, the options **−optimized**, **−missed**, **−note**, and **−all** can be provided, with the same meaning as for **−fopt−info**, defaulting to **−optimized**.

For example, **−fdump−ipa−inline−optimized−missed** will emit information on callsites that were inlined, along with callsites that were not inlined.

By default, the dump will contain messages about successful optimizations (equivalent to **−optimized**) together with low−level details about the analysis.

−fdump−ipa−clones

Create a dump file containing information about creation of call graph node clones and removals of call graph nodes during inter−procedural optimizations and transformations. Its main intended use is that tools that create live−patches can determine the set of functions that need to be live−patched to completely replace a particular function (see **−flive−patching**). The file name is generated by appending suffix `ipa-clones` to the source file name, and the file is created in the same directory as the output file. Each entry in the file is on a separate line containing semicolon separated fields.

In the case of call graph clone creation, the individual fields are:

1. String `Callgraph clone`.

2. Name of the function being cloned as it is presented to the assembler.

3. A number that uniquely represents the function being cloned in the call graph. Note that the number is unique only within a compilation unit or within whole−program analysis but is likely to be different in the two phases.

4. The file name of the source file where the function is defined.

5. The line on which the function definition is located.

6. The column where the function definition is located.

7. Name of the new function clone as it is presented to the assembler.

8. A number that uniquely represents the new function clone in the call graph. Note that the number is unique only within a compilation unit or within whole−program analysis but is likely to be different in the two phases.

9. The file name of the source file where the source code location of the new clone points to.

10. The line to which the source code location of the new clone points to.

11. The column to which the source code location of the new clone points to.

12. A string that determines the reason for cloning.

In the case of call graph clone removal, the individual fields are:

1. String `Callgraph removal`.

2. Name of the function being removed as it would be presented to the assembler.

3. A number that uniquely represents the function being cloned in the call graph. Note that the number is unique only within a compilation unit or within whole−program analysis but is likely to be different in the two phases.

4. The file name of the source file where the function is defined.

5.   The line on which the function definition is located.

6.   The column where the function definition is located.

−fdump−lang

Dump language−specific information. The file name is made by appending *.lang* to the source file name.

−fdump−lang−all
−fdump−lang−*switch*
−fdump−lang−*switch−options*
−fdump−lang−*switch−options***=***filename*

Control the dumping of language−specific information. The *options* and *filename* portions behave as described in the **−fdump−tree** option. The following *switch* values are accepted:

all   Enable all language−specific dumps.

class

Dump class hierarchy information. Virtual table information is emitted unless '**slim**' is specified. This option is applicable to C++ only.

module

Dump module information. Options **lineno** (locations), **graph** (reachability), **blocks** (clusters), **uid** (serialization), **alias** (mergeable), **asmname** (Elrond), **eh** (mapper) & **vops** (macros) may provide additional information. This option is applicable to C++ only.

raw

Dump the raw internal tree data. This option is applicable to C++ only.

−fdump−passes

Print on *stderr* the list of optimization passes that are turned on and off by the current command−line options.

−fdump−statistics−*option*

Enable and control dumping of pass statistics in a separate file. The file name is generated by appending a suffix ending in **.statistics** to the source file name, and the file is created in the same directory as the output file. If the **−***option* form is used, **−stats** causes counters to be summed over the whole compilation unit while **−details** dumps every event as the passes generate them. The default with no option is to sum counters for each function compiled.

−fdump−tree−all
−fdump−tree−*switch*
−fdump−tree−*switch−options*
−fdump−tree−*switch−options***=***filename*

Control the dumping at various stages of processing the intermediate language tree to a file. If the **−***options* form is used, *options* is a list of **−** separated options which control the details of the dump. Not all options are applicable to all dumps; those that are not meaningful are ignored. The following options are available

address

Print the address of each node. Usually this is not meaningful as it changes according to the environment and source file. Its primary use is for tying up a dump file with a debug environment.

asmname

If DECL_ASSEMBLER_NAME has been set for a given decl, use that in the dump instead of DECL_NAME. Its primary use is ease of use working backward from mangled names in the assembly file.

slim

When dumping front−end intermediate representations, inhibit dumping of members of a scope or body of a function merely because that scope has been reached. Only dump such items when

they are directly reachable by some other path.

When dumping pretty−printed trees, this option inhibits dumping the bodies of control structures.

When dumping RTL, print the RTL in slim (condensed) form instead of the default LISP−like representation.

raw

Print a raw representation of the tree. By default, trees are pretty−printed into a C−like representation.

details

Enable more detailed dumps (not honored by every dump option). Also include information from the optimization passes.

stats

Enable dumping various statistics about the pass (not honored by every dump option).

blocks

Enable showing basic block boundaries (disabled in raw dumps).

graph

For each of the other indicated dump files (**−fdump−rtl−***pass*), dump a representation of the control flow graph suitable for viewing with GraphViz to *file.passid.pass.dot*. Each function in the file is pretty−printed as a subgraph, so that GraphViz can render them all in a single plot.

This option currently only works for RTL dumps, and the RTL is always dumped in slim form.

vops

Enable showing virtual operands for every statement.

lineno

Enable showing line numbers for statements.

uid   Enable showing the unique ID (DECL_UID) for each variable.

verbose

Enable showing the tree dump for each statement.

eh   Enable showing the EH region number holding each statement.

scev

Enable showing scalar evolution analysis details.

optimized

Enable showing optimization information (only available in certain passes).

missed

Enable showing missed optimization information (only available in certain passes).

note

Enable other detailed optimization information (only available in certain passes).

all   Turn on all options, except **raw**, **slim**, **verbose** and **lineno**.

optall

Turn on all optimization options, i.e., **optimized**, **missed**, and **note**.

To determine what tree dumps are available or find the dump for a pass of interest follow the steps below.

1.   Invoke GCC with **−fdump−passes** and in the *stderr* output look for a code that corresponds to the pass you are interested in. For example, the codes tree-evrp, tree-vrp1, and tree-vrp2 correspond to the three Value Range Propagation passes. The number at the end distinguishes distinct invocations of the same pass.

2.   To enable the creation of the dump file, append the pass code to the **–fdump–** option prefix and invoke GCC with it. For example, to enable the dump from the Early Value Range Propagation pass, invoke GCC with the **–fdump–tree–evrp** option. Optionally, you may specify the name of the dump file. If you don't specify one, GCC creates as described below.

3.   Find the pass dump in a file whose name is composed of three components separated by a period: the name of the source file GCC was invoked to compile, a numeric suffix indicating the pass number followed by the letter **t** for tree passes (and the letter **r** for RTL passes), and finally the pass code. For example, the Early VRP pass dump might be in a file named *myfile.c.038t.evrp* in the current working directory. Note that the numeric codes are not stable and may change from one version of GCC to another.

–fopt–info
–fopt–info–*options*
–fopt–info–*options***=***filename*
    Controls optimization dumps from various optimization passes. If the **–***options* form is used, *options* is a list of **–** separated option keywords to select the dump details and optimizations.

    The *options* can be divided into three groups:

    1.   options describing what kinds of messages should be emitted,

    2.   options describing the verbosity of the dump, and

    3.   options describing which optimizations should be included.

    The options from each group can be freely mixed as they are non–overlapping. However, in case of any conflicts, the later options override the earlier options on the command line.

    The following options control which kinds of messages should be emitted:

    optimized
        Print information when an optimization is successfully applied. It is up to a pass to decide which information is relevant. For example, the vectorizer passes print the source location of loops which are successfully vectorized.

    missed
        Print information about missed optimizations. Individual passes control which information to include in the output.

    note
        Print verbose information about optimizations, such as certain transformations, more detailed messages about decisions etc.

    all   Print detailed optimization information. This includes **optimized**, **missed**, and **note**.

    The following option controls the dump verbosity:

    internals
        By default, only "high–level" messages are emitted. This option enables additional, more detailed, messages, which are likely to only be of interest to GCC developers.

    One or more of the following option keywords can be used to describe a group of optimizations:

    ipa   Enable dumps from all interprocedural optimizations.

    loop
        Enable dumps from all loop optimizations.

    inline
        Enable dumps from all inlining optimizations.

    omp
        Enable dumps from all OMP (Offloading and Multi Processing) optimizations.

vec Enable dumps from all vectorization optimizations.

optall

Enable dumps from all optimizations. This is a superset of the optimization groups listed above.

If *options* is omitted, it defaults to **optimized−optall**, which means to dump messages about successful optimizations from all the passes, omitting messages that are treated as "internals".

If the *filename* is provided, then the dumps from all the applicable optimizations are concatenated into the *filename*. Otherwise the dump is output onto *stderr*. Though multiple **−fopt−info** options are accepted, only one of them can include a *filename*. If other filenames are provided then all but the first such option are ignored.

Note that the output *filename* is overwritten in case of multiple translation units. If a combined output from multiple translation units is desired, *stderr* should be used instead.

In the following example, the optimization info is output to *stderr*:

```
gcc −O3 −fopt-info
```

This example:

```
gcc −O3 −fopt-info-missed=missed.all
```

outputs missed optimization report from all the passes into *missed.all*, and this one:

```
gcc −O2 −ftree-vectorize −fopt-info-vec-missed
```

prints information about missed optimization opportunities from vectorization passes on *stderr*. Note that **−fopt−info−vec−missed** is equivalent to **−fopt−info−missed−vec**. The order of the optimization group names and message types listed after **−fopt−info** does not matter.

As another example,

```
gcc −O3 −fopt-info-inline-optimized-missed=inline.txt
```

outputs information about missed optimizations as well as optimized locations from all the inlining passes into *inline.txt*.

Finally, consider:

```
gcc −fopt-info-vec-missed=vec.miss −fopt-info-loop-optimized=loop.opt
```

Here the two output filenames *vec.miss* and *loop.opt* are in conflict since only one output file is allowed. In this case, only the first option takes effect and the subsequent options are ignored. Thus only *vec.miss* is produced which contains dumps from the vectorizer about missed opportunities.

−fsave−optimization−record

Write a SRCFILE.opt−record.json.gz file detailing what optimizations were performed, for those optimizations that support **−fopt−info**.

This option is experimental and the format of the data within the compressed JSON file is subject to change.

It is roughly equivalent to a machine−readable version of **−fopt−info−all**, as a collection of messages with source file, line number and column number, with the following additional data for each message:

*   the execution count of the code being optimized, along with metadata about whether this was from actual profile data, or just an estimate, allowing consumers to prioritize messages by code hotness,

*   the function name of the code being optimized, where applicable,

*   the "inlining chain" for the code being optimized, so that when a function is inlined into several different places (which might themselves be inlined), the reader can distinguish between the copies,

       *   objects identifying those parts of the message that refer to expressions, statements or symbol−table nodes, which of these categories they are, and, when available, their source code location,

       *   the GCC pass that emitted the message, and

       *   the location in GCC's own code from which the message was emitted

Additionally, some messages are logically nested within other messages, reflecting implementation details of the optimization passes.

−fsched−verbose=*n*
> On targets that use instruction scheduling, this option controls the amount of debugging output the scheduler prints to the dump files.
>
> For *n* greater than zero, **−fsched−verbose** outputs the same information as **−fdump−rtl−sched1** and **−fdump−rtl−sched2**. For *n* greater than one, it also output basic block probabilities, detailed ready list information and unit/insn info. For *n* greater than two, it includes RTL at abort point, control−flow and regions info. And for *n* over four, **−fsched−verbose** also includes dependence info.

−fenable−*kind***−***pass*
−fdisable−*kind***−***pass***=***range−list*
> This is a set of options that are used to explicitly disable/enable optimization passes. These options are intended for use for debugging GCC. Compiler users should use regular options for enabling/disabling passes instead.
>
> **−fdisable−ipa−***pass*
>> Disable IPA pass *pass*. *pass* is the pass name. If the same pass is statically invoked in the compiler multiple times, the pass name should be appended with a sequential number starting from 1.
>
> **−fdisable−rtl−***pass*
> **−fdisable−rtl−***pass***=***range−list*
>> Disable RTL pass *pass*. *pass* is the pass name. If the same pass is statically invoked in the compiler multiple times, the pass name should be appended with a sequential number starting from 1. *range−list* is a comma−separated list of function ranges or assembler names. Each range is a number pair separated by a colon. The range is inclusive in both ends. If the range is trivial, the number pair can be simplified as a single number. If the function's call graph node's *uid* falls within one of the specified ranges, the *pass* is disabled for that function. The *uid* is shown in the function header of a dump file, and the pass names can be dumped by using option **−fdump−passes**.
>
> **−fdisable−tree−***pass*
> **−fdisable−tree−***pass***=***range−list*
>> Disable tree pass *pass*. See **−fdisable−rtl** for the description of option arguments.
>
> **−fenable−ipa−***pass*
>> Enable IPA pass *pass*. *pass* is the pass name. If the same pass is statically invoked in the compiler multiple times, the pass name should be appended with a sequential number starting from 1.
>
> **−fenable−rtl−***pass*
> **−fenable−rtl−***pass***=***range−list*
>> Enable RTL pass *pass*. See **−fdisable−rtl** for option argument description and examples.
>
> **−fenable−tree−***pass*
> **−fenable−tree−***pass***=***range−list*
>> Enable tree pass *pass*. See **−fdisable−rtl** for the description of option arguments.

Here are some examples showing uses of these options.

```
# disable ccp1 for all functions
  -fdisable-tree-ccp1
# disable complete unroll for function whose cgraph node uid is 1
  -fenable-tree-cunroll=1
# disable gcse2 for functions at the following ranges [1,1],
# [300,400], and [400,1000]
# disable gcse2 for functions foo and foo2
  -fdisable-rtl-gcse2=foo,foo2
# disable early inlining
  -fdisable-tree-einline
# disable ipa inlining
  -fdisable-ipa-inline
# enable tree full unroll
  -fenable-tree-unroll
```

−fchecking
−fchecking=*n*
> Enable internal consistency checking. The default depends on the compiler configuration. **−fchecking=2** enables further internal consistency checking that might affect code generation.

−frandom−seed=*string*
> This option provides a seed that GCC uses in place of random numbers in generating certain symbol names that have to be different in every compiled file. It is also used to place unique stamps in coverage data files and the object files that produce them. You can use the **−frandom−seed** option to produce reproducibly identical object files.

> The *string* can either be a number (decimal, octal or hex) or an arbitrary string (in which case it's converted to a number by computing CRC32).

> The *string* should be different for every file you compile.

−save−temps
> Store the usual "temporary" intermediate files permanently; name them as auxiliary output files, as specified described under **−dumpbase** and **−dumpdir**.

> When used in combination with the **−x** command−line option, **−save−temps** is sensible enough to avoid overwriting an input source file with the same extension as an intermediate file. The corresponding intermediate file may be obtained by renaming the source file before using **−save−temps**.

−save−temps=cwd
> Equivalent to **−save−temps −dumpdir ./**.

−save−temps=obj
> Equivalent to **−save−temps −dumpdir** *outdir/*, where *outdir/* is the directory of the output file specified after the **−o** option, including any directory separators. If the **−o** option is not used, the **−save−temps=obj** switch behaves like **−save−temps=cwd**.

−time[**=***file*]
> Report the CPU time taken by each subprocess in the compilation sequence. For C source files, this is the compiler proper and assembler (plus the linker if linking is done).

> Without the specification of an output file, the output looks like this:

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

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

> With the specification of an output file, the output is appended to the named file, and it looks like this:

```
            0.12 0.01 cc1 <options>
            0.00 0.01 as <options>
```

The "user time" and the "system time" are moved before the program name, and the options passed to the program are displayed, so that one can later tell what file was being compiled, and with which options.

−fdump−final−insns[**=***file*]

   Dump the final internal representation (RTL) to *file*. If the optional argument is omitted (or if *file* is `.`), the name of the dump file is determined by appending `.gkd` to the dump base name, see **−dumpbase**.

−fcompare−debug[**=***opts*]

   If no error occurs during compilation, run the compiler a second time, adding *opts* and **−fcompare−debug−second** to the arguments passed to the second compilation. Dump the final internal representation in both compilations, and print an error if they differ.

   If the equal sign is omitted, the default **−gtoggle** is used.

   The environment variable **GCC_COMPARE_DEBUG**, if defined, non−empty and nonzero, implicitly enables **−fcompare−debug**. If **GCC_COMPARE_DEBUG** is defined to a string starting with a dash, then it is used for *opts*, otherwise the default **−gtoggle** is used.

   −fcompare−debug=, with the equal sign but without *opts*, is equivalent to **−fno−compare−debug**, which disables the dumping of the final representation and the second compilation, preventing even **GCC_COMPARE_DEBUG** from taking effect.

   To verify full coverage during **−fcompare−debug** testing, set **GCC_COMPARE_DEBUG** to say **−fcompare−debug−not−overridden**, which GCC rejects as an invalid option in any actual compilation (rather than preprocessing, assembly or linking). To get just a warning, setting **GCC_COMPARE_DEBUG** to **−w%n−fcompare−debug not overridden** will do.

−fcompare−debug−second

   This option is implicitly passed to the compiler for the second compilation requested by **−fcompare−debug**, along with options to silence warnings, and omitting other options that would cause the compiler to produce output to files or to standard output as a side effect. Dump files and preserved temporary files are renamed so as to contain the `.gk` additional extension during the second compilation, to avoid overwriting those generated by the first.

   When this option is passed to the compiler driver, it causes the *first* compilation to be skipped, which makes it useful for little other than debugging the compiler proper.

−gtoggle

   Turn off generation of debug info, if leaving out this option generates it, or turn it on at level 2 otherwise. The position of this argument in the command line does not matter; it takes effect after all other options are processed, and it does so only once, no matter how many times it is given. This is mainly intended to be used with **−fcompare−debug**.

−fvar−tracking−assignments−toggle

   Toggle **−fvar−tracking−assignments**, in the same way that **−gtoggle** toggles **−g**.

−Q   When used on the command line prior to **−−help=**, **−Q** acts as a modifier to the help output.

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

−ftime−report

   Makes the compiler print some statistics to stderr about the time consumed by each pass when it finishes.

   If SARIF output of diagnostics was requested via **−fdiagnostics−format=sarif−file** or **−fdiagnostics−format=sarif−stderr** then the **−ftime−report** information is instead emitted in JSON form as part of SARIF output. The precise format of this JSON data is subject to change, and the

values may not exactly match those emitted to stderr due to being written out at a slightly different place within the compiler.

−ftime−report−details

Record the time consumed by infrastructure parts separately for each pass.

−fira−verbose=*n*

Control the verbosity of the dump file for the integrated register allocator. The default value is 5. If the value *n* is greater or equal to 10, the dump output is sent to stderr using the same format as *n* minus 10.

−flto−report

Prints a report with internal details on the workings of the link−time optimizer. The contents of this report vary from version to version. It is meant to be useful to GCC developers when processing object files in LTO mode (via **−flto**).

Disabled by default.

−flto−report−wpa

Like **−flto−report**, but only print for the WPA phase of link−time optimization.

−fmem−report

Makes the compiler print some statistics about permanent memory allocation when it finishes.

−fmem−report−wpa

Makes the compiler print some statistics about permanent memory allocation for the WPA phase only.

−fpre−ipa−mem−report
−fpost−ipa−mem−report

Makes the compiler print some statistics about permanent memory allocation before or after interprocedural optimization.

−fmultiflags

This option enables multilib−aware `TFLAGS` to be used to build target libraries with options different from those the compiler is configured to use by default, through the use of specs set up by compiler internals, by the target, or by builders at configure time.

Like `TFLAGS`, this allows the target libraries to be built for portable baseline environments, while the compiler defaults to more demanding ones. That's useful because users can easily override the defaults the compiler is configured to use to build their own programs, if the defaults are not ideal for their target environment, whereas rebuilding the runtime libraries is usually not as easy or desirable.

Unlike `TFLAGS`, the use of specs enables different flags to be selected for different multilibs. The way to accomplish that is to build with **make TFLAGS=−fmultiflags**, after configuring **−−with−specs=%{fmultiflags:...}**.

This option is discarded by the driver once it's done processing driver self spec.

It is also useful to check that `TFLAGS` are being used to build all target libraries, by configuring a non−bootstrap compiler **−−with−specs='%{!fmultiflags:%emissing TFLAGS}'** and building the compiler and target libraries.

−fprofile−report

Makes the compiler print some statistics about consistency of the (estimated) profile and effect of individual passes.

−fstack−usage

Makes the compiler output stack usage information for the program, on a per−function basis. The filename for the dump is made by appending *.su* to the *auxname*. *auxname* is generated from the name of the output file, if explicitly specified and it is not an executable, otherwise it is the basename of the source file. An entry is made up of three fields:

* The name of the function.

* A number of bytes.

* One or more qualifiers: static, dynamic, bounded.

The qualifier static means that the function manipulates the stack statically: a fixed number of bytes are allocated for the frame on function entry and released on function exit; no stack adjustments are otherwise made in the function. The second field is this fixed number of bytes.

The qualifier dynamic means that the function manipulates the stack dynamically: in addition to the static allocation described above, stack adjustments are made in the body of the function, for example to push/pop arguments around function calls. If the qualifier bounded is also present, the amount of these adjustments is bounded at compile time and the second field is an upper bound of the total amount of stack used by the function. If it is not present, the amount of these adjustments is not bounded at compile time and the second field only represents the bounded part.

−fstats

Emit statistics about front−end processing at the end of the compilation. This option is supported only by the C++ front end, and the information is generally only useful to the G++ development team.

−fdbg−cnt−list

Print the name and the counter upper bound for all debug counters.

−fdbg−cnt=*counter−value−list*

Set the internal debug counter lower and upper bound. *counter−value−list* is a comma−separated list of *name*:*lower_bound1−upper_bound1* [:*lower_bound2−upper_bound2*...] tuples which sets the name of the counter and list of closed intervals. The *lower_bound* is optional and is zero initialized if not set. For example, with **−fdbg−cnt=dce:2−4:10−11,tail_call:10**, dbg_cnt(dce) returns true only for second, third, fourth, tenth and eleventh invocation. For dbg_cnt(tail_call) true is returned for first 10 invocations.

−print−file−name=*library*

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

−print−multi−directory

Print the directory name corresponding to the multilib selected by any other switches present in the command line. This directory is supposed to exist in **GCC_EXEC_PREFIX**.

−print−multi−lib

Print the mapping from multilib directory names to compiler switches that enable them. The directory name is separated from the switches by **;**, and each switch starts with an **@** instead of the **−**, without spaces between multiple switches. This is supposed to ease shell processing.

−print−multi−os−directory

Print the path to OS libraries for the selected multilib, relative to some *lib* subdirectory. If OS libraries are present in the *lib* subdirectory and no multilibs are used, this is usually just ., if OS libraries are present in *libsuffix* sibling directories this prints e.g. *../lib64*, *../lib* or *../lib32*, or if OS libraries are present in *lib/subdir* subdirectories it prints e.g. *amd64*, *sparcv9* or *ev6*.

−print−multiarch

Print the path to OS libraries for the selected multiarch, relative to some *lib* subdirectory.

−print−prog−name=*program*

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

−print−libgcc−file−name

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

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

```
                  gcc −nostdlib <files>... `gcc −print−libgcc−file−name`
```

−print−search−dirs

Print the name of the configured installation directory and a list of program and library directories **gcc** searches−−−and don't do anything else.

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

−print−sysroot

Print the target sysroot directory that is used during compilation. This is the target sysroot specified either at configure time or using the **−−sysroot** option, possibly with an extra suffix that depends on compilation options. If no target sysroot is specified, the option prints nothing.

−print−sysroot−headers−suffix

Print the suffix added to the target sysroot when searching for headers, or give an error if the compiler is not configured with such a suffix−−−and don't do anything else.

−dumpmachine

Print the compiler's target machine (for example, **i686−pc−linux−gnu**)−−−and don't do anything else.

−dumpversion

Print the compiler version (for example, 3.0, 6.3.0 or 7)−−−and don't do anything else. This is the compiler version used in filesystem paths and specs. Depending on how the compiler has been configured it can be just a single number (major version), two numbers separated by a dot (major and minor version) or three numbers separated by dots (major, minor and patchlevel version).

−dumpfullversion

Print the full compiler version−−−and don't do anything else. The output is always three numbers separated by dots, major, minor and patchlevel version.

−dumpspecs

Print the compiler's built−in specs−−−and don't do anything else. (This is used when GCC itself is being built.)

Machine−Dependent Options

Each target machine supported by GCC can have its own options−−−for example, to allow you to compile for a particular processor variant or ABI, or to control optimizations specific to that machine. By convention, the names of machine−specific options start with **−m**.

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

AArch64 Options

These options are defined for AArch64 implementations:

−mabi=*name*

Generate code for the specified data model. Permissible values are **ilp32** for SysV−like data model where int, long int and pointers are 32 bits, and **lp64** for SysV−like data model where int is 32 bits, but long int and pointers are 64 bits.

The default depends on the specific target configuration. Note that the LP64 and ILP32 ABIs are not link−compatible; you must compile your entire program with the same ABI, and link with a compatible set of libraries.

The **ilp32** model is deprecated.

−mbig−endian

Generate big−endian code. This is the default when GCC is configured for an **aarch64_be−*−*** target.

−mgeneral−regs−only

Generate code which uses only the general−purpose registers. This will prevent the compiler from using floating−point and Advanced SIMD registers but will not impose any restrictions on the assembler.

−mlittle−endian

Generate little−endian code. This is the default when GCC is configured for an **aarch64−*−*** but not an **aarch64_be−*−*** target.

−mcmodel=tiny

Generate code for the tiny code model. The program and its statically defined symbols must be within 1MB of each other. Programs can be statically or dynamically linked.

−mcmodel=small

Generate code for the small code model. The program and its statically defined symbols must be within 4GB of each other. Programs can be statically or dynamically linked. This is the default code model.

−mcmodel=large

Generate code for the large code model. This makes no assumptions about addresses and sizes of sections. Programs can be statically linked only. The **−mcmodel=large** option is incompatible with **−mabi=ilp32**, **−fpic** and **−fPIC**.

−mtp=*name*

Specify the system register to use as a thread pointer. The valid values are **tpidr_el0**, **tpidrro_el0**, **tpidr_el1**, **tpidr_el2**, **tpidr_el3**. For backwards compatibility the aliases **el0**, **el1**, **el2**, **el3** are also accepted. The default setting is **tpidr_el0**. It is recommended to compile all code intended to interoperate with the same value of this option to avoid accessing a different thread pointer from the wrong exception level.

−mstrict−align
−mno−strict−align

Avoid or allow generating memory accesses that may not be aligned on a natural object boundary as described in the architecture specification.

−momit−leaf−frame−pointer
−mno−omit−leaf−frame−pointer

Omit or keep the frame pointer in leaf functions. The former behavior is the default.

−mstack−protector−guard=*guard*
−mstack−protector−guard−reg=*reg*
−mstack−protector−guard−offset=*offset*

Generate stack protection code using canary at *guard*. Supported locations are **global** for a global canary or **sysreg** for a canary in an appropriate system register.

With the latter choice the options **−mstack−protector−guard−reg=***reg* and **−mstack−protector−guard−offset=***offset* furthermore specify which system register to use as base register for reading the canary, and from what offset from that base register. There is no default register or offset as this is entirely for use within the Linux kernel.

−mtls−dialect=desc

Use TLS descriptors as the thread−local storage mechanism for dynamic accesses of TLS variables. This is the default.

−mtls−dialect=traditional

Use traditional TLS as the thread−local storage mechanism for dynamic accesses of TLS variables.

−mtls−size=*size*

Specify bit size of immediate TLS offsets. Valid values are 12, 24, 32, 48. This option requires binutils 2.26 or newer.

−mfix−cortex−a53−835769

−mno−fix−cortex−a53−835769

> Enable or disable the workaround for the ARM Cortex−A53 erratum number 835769. This involves inserting a NOP instruction between memory instructions and 64−bit integer multiply−accumulate instructions. This flag will be ignored if an architecture or cpu is specified on the command line which does not need the workaround.

−mfix−cortex−a53−843419

−mno−fix−cortex−a53−843419

> Enable or disable the workaround for the ARM Cortex−A53 erratum number 843419. This erratum workaround is made at link time and this will only pass the corresponding flag to the linker. This flag will be ignored if an architecture or cpu is specified on the command line which does not need the workaround.

−mlow−precision−recip−sqrt

−mno−low−precision−recip−sqrt

> Enable or disable the reciprocal square root approximation. This option only has an effect if **−ffast−math** or **−funsafe−math−optimizations** is used as well. Enabling this reduces precision of reciprocal square root results to about 16 bits for single precision and to 32 bits for double precision.

−mlow−precision−sqrt

−mno−low−precision−sqrt

> Enable or disable the square root approximation. This option only has an effect if **−ffast−math** or **−funsafe−math−optimizations** is used as well. Enabling this reduces precision of square root results to about 16 bits for single precision and to 32 bits for double precision. If enabled, it implies **−mlow−precision−recip−sqrt**.

−mlow−precision−div

−mno−low−precision−div

> Enable or disable the division approximation. This option only has an effect if **−ffast−math** or **−funsafe−math−optimizations** is used as well. Enabling this reduces precision of division results to about 16 bits for single precision and to 32 bits for double precision.

−mtrack−speculation

−mno−track−speculation

> Enable or disable generation of additional code to track speculative execution through conditional branches. The tracking state can then be used by the compiler when expanding calls to `__builtin_speculation_safe_copy` to permit a more efficient code sequence to be generated.

−moutline−atomics

−mno−outline−atomics

> Enable or disable calls to out−of−line helpers to implement atomic operations. These helpers will, at runtime, determine if the LSE instructions from ARMv8.1−A can be used; if not, they will use the load/store−exclusive instructions that are present in the base ARMv8.0 ISA.
>
> This option is only applicable when compiling for the base ARMv8.0 instruction set. If using a later revision, e.g. **−march=armv8.1−a** or **−march=armv8−a+lse**, the ARMv8.1−Atomics instructions will be used directly. The same applies when using **−mcpu=** when the selected cpu supports the **lse** feature. This option is on by default.

−march=_name_

> Specify the name of the target architecture and, optionally, one or more feature modifiers. This option has the form **−march=**_arch_{**+**[**no**]_feature_}*.
>
> The table below summarizes the permissible values for _arch_ and the features that they enable by default:

arch value : *Architecture* : *Includes by default*
armv8−a : Armv8−A : **+fp**, **+simd**
armv8.1−a : Armv8.1−A : **armv8−a**, **+crc**, **+lse**, **+rdma**
armv8.2−a : Armv8.2−A : **armv8.1−a**
armv8.3−a : Armv8.3−A : **armv8.2−a**, **+pauth**, **+fcma**, **+jscvt**
armv8.4−a : Armv8.4−A : **armv8.3−a**, **+flagm**, **+fp16fml**, **+dotprod**, **+rcpc2**
armv8.5−a : Armv8.5−A : **armv8.4−a**, **+sb**, **+ssbs**, **+predres**, **+frintts**, **+flagm2**
armv8.6−a : Armv8.6−A : **armv8.5−a**, **+bf16**, **+i8mm**
armv8.7−a : Armv8.7−A : **armv8.6−a**, **+wfxt**, **+xs**
armv8.8−a : Armv8.8−a : **armv8.7−a**, **+mops**
armv8.9−a : Armv8.9−a : **armv8.8−a**
armv9−a : Armv9−A : **armv8.5−a**, **+sve**, **+sve2**
armv9.1−a : Armv9.1−A : **armv9−a**, **+bf16**, **+i8mm**
armv9.2−a : Armv9.2−A : **armv9.1−a**, **+wfxt**, **+xs**
armv9.3−a : Armv9.3−A : **armv9.2−a**, **+mops**
armv9.4−a : Armv9.4−A : **armv9.3−a**, **+sve2p1**
armv9.5−a : Armv9.4−A : **armv9.4−a**, **cpa**, **+faminmax**, **+lut**

armv8−r : Armv8−R : **armv8−r**

The value **native** is available on native AArch64 GNU/Linux and causes the compiler to pick the architecture of the host system. This option has no effect if the compiler is unable to recognize the architecture of the host system. When **−march=native** is given and no other **−mcpu** or **−mtune** is given then GCC will pick the host CPU as the CPU to tune for as well as select the architecture features from. That is, **−march=native** is treated as **−mcpu=native**.

The permissible values for *feature* are listed in the sub−section on **aarch64−feature−modifiers,,−march and −mcpu Feature Modifiers**. Where conflicting feature modifiers are specified, the right−most feature is used.

GCC uses *name* to determine what kind of instructions it can emit when generating assembly code. If **−march** is specified without either of **−mtune** or **−mcpu** also being specified, the code is tuned to perform well across a range of target processors implementing the target architecture.

−mtune=*name*
Specify the name of the target processor for which GCC should tune the performance of the code. Permissible values for this option are: **generic**, **cortex−a35**, **cortex−a53**, **cortex−a55**, **cortex−a57**, **cortex−a72**, **cortex−a73**, **cortex−a75**, **cortex−a76**, **cortex−a76ae**, **cortex−a77**, **cortex−a65**, **cortex−a65ae**, **cortex−a34**, **cortex−a78**, **cortex−a78ae**, **cortex−a78c**, **ares**, **exynos−m1**, **emag**, **falkor**, **oryon−1**, **neoverse−512tvb**, **neoverse−e1**, **neoverse−n1**, **neoverse−n2**, **neoverse−v1**, **neoverse−v2**, **grace**, **neoverse−v3**, **neoverse−v3ae**, **neoverse−n3**, **olympus**, **cortex−a725**, **cortex−x925**, **qdf24xx**, **saphira**, **phecda**, **xgene1**, **vulcan**, **octeontx**, **octeontx81**, **octeontx83**, **octeontx2**, **octeontx2t98**, **octeontx2t96** **octeontx2t93**, **octeontx2f95**, **octeontx2f95n**, **octeontx2f95mm**, **a64fx**, **fujitsu−monaka**, **thunderx**, **thunderxt88**, **thunderxt88p1**, **thunderxt81**, **tsv110**, **thunderxt83**, **thunderx2t99**, **thunderx3t110**, **zeus**, **cortex−a57.cortex−a53**, **cortex−a72.cortex−a53**, **cortex−a73.cortex−a35**, **cortex−a73.cortex−a53**, **cortex−a75.cortex−a55**, **cortex−a76.cortex−a55**, **cortex−r82**, **cortex−r82ae**, **cortex−x1**, **cortex−x1c**, **cortex−x2**, **cortex−x3**, **cortex−x4**, **cortex−a510**, **cortex−a520**, **cortex−a520ae**, **cortex−a710**, **cortex−a715**, **cortex−a720**, **cortex−a720ae**, **ampere1**, **ampere1a**, **ampere1b**, **cobalt−100**, **apple−m1**, **apple−m2**, **apple−m3** and **native**.

The values **cortex−a57.cortex−a53**, **cortex−a72.cortex−a53**, **cortex−a73.cortex−a35**, **cortex−a73.cortex−a53**, **cortex−a75.cortex−a55**, **cortex−a76.cortex−a55**, **apple−m1**, **apple−m2**, **apple−m3** specify that GCC should tune for a big.LITTLE system.

The value **neoverse−512tvb** specifies that GCC should tune for Neoverse cores that (a) implement SVE and (b) have a total vector bandwidth of 512 bits per cycle. In other words, the option tells GCC to tune for Neoverse cores that can execute 4 128−bit Advanced SIMD arithmetic instructions a cycle and that can execute an equivalent number of SVE arithmetic instructions per cycle (2 for 256−bit

SVE, 4 for 128−bit SVE). This is more general than tuning for a specific core like Neoverse V1 but is more specific than the default tuning described below.

Additionally on native AArch64 GNU/Linux systems the value **native** tunes performance to the host system. This option has no effect if the compiler is unable to recognize the processor of the host system.

Where none of **−mtune=**, **−mcpu=** or **−march=** are specified, the code is tuned to perform well across a range of target processors.

This option cannot be suffixed by feature modifiers.

−mcpu=*name*
> Specify the name of the target processor, optionally suffixed by one or more feature modifiers. This option has the form **−mcpu=***cpu*{**+**[**no**]*feature*}*, where the permissible values for *cpu* are the same as those available for **−mtune**. The permissible values for *feature* are documented in the sub−section on **aarch64−feature−modifiers,,−march and −mcpu Feature Modifiers**. Where conflicting feature modifiers are specified, the right−most feature is used.
>
> GCC uses *name* to determine what kind of instructions it can emit when generating assembly code (as if by **−march**) and to determine the target processor for which to tune for performance (as if by **−mtune**). Where this option is used in conjunction with **−march** or **−mtune**, those options take precedence over the appropriate part of this option.
>
> **−mcpu=neoverse−512tvb** is special in that it does not refer to a specific core, but instead refers to all Neoverse cores that (a) implement SVE and (b) have a total vector bandwidth of 512 bits a cycle. Unless overridden by **−march**, **−mcpu=neoverse−512tvb** generates code that can run on a Neoverse V1 core, since Neoverse V1 is the first Neoverse core with these properties. Unless overridden by **−mtune**, **−mcpu=neoverse−512tvb** tunes code in the same way as for **−mtune=neoverse−512tvb**.

−moverride=*string*
> Override tuning decisions made by the back−end in response to a **−mtune=** switch. The syntax, semantics, and accepted values for *string* in this option are not guaranteed to be consistent across releases.
>
> This option is only intended to be useful when developing GCC.

−mverbose−cost−dump
> Enable verbose cost model dumping in the debug dump files. This option is provided for use in debugging the compiler.

−mpc−relative−literal−loads
−mno−pc−relative−literal−loads
> Enable or disable PC−relative literal loads. With this option literal pools are accessed using a single instruction and emitted after each function. This limits the maximum size of functions to 1MB. This is enabled by default for **−mcmodel=tiny**.

−msign−return−address=*scope*
> Select the function scope on which return address signing will be applied. Permissible values are **none**, which disables return address signing, **non−leaf**, which enables pointer signing for functions which are not leaf functions, and **all**, which enables pointer signing for all functions. The default value is **none**. This option has been deprecated by −mbranch−protection.

−mbranch−protection=*none|standard|pac−ret*[**+***leaf***+***b−key*]|*bti*|*gcs*
> Select the branch protection features to use. **none** is the default and turns off all types of branch protection. **standard** turns on all types of branch protection features. If a feature has additional tuning options, then **standard** sets it to its standard level. **pac−ret**[**+***leaf*] turns on return address signing to its standard level: signing functions that save the return address to memory (non−leaf functions will practically always do this) using the a−key. The optional argument **leaf** can be used to extend the signing to include leaf functions. The optional argument **b−key** can be used to sign the functions with the B−key instead of the A−key. **bti** turns on branch target identification mechanism.

gcs turns on guarded control stack compatible code generation.

−mharden−sls=_opts_

Enable compiler hardening against straight line speculation (SLS). *opts* is a comma−separated list of the following options:

retbr
blr

In addition, **−mharden−sls=all** enables all SLS hardening while **−mharden−sls=none** disables all SLS hardening.

−mearly−ra=_scope_

Determine when to enable an early register allocation pass. This pass runs before instruction scheduling and tries to find a spill−free allocation of floating−point and vector code. It also tries to make use of strided multi−register instructions, such as SME2's strided LD1 and ST1.

The possible values of *scope* are: *all*, which runs the pass on all functions; *strided*, which runs the pass on functions that have access to strided multi−register instructions; and *none*, which disables the pass.

−mearly−ra=all is the default for **−O2** and above, and for **−Os**. **−mearly−ra=none** is the default otherwise.

−mearly−ldp−fusion

Enable the copy of the AArch64 load/store pair fusion pass that runs before register allocation. Enabled by default at **−O** and above.

−mlate−ldp−fusion

Enable the copy of the AArch64 load/store pair fusion pass that runs after register allocation. Enabled by default at **−O** and above.

−msve−vector−bits=_bits_

Specify the number of bits in an SVE vector register. This option only has an effect when SVE is enabled.

GCC supports two forms of SVE code generation: "vector−length agnostic" output that works with any size of vector register and "vector−length specific" output that allows GCC to make assumptions about the vector length when it is useful for optimization reasons. The possible values of **bits** are: **scalable**, **128**, **256**, **512**, **1024** and **2048**. Specifying **scalable** selects vector−length agnostic output. At present **−msve−vector−bits=128** also generates vector−length agnostic output for big−endian targets. All other values generate vector−length specific code. The behavior of these values may change in future releases and no value except **scalable** should be relied on for producing code that is portable across different hardware SVE vector lengths.

The default is **−msve−vector−bits=scalable**, which produces vector−length agnostic code.

−Wexperimental−fmv−target

Warn about use of experimental Function Multi Versioning. The Arm C Language Extension specification for Function Multi Versioning is beta and subject to change. Any usage of FMV is caveated that future behavior change and incompatibility is likely.

−march and **−mcpu** Feature Modifiers

Feature modifiers used with **−march** and **−mcpu** can be any of the following and their inverses **no**_feature_:

crc   Enable CRC extension. This is on by default for **−march=armv8.1−a**.

crypto

Enable Crypto extension. This also enables Advanced SIMD and floating−point instructions.

fp   Enable floating−point instructions. This is on by default for all possible values for options **−march** and **−mcpu**.

simd
  Enable Advanced SIMD instructions. This also enables floating−point instructions. This is on by default for all possible values for options **−march** and **−mcpu**.

sve Enable Scalable Vector Extension instructions. This also enables Advanced SIMD and floating−point instructions.

lse Enable Large System Extension instructions. This is on by default for **−march=armv8.1−a**.

rdma
  Enable Round Double Multiply Accumulate instructions. This is on by default for **−march=armv8.1−a**.

fp16
  Enable FP16 extension. This also enables floating−point instructions.

fp16fml
  Enable FP16 fmla extension. This also enables FP16 extensions and floating−point instructions. This option is enabled by default for **−march=armv8.4−a**. Use of this option with architectures prior to Armv8.2−A is not supported.

rcpc
  Enable the RCpc extension. This enables the use of the LDAPR instructions for load−acquire atomic semantics, and passes it on to the assembler, enabling inline asm statements to use instructions from the RCpc extension.

dotprod
  Enable the Dot Product extension. This also enables Advanced SIMD instructions.

aes Enable the Armv8−a aes and pmull crypto extension. This also enables Advanced SIMD instructions.

sha2
  Enable the Armv8−a sha2 crypto extension. This also enables Advanced SIMD instructions.

sha3
  Enable the sha512 and sha3 crypto extension. This also enables Advanced SIMD instructions. Use of this option with architectures prior to Armv8.2−A is not supported.

sm4
  Enable the sm3 and sm4 crypto extension. This also enables Advanced SIMD instructions. Use of this option with architectures prior to Armv8.2−A is not supported.

profile
  Enable the Statistical Profiling extension. This option is only to enable the extension at the assembler level and does not affect code generation.

rng Enable the Armv8.5−a Random Number instructions. This option is only to enable the extension at the assembler level and does not affect code generation.

memtag
  Enable the Armv8.5−a Memory Tagging Extensions. Use of this option with architectures prior to Armv8.5−A is not supported.

sb Enable the Armv8−a Speculation Barrier instruction. This option is only to enable the extension at the assembler level and does not affect code generation. This option is enabled by default for **−march=armv8.5−a**.

ssbs
  Enable the Armv8−a Speculative Store Bypass Safe instruction. This option is only to enable the extension at the assembler level and does not affect code generation. This option is enabled by default for **−march=armv8.5−a**.

predres
  Enable the Armv8−a Execution and Data Prediction Restriction instructions. This option is only to enable the extension at the assembler level and does not affect code generation. This option is enabled

by default for **−march=armv8.5−a**.

sve2

Enable the Armv8−a Scalable Vector Extension 2. This also enables SVE instructions.

sve2−bitperm

Enable SVE2 bitperm instructions. This also enables SVE2 instructions.

sve2−sm4

Enable SVE2 sm4 instructions. This also enables SVE2 instructions.

sve2−aes

Enable SVE2 aes instructions. This also enables SVE2 instructions.

sve2−sha3

Enable SVE2 sha3 instructions. This also enables SVE2 instructions.

sve2p1

Enable SVE2.1 instructions. This also enables SVE2 instructions.

tme

Enable the Transactional Memory Extension.

i8mm

Enable 8−bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating−point instructions. This option is enabled by default for **−march=armv8.6−a**. Use of this option with architectures prior to Armv8.2−A is not supported.

f32mm

Enable 32−bit Floating point Matrix Multiply instructions. This also enables SVE instructions. Use of this option with architectures prior to Armv8.2−A is not supported.

f64mm

Enable 64−bit Floating point Matrix Multiply instructions. This also enables SVE instructions. Use of this option with architectures prior to Armv8.2−A is not supported.

bf16

Enable brain half−precision floating−point instructions. This also enables Advanced SIMD and floating−point instructions. This option is enabled by default for **−march=armv8.6−a**. Use of this option with architectures prior to Armv8.2−A is not supported.

ls64

Enable the 64−byte atomic load and store instructions for accelerators.

mops

Enable the instructions to accelerate memory operations like `memcpy`, `memmove`, `memset`. This option is enabled by default for **−march=armv8.8−a**

flagm

Enable the Flag Manipulation instructions Extension.

flagm2

Enable the FlagM2 flag conversion instructions.

pauth

Enable the Pointer Authentication Extension.

cssc

Enable the Common Short Sequence Compression instructions.

sme

Enable the Scalable Matrix Extension. This is only supported when SVE2 is also enabled.

sme−i16i64

Enable the FEAT_SME_I16I64 extension to SME. This also enables SME instructions.

sme−f64f64
Enable the FEAT_SME_F64F64 extension to SME. This also enables SME instructions.

sme2
Enable the Scalable Matrix Extension 2. This also enables SME instructions.

sme−b16b16
Enable the FEAT_SME_B16B16 extension to SME. This also enables SME2 and SVE_B16B16 instructions.

sme−f16f16
Enable the FEAT_SME_F16F16 extension to SME. This also enables SME2 instructions.

sme2p1
Enable the Scalable Matrix Extension version 2.1. This also enables SME2 instructions.

fcma
Enable the complex number SIMD extensions.

jscvt
Enable the `fjcvtzs` JavaScript conversion instruction.

frintts
Enable floating−point round to integral value instructions.

wfxt
Enable `wfet` and `wfit` instructions.

xs   Enable the XS memory attribute extension.

lse128
Enable the LSE128 128−bit atomic instructions extension. This also enables LSE instructions.

d128
Enable support for 128−bit system register read/write instructions. This also enables the LSE128 extension.

gcs   Enable support for Armv9.4−a Guarded Control Stack extension.

the   Enable support for Armv8.9−a/9.4−a translation hardening extension.

rcpc2
Enable the RCpc2 extension.

rcpc3
Enable the RCpc3 (Release Consistency) extension.

fp8   Enable the fp8 (8−bit floating point) extension.

fp8fma
Enable the fp8 (8−bit floating point) multiply accumulate extension.

ssve−fp8fma
Enable the fp8 (8−bit floating point) multiply accumulate extension in streaming mode.

fp8dot4
Enable the fp8 (8−bit floating point) to single−precision 4−way dot product extension.

ssve−fp8dot4
Enable the fp8 (8−bit floating point) to single−precision 4−way dot product extension in streaming mode.

fp8dot2
Enable the fp8 (8−bit floating point) to half−precision 2−way dot product extension.

ssve−fp8dot2
Enable the fp8 (8−bit floating point) to half−precision 2−way dot product extension in streaming mode.

faminmax
> Enable the Floating Point Absolute Maximum/Minimum extension.

lut   Enable the Lookup Table extension.

cpa   Enable the Checked Pointer Arithmetic instructions.

sve−b16b16
> Enable the SVE non−widening brain floating−point (bf16) extension. This only has an effect when sve2 or sme2 are also enabled.

Feature **crypto** implies **aes**, **sha2**, and **simd**, which implies **fp**. Conversely, **nofp** implies **nosimd**, which implies **nocrypto**, **noaes** and **nosha2**.

Adapteva Epiphany Options
These **−m** options are defined for Adapteva Epiphany:

−mhalf−reg−file
> Don't allocate any register in the range r32...r63. That allows code to run on hardware variants that lack these registers.

−mprefer−short−insn−regs
> Preferentially allocate registers that allow short instruction generation. This can result in increased instruction count, so this may either reduce or increase overall code size.

−mbranch−cost=_num_
> Set the cost of branches to roughly _num_ "simple" instructions. This cost is only a heuristic and is not guaranteed to produce consistent results across releases.

−mcmove
> Enable the generation of conditional moves.

−mnops=_num_
> Emit _num_ NOPs before every other generated instruction.

−mno−soft−cmpsf
> For single−precision floating−point comparisons, emit an fsub instruction and test the flags. This is faster than a software comparison, but can get incorrect results in the presence of NaNs, or when two different small numbers are compared such that their difference is calculated as zero. The default is **−msoft−cmpsf**, which uses slower, but IEEE−compliant, software comparisons.

−mstack−offset=_num_
> Set the offset between the top of the stack and the stack pointer. E.g., a value of 8 means that the eight bytes in the range sp+0...sp+7 can be used by leaf functions without stack allocation. Values other than **8** or **16** are untested and unlikely to work. Note also that this option changes the ABI; compiling a program with a different stack offset than the libraries have been compiled with generally does not work. This option can be useful if you want to evaluate if a different stack offset would give you better code, but to actually use a different stack offset to build working programs, it is recommended to configure the toolchain with the appropriate **−−with−stack−offset=**_num_ option.

−mno−round−nearest
> Make the scheduler assume that the rounding mode has been set to truncating. The default is **−mround−nearest**.

−mlong−calls
> If not otherwise specified by an attribute, assume all calls might be beyond the offset range of the b / bl instructions, and therefore load the function address into a register before performing a (otherwise direct) call. This is the default.

−mshort−calls
> If not otherwise specified by an attribute, assume all direct calls are in the range of the b / bl instructions, so use these instructions for direct calls. The default is **−mlong−calls**.

−msmall16

Assume addresses can be loaded as 16−bit unsigned values. This does not apply to function addresses for which **−mlong−calls** semantics are in effect.

−mfp−mode=*mode*

Set the prevailing mode of the floating−point unit. This determines the floating−point mode that is provided and expected at function call and return time. Making this mode match the mode you predominantly need at function start can make your programs smaller and faster by avoiding unnecessary mode switches.

mode can be set to one the following values:

caller

Any mode at function entry is valid, and retained or restored when the function returns, and when it calls other functions. This mode is useful for compiling libraries or other compilation units you might want to incorporate into different programs with different prevailing FPU modes, and the convenience of being able to use a single object file outweighs the size and speed overhead for any extra mode switching that might be needed, compared with what would be needed with a more specific choice of prevailing FPU mode.

truncate

This is the mode used for floating−point calculations with truncating (i.e. round towards zero) rounding mode. That includes conversion from floating point to integer.

round−nearest

This is the mode used for floating−point calculations with round−to−nearest−or−even rounding mode.

int This is the mode used to perform integer calculations in the FPU, e.g. integer multiply, or integer multiply−and−accumulate.

The default is **−mfp−mode=caller**

−mno−split−lohi
−mno−postinc
−mno−postmodify

Code generation tweaks that disable, respectively, splitting of 32−bit loads, generation of post−increment addresses, and generation of post−modify addresses. The defaults are **msplit−lohi**, **−mpost−inc**, and **−mpost−modify**.

−mnovect−double

Change the preferred SIMD mode to SImode. The default is **−mvect−double**, which uses DImode as preferred SIMD mode.

−max−vect−align=*num*

The maximum alignment for SIMD vector mode types. *num* may be 4 or 8. The default is 8. Note that this is an ABI change, even though many library function interfaces are unaffected if they don't use SIMD vector modes in places that affect size and/or alignment of relevant types.

−msplit−vecmove−early

Split vector moves into single word moves before reload. In theory this can give better register allocation, but so far the reverse seems to be generally the case.

−m1reg−*reg*

Specify a register to hold the constant −1, which makes loading small negative constants and certain bitmasks faster. Allowable values for *reg* are **r43** and **r63**, which specify use of that register as a fixed register, and **none**, which means that no register is used for this purpose. The default is **−m1reg−none**.

AMD GCN Options

These options are defined specifically for the AMD GCN port.

−march=*gpu*
−mtune=*gpu*
> Set architecture type or tuning for *gpu*. Supported values for *gpu* are

> **gfx900**
>> Compile for GCN5 Vega 10 devices (gfx900).

> **gfx902**
>> Compile for GCN5 Vega gfx902 devices. (Experimental)

> **gfx904**
>> Compile for GCN5 Vega gfx904 devices. (Experimental)

> **gfx906**
>> Compile for GCN5 Vega 20 devices (gfx906).

> **gfx908**
>> Compile for CDNA1 Instinct MI100 series devices (gfx908).

> **gfx909**
>> Compile for GCN5 Vega gfx909 devices. (Experimental)

> **gfx90a**
>> Compile for CDNA2 Instinct MI200 series devices (gfx90a).

> **gfx90c**
>> Compile for GCN5 Vega 7 devices (gfx90c).

> **gfx9−generic**
>> Compile generic code for Vega devices, executable on the following subset of GFX9 devices: gfx900, gfx902, gfx904, gfx906, gfx909 and gfx90c. (Experimental)

> **gfx1030**
>> Compile for RDNA2 gfx1030 devices (GFX10 series).

> **gfx1031**
>> Compile for RDNA2 gfx1031 devices (GFX10 series). (Experimental)

> **gfx1032**
>> Compile for RDNA2 gfx1032 devices (GFX10 series). (Experimental)

> **gfx1033**
>> Compile for RDNA2 gfx1033 devices (GFX10 series). (Experimental)

> **gfx1034**
>> Compile for RDNA2 gfx1034 devices (GFX10 series). (Experimental)

> **gfx1035**
>> Compile for RDNA2 gfx1035 devices (GFX10 series). (Experimental)

> **gfx1036**
>> Compile for RDNA2 gfx1036 devices (GFX10 series).

> **gfx10−3−generic**
>> Compile generic code for GFX10−3 devices, executable on gfx1030, gfx1031, gfx1032, gfx1033, gfx1034, gfx1035, and gfx1036. (Experimental)

> **gfx1100**
>> Compile for RDNA3 gfx1100 devices (GFX11 series).

> **gfx1101**
>> Compile for RDNA3 gfx1101 devices (GFX11 series). (Experimental)

> **gfx1102**
>> Compile for RDNA3 gfx1102 devices (GFX11 series). (Experimental)

gfx1103

Compile for RDNA3 gfx1103 devices (GFX11 series).

gfx1150

Compile for RDNA3 gfx1150 devices (GFX11 series). (Experimental)

gfx1151

Compile for RDNA3 gfx1151 devices (GFX11 series). (Experimental)

gfx1152

Compile for RDNA3 gfx1152 devices (GFX11 series). (Experimental)

gfx1153

Compile for RDNA3 gfx1153 devices (GFX11 series). (Experimental)

gfx11−generic

Compile generic code for GFX11 devices, executable on gfx1100, gfx1101, gfx1102, gfx1103, gfx1150, gfx1151, gfx1152, and gfx1153. (Experimental)

−msram−ecc=on
−msram−ecc=off
−msram−ecc=any

Compile binaries suitable for devices with the SRAM−ECC feature enabled, disabled, or either mode. This feature can be enabled per−process on some devices. The compiled code must match the device mode. The default is **any**, for devices that support it.

−mstack−size=*bytes*

Specify how many *bytes* of stack space will be requested for each GPU thread (wave−front). Beware that there may be many threads and limited memory available. The size of the stack allocation may also have an impact on run−time performance. The default is 32KB when using OpenACC or OpenMP, and 1MB otherwise.

−mxnack=on
−mxnack=off
−mxnack=any

Compile binaries suitable for devices with the XNACK feature enabled, disabled, or either mode. Some devices always require XNACK and some allow the user to configure XNACK. The compiled code must match the device mode. The default is **−mxnack=any** on devices that support Unified Shared Memory, and **−mxnack=no** otherwise.

ARC Options

The following options control the architecture variant for which code is being compiled:

−mbarrel−shifter

Generate instructions supported by barrel shifter. This is the default unless **−mcpu=ARC601** or **−mcpu=ARCEM** is in effect.

−mjli−always

Force to call a function using jli_s instruction. This option is valid only for ARCv2 architecture.

−mcpu=*cpu*

Set architecture type, register usage, and instruction scheduling parameters for *cpu*. There are also shortcut alias options available for backward compatibility and convenience. Supported values for *cpu* are

arc600

Compile for ARC600. Aliases: **−mA6**, **−mARC600**.

arc601

Compile for ARC601. Alias: **−mARC601**.

arc700

   Compile for ARC700. Aliases: **−mA7**, **−mARC700**. This is the default when configured with **−−with−cpu=arc700**.

arcem

   Compile for ARC EM.

archs

   Compile for ARC HS.

em   Compile for ARC EM CPU with no hardware extensions.

em4

   Compile for ARC EM4 CPU.

em4_dmips

   Compile for ARC EM4 DMIPS CPU.

em4_fpus

   Compile for ARC EM4 DMIPS CPU with the single−precision floating−point extension.

em4_fpuda

   Compile for ARC EM4 DMIPS CPU with single−precision floating−point and double assist instructions.

hs   Compile for ARC HS CPU with no hardware extensions except the atomic instructions.

hs34

   Compile for ARC HS34 CPU.

hs38

   Compile for ARC HS38 CPU.

hs38_linux

   Compile for ARC HS38 CPU with all hardware extensions on.

hs4x

   Compile for ARC HS4x CPU.

hs4xd

   Compile for ARC HS4xD CPU.

hs4x_rel31

   Compile for ARC HS4x CPU release 3.10a.

arc600_norm

   Compile for ARC 600 CPU with `norm` instructions enabled.

arc600_mul32x16

   Compile for ARC 600 CPU with `norm` and 32x16−bit multiply instructions enabled.

arc600_mul64

   Compile for ARC 600 CPU with `norm` and `mul64`−family instructions enabled.

arc601_norm

   Compile for ARC 601 CPU with `norm` instructions enabled.

arc601_mul32x16

   Compile for ARC 601 CPU with `norm` and 32x16−bit multiply instructions enabled.

arc601_mul64

   Compile for ARC 601 CPU with `norm` and `mul64`−family instructions enabled.

nps400

   Compile for ARC 700 on NPS400 chip.

em_mini
> Compile for ARC EM minimalist configuration featuring reduced register set.

−mdpfp
−mdpfp−compact
> Generate double−precision FPX instructions, tuned for the compact implementation.

−mdpfp−fast
> Generate double−precision FPX instructions, tuned for the fast implementation.

−mno−dpfp−lrsr
> Disable `lr` and `sr` instructions from using FPX extension aux registers.

−mea
> Generate extended arithmetic instructions. Currently only `divaw`, `adds`, `subs`, and `sat16` are supported. Only valid for **−mcpu=ARC700**.

−mno−mpy
> Do not generate `mpy`−family instructions for ARC700. This option is deprecated.

−mmul32x16
> Generate 32x16−bit multiply and multiply−accumulate instructions.

−mmul64
> Generate `mul64` and `mulu64` instructions. Only valid for **−mcpu=ARC600**.

−mnorm
> Generate `norm` instructions. This is the default if **−mcpu=ARC700** is in effect.

−mspfp
−mspfp−compact
> Generate single−precision FPX instructions, tuned for the compact implementation.

−mspfp−fast
> Generate single−precision FPX instructions, tuned for the fast implementation.

−msimd
> Enable generation of ARC SIMD instructions via target−specific builtins. Only valid for **−mcpu=ARC700**.

−msoft−float
> This option ignored; it is provided for compatibility purposes only. Software floating−point code is emitted by default, and this default can overridden by FPX options; **−mspfp**, **−mspfp−compact**, or **−mspfp−fast** for single precision, and **−mdpfp**, **−mdpfp−compact**, or **−mdpfp−fast** for double precision.

−mswap
> Generate `swap` instructions.

−matomic
> This enables use of the locked load/store conditional extension to implement atomic memory built−in functions. Not available for ARC 6xx or ARC EM cores.

−mdiv−rem
> Enable `div` and `rem` instructions for ARCv2 cores.

−mcode−density
> Enable code density instructions for ARC EM. This option is on by default for ARC HS.

−mll64
> Enable double load/store operations for ARC HS cores.

−mtp−regno=*regno*
> Specify thread pointer register number.

−mmpy−option=*multo*

Compile ARCv2 code with a multiplier design option. You can specify the option using either a string or numeric value for *multo*. **wlh1** is the default value. The recognized values are:

0

none

No multiplier available.

1

w 16x16 multiplier, fully pipelined. The following instructions are enabled: `mpyw` and `mpyuw`.

2

wlh1

32x32 multiplier, fully pipelined (1 stage). The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.

3

wlh2

32x32 multiplier, fully pipelined (2 stages). The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.

4

wlh3

Two 16x16 multipliers, blocking, sequential. The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.

5

wlh4

One 16x16 multiplier, blocking, sequential. The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.

6

wlh5

One 32x4 multiplier, blocking, sequential. The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.

7

plus_dmpy

ARC HS SIMD support.

8

plus_macd

ARC HS SIMD support.

9

plus_qmacw

ARC HS SIMD support.

This option is only available for ARCv2 cores.

−mfpu=*fpu*

Enables support for specific floating−point hardware extensions for ARCv2 cores. Supported values for *fpu* are:

fpus

Enables support for single−precision floating−point hardware extensions.

fpud

Enables support for double−precision floating−point hardware extensions. The single−precision floating−point extension is also enabled. Not available for ARC EM.

fpuda

Enables support for double−precision floating−point hardware extensions using double−precision assist instructions. The single−precision floating−point extension is also enabled. This option is only available for ARC EM.

fpuda_div

Enables support for double−precision floating−point hardware extensions using double−precision assist instructions. The single−precision floating−point, square−root, and divide extensions are also enabled. This option is only available for ARC EM.

fpuda_fma

Enables support for double−precision floating−point hardware extensions using double−precision assist instructions. The single−precision floating−point and fused multiply and add hardware extensions are also enabled. This option is only available for ARC EM.

fpuda_all

Enables support for double−precision floating−point hardware extensions using double−precision assist instructions. All single−precision floating−point hardware extensions are also enabled. This option is only available for ARC EM.

fpus_div

Enables support for single−precision floating−point, square−root and divide hardware extensions.

fpud_div

Enables support for double−precision floating−point, square−root and divide hardware extensions. This option includes option **fpus_div**. Not available for ARC EM.

fpus_fma

Enables support for single−precision floating−point and fused multiply and add hardware extensions.

fpud_fma

Enables support for double−precision floating−point and fused multiply and add hardware extensions. This option includes option **fpus_fma**. Not available for ARC EM.

fpus_all

Enables support for all single−precision floating−point hardware extensions.

fpud_all

Enables support for all single− and double−precision floating−point hardware extensions. Not available for ARC EM.

−mirq−ctrl−saved=*register−range***,** *blink***,** *lp_count*

Specifies general−purposes registers that the processor automatically saves/restores on interrupt entry and exit. *register−range* is specified as two registers separated by a dash. The register range always starts with `r0`, the upper limit is `fp` register. *blink* and *lp_count* are optional. This option is only valid for ARC EM and ARC HS cores.

−mrgf−banked−regs=*number*

Specifies the number of registers replicated in second register bank on entry to fast interrupt. Fast interrupts are interrupts with the highest priority level P0. These interrupts save only PC and STATUS32 registers to avoid memory transactions during interrupt entry and exit sequences. Use this option when you are using fast interrupts in an ARC V2 family processor. Permitted values are 4, 8, 16, and 32.

−mlpc−width=*width*

Specify the width of the `lp_count` register. Valid values for *width* are 8, 16, 20, 24, 28 and 32 bits. The default width is fixed to 32 bits. If the width is less than 32, the compiler does not attempt to transform loops in your program to use the zero−delay loop mechanism unless it is known that the `lp_count` register can hold the required loop−counter value. Depending on the width specified, the compiler and run−time library might continue to use the loop mechanism for various needs. This option defines macro `__ARC_LPC_WIDTH__` with the value of *width*.

−mrf16

This option instructs the compiler to generate code for a 16−entry register file. This option defines the `__ARC_RF16__` preprocessor macro.

−mbranch−index

Enable use of `bi` or `bih` instructions to implement jump tables.

The following options are passed through to the assembler, and also define preprocessor macro symbols.

−mdsp−packa

Passed down to the assembler to enable the DSP Pack A extensions. Also sets the preprocessor symbol `__Xdsp_packa`. This option is deprecated.

−mdvbf

Passed down to the assembler to enable the dual Viterbi butterfly extension. Also sets the preprocessor symbol `__Xdvbf`. This option is deprecated.

−mlock

Passed down to the assembler to enable the locked load/store conditional extension. Also sets the preprocessor symbol `__Xlock`.

−mmac−d16

Passed down to the assembler. Also sets the preprocessor symbol `__Xxmac_d16`. This option is deprecated.

−mmac−24

Passed down to the assembler. Also sets the preprocessor symbol `__Xxmac_24`. This option is deprecated.

−mrtsc

Passed down to the assembler to enable the 64−bit time−stamp counter extension instruction. Also sets the preprocessor symbol `__Xrtsc`. This option is deprecated.

−mswape

Passed down to the assembler to enable the swap byte ordering extension instruction. Also sets the preprocessor symbol `__Xswape`.

−mtelephony

Passed down to the assembler to enable dual− and single−operand instructions for telephony. Also sets the preprocessor symbol `__Xtelephony`. This option is deprecated.

−mxy

Passed down to the assembler to enable the XY memory extension. Also sets the preprocessor symbol `__Xxy`.

The following options control how the assembly code is annotated:

−misize

Annotate assembler instructions with estimated addresses.

−mannotate−align

Does nothing. Preserved for backward compatibility.

The following options are passed through to the linker:

−marclinux

Passed through to the linker, to specify use of the `arclinux` emulation. This option is enabled by default in tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets when profiling is not requested.

−marclinux_prof

Passed through to the linker, to specify use of the `arclinux_prof` emulation. This option is enabled by default in tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets when profiling is requested.

The following options control the semantics of generated code:

−mlong−calls
  Generate calls as register indirect calls, thus providing access to the full 32−bit address range.

−mmedium−calls
  Don't use less than 25−bit addressing range for calls, which is the offset available for an unconditional branch−and−link instruction. Conditional execution of function calls is suppressed, to allow use of the 25−bit range, rather than the 21−bit range with conditional branch−and−link. This is the default for tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets.

−G *num*
  Put definitions of externally−visible data in a small data section if that data is no bigger than *num* bytes. The default value of *num* is 4 for any ARC configuration, or 8 when we have double load/store operations.

−mno−sdata
  Do not generate sdata references. This is the default for tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets.

−mvolatile−cache
  Use ordinarily cached memory accesses for volatile references. This is the default.

−mno−volatile−cache
  Enable cache bypass for volatile references.

The following options fine tune code generation:

−malign−call
  Does nothing. Preserved for backward compatibility.

−mauto−modify−reg
  Enable the use of pre/post modify with register displacement.

−mbbit−peephole
  Does nothing. Preserved for backward compatibility.

−mno−brcc
  This option disables a target−specific pass in *arc_reorg* to generate compare−and−branch (`brcc`) instructions. It has no effect on generation of these instructions driven by the combiner pass.

−mcase−vector−pcrel
  Use PC−relative switch case tables to enable case table shortening. This is the default for **−Os**.

−mcompact−casesi
  Enable compact `casesi` pattern. This is the default for **−Os**, and only available for ARCv1 cores. This option is deprecated.

−mno−cond−exec
  Disable the ARCompact−specific pass to generate conditional execution instructions.

  Due to delay slot scheduling and interactions between operand numbers, literal sizes, instruction lengths, and the support for conditional execution, the target−independent pass to generate conditional execution is often lacking, so the ARC port has kept a special pass around that tries to find more conditional execution generation opportunities after register allocation, branch shortening, and delay slot scheduling have been done. This pass generally, but not always, improves performance and code size, at the cost of extra compilation time, which is why there is an option to switch it off. If you have a problem with call instructions exceeding their allowable offset range because they are conditionalized, you should consider using **−mmedium−calls** instead.

−mearly−cbranchsi
  Enable pre−reload use of the `cbranchsi` pattern.

−mexpand−adddi

Expand `adddi3` and `subdi3` at RTL generation time into `add.f`, `adc` etc. This option is deprecated.

−mindexed−loads

Enable the use of indexed loads. This can be problematic because some optimizers then assume that indexed stores exist, which is not the case.

−mlra

Does nothing. Preserved for backward compatibility.

−mlra−priority−none

Don't indicate any priority for target registers.

−mlra−priority−compact

Indicate target register priority for r0..r3 / r12..r15.

−mlra−priority−noncompact

Reduce target register priority for r0..r3 / r12..r15.

−mmillicode

When optimizing for size (using **−Os**), prologues and epilogues that have to save or restore a large number of registers are often shortened by using call to a special function in libgcc; this is referred to as a *millicode* call. As these calls can pose performance issues, and/or cause linking issues when linking in a nonstandard way, this option is provided to turn on or off millicode call generation.

−mcode−density−frame

This option enable the compiler to emit `enter` and `leave` instructions. These instructions are only valid for CPUs with code−density feature.

−mmixed−code

Does nothing. Preserved for backward compatibility.

−mq−class

Ths option is deprecated. Enable **q** instruction alternatives. This is the default for **−Os**.

−mRcq

Does nothing. Preserved for backward compatibility.

−mRcw

Does nothing. Preserved for backward compatibility.

−msize−level=*level*

Fine−tune size optimization with regards to instruction lengths and alignment. The recognized values for *level* are:

0   No size optimization. This level is deprecated and treated like **1**.

1   Short instructions are used opportunistically.

2   In addition, alignment of loops and of code after barriers are dropped.

3   In addition, optional data alignment is dropped, and the option **Os** is enabled.

This defaults to **3** when **−Os** is in effect. Otherwise, the behavior when this is not set is equivalent to level **1**.

−mtune=*cpu*

Set instruction scheduling parameters for *cpu*, overriding any implied by **−mcpu=**.

Supported values for *cpu* are

ARC600

Tune for ARC600 CPU.

ARC601
>   Tune for ARC601 CPU.

ARC700
>   Tune for ARC700 CPU with standard multiplier block.

ARC700−xmac
>   Tune for ARC700 CPU with XMAC block.

ARC725D
>   Tune for ARC725D CPU.

ARC750D
>   Tune for ARC750D CPU.

core3
>   Tune for ARCv2 core3 type CPU. This option enable usage of `dbnz` instruction.

release31a
>   Tune for ARC4x release 3.10a.

−mmultcost=_num_
>   Cost to assume for a multiply instruction, with **4** being equal to a normal instruction.

−munalign−prob−threshold=_probability_
>   Does nothing. Preserved for backward compatibility.

The following options are maintained for backward compatibility, but are now deprecated and will be removed in a future release:

−margonaut
>   Obsolete FPX.

−mbig−endian
−EB
>   Compile code for big−endian targets. Use of these options is now deprecated. Big−endian code is supported by configuring GCC to build `arceb-elf32` and `arceb-linux-uclibc` targets, for which big endian is the default.

−mlittle−endian
−EL
>   Compile code for little−endian targets. Use of these options is now deprecated. Little−endian code is supported by configuring GCC to build `arc-elf32` and `arc-linux-uclibc` targets, for which little endian is the default.

−mbarrel_shifter
>   Replaced by **−mbarrel−shifter**.

−mdpfp_compact
>   Replaced by **−mdpfp−compact**.

−mdpfp_fast
>   Replaced by **−mdpfp−fast**.

−mdsp_packa
>   Replaced by **−mdsp−packa**.

−mEA
>   Replaced by **−mea**.

−mmac_24
>   Replaced by **−mmac−24**.

−mmac_d16
>   Replaced by **−mmac−d16**.

−mspfp_compact

  Replaced by **−mspfp−compact**.

−mspfp_fast

  Replaced by **−mspfp−fast**.

−mtune=*cpu*

  Values **arc600**, **arc601**, **arc700** and **arc700−xmac** for *cpu* are replaced by **ARC600**, **ARC601**, **ARC700** and **ARC700−xmac** respectively.

−multcost=*num*

  Replaced by **−mmultcost**.

ARM Options

  These **−m** options are defined for the ARM port:

−mabi=*name*

  Generate code for the specified ABI. Permissible values are: **apcs−gnu**, **atpcs**, **aapcs**, **aapcs−linux** and **iwmmxt**.

−mapcs−frame

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

−mapcs

  This is a synonym for **−mapcs−frame** and is deprecated.

−mthumb−interwork

  Generate code that supports calling between the ARM and Thumb instruction sets. Without this option, on pre−v5 architectures, the two instruction sets cannot be reliably used inside one program. The default is **−mno−thumb−interwork**, since slightly larger code is generated when **−mthumb−interwork** is specified. In AAPCS configurations this option is meaningless.

−mno−sched−prolog

  Prevent the reordering of instructions in the function prologue, or the merging of those instruction with the instructions in the function's body. This means that all functions start with a recognizable set of instructions (or in fact one of a choice from a small set of different function prologues), and this information can be used to locate the start of functions inside an executable piece of code. The default is **−msched−prolog**.

−mfloat−abi=*name*

  Specifies which floating−point ABI to use. Permissible values are: **soft**, **softfp** and **hard**.

  Specifying **soft** causes GCC to generate output containing library calls for floating−point operations. **softfp** allows the generation of code using hardware floating−point instructions, but still uses the soft−float calling conventions. **hard** allows generation of floating−point instructions and uses FPU−specific calling conventions.

  The default depends on the specific target configuration. Note that the hard−float and soft−float ABIs are not link−compatible; you must compile your entire program with the same ABI, and link with a compatible set of libraries.

−mgeneral−regs−only

  Generate code which uses only the general−purpose registers. This will prevent the compiler from using floating−point and Advanced SIMD registers but will not impose any restrictions on the assembler.

−mlittle−endian

  Generate code for a processor running in little−endian mode. This is the default for all standard configurations.

−mbig−endian

> Generate code for a processor running in big−endian mode; the default is to compile code for a little−endian processor.

−mbe8
−mbe32

> When linking a big−endian image select between BE8 and BE32 formats. The option has no effect for little−endian images and is ignored. The default is dependent on the selected target architecture. For ARMv6 and later architectures the default is BE8, for older architectures the default is BE32. BE32 format has been deprecated by ARM.

−march=_name_[**+extension...**]

> This specifies the name of the target ARM architecture. GCC uses this name to determine what kind of instructions it can emit when generating assembly code. This option can be used in conjunction with or instead of the **−mcpu=** option.
>
> Permissible names are: **armv4t**, **armv5t**, **armv5te**, **armv6**, **armv6j**, **armv6k**, **armv6kz**, **armv6t2**, **armv6z**, **armv6zk**, **armv7**, **armv7−a**, **armv7ve**, **armv8−a**, **armv8.1−a**, **armv8.2−a**, **armv8.3−a**, **armv8.4−a**, **armv8.5−a**, **armv8.6−a**, **armv9−a**, **armv7−r**, **armv8−r**, **armv6−m**, **armv6s−m**, **armv7−m**, **armv7e−m**, **armv8−m.base**, **armv8−m.main**, **armv8.1−m.main**, **iwmmxt** and **iwmmxt2**.
>
> Additionally, the following architectures, which lack support for the Thumb execution state, are recognized but support is deprecated: **armv4**.
>
> Many of the architectures support extensions. These can be added by appending **+**_extension_ to the architecture name. Extension options are processed in order and capabilities accumulate. An extension will also enable any necessary base extensions upon which it depends. For example, the **+crypto** extension will always enable the **+simd** extension. The exception to the additive construction is for extensions that are prefixed with **+no...**: these extensions disable the specified option and any other extensions that may depend on the presence of that extension.
>
> For example, **−march=armv7−a+simd+nofp+vfpv4** is equivalent to writing **−march=armv7−a+vfpv4** since the **+simd** option is entirely disabled by the **+nofp** option that follows it.
>
> Most extension names are generically named, but have an effect that is dependent upon the architecture to which it is applied. For example, the **+simd** option can be applied to both **armv7−a** and **armv8−a** architectures, but will enable the original ARMv7−A Advanced SIMD (Neon) extensions for **armv7−a** and the ARMv8−A variant for **armv8−a**.
>
> The table below lists the supported extensions for each architecture. Architectures not mentioned do not support any extensions.

> **armv5te**
> **armv6**
> **armv6j**
> **armv6k**
> **armv6kz**
> **armv6t2**
> **armv6z**
> **armv6zk**
>
> > **+fp** The VFPv2 floating−point instructions. The extension **+vfpv2** can be used as an alias for this extension.
> >
> > **+nofp**
> > > Disable the floating−point instructions.

armv7

The common subset of the ARMv7−A, ARMv7−R and ARMv7−M architectures.

+fp The VFPv3 floating−point instructions, with 16 double−precision registers. The extension **+vfpv3−d16** can be used as an alias for this extension. Note that floating−point is not supported by the base ARMv7−M architecture, but is compatible with both the ARMv7−A and ARMv7−R architectures.

+nofp

Disable the floating−point instructions.

armv7−a

+mp

The multiprocessing extension.

+sec

The security extension.

+fp The VFPv3 floating−point instructions, with 16 double−precision registers. The extension **+vfpv3−d16** can be used as an alias for this extension.

+simd

The Advanced SIMD (Neon) v1 and the VFPv3 floating−point instructions. The extensions **+neon** and **+neon−vfpv3** can be used as aliases for this extension.

+vfpv3

The VFPv3 floating−point instructions, with 32 double−precision registers.

+vfpv3−d16−fp16

The VFPv3 floating−point instructions, with 16 double−precision registers and the half−precision floating−point conversion operations.

+vfpv3−fp16

The VFPv3 floating−point instructions, with 32 double−precision registers and the half−precision floating−point conversion operations.

+vfpv4−d16

The VFPv4 floating−point instructions, with 16 double−precision registers.

+vfpv4

The VFPv4 floating−point instructions, with 32 double−precision registers.

+neon−fp16

The Advanced SIMD (Neon) v1 and the VFPv3 floating−point instructions, with the half−precision floating−point conversion operations.

+neon−vfpv4

The Advanced SIMD (Neon) v2 and the VFPv4 floating−point instructions.

+nosimd

Disable the Advanced SIMD instructions (does not disable floating point).

+nofp

Disable the floating−point and Advanced SIMD instructions.

armv7ve

The extended version of the ARMv7−A architecture with support for virtualization.

+fp The VFPv4 floating−point instructions, with 16 double−precision registers. The extension **+vfpv4−d16** can be used as an alias for this extension.

+simd

The Advanced SIMD (Neon) v2 and the VFPv4 floating−point instructions. The extension **+neon−vfpv4** can be used as an alias for this extension.

+vfpv3−d16
> The VFPv3 floating−point instructions, with 16 double−precision registers.

+vfpv3
> The VFPv3 floating−point instructions, with 32 double−precision registers.

+vfpv3−d16−fp16
> The VFPv3 floating−point instructions, with 16 double−precision registers and the half−precision floating−point conversion operations.

+vfpv3−fp16
> The VFPv3 floating−point instructions, with 32 double−precision registers and the half−precision floating−point conversion operations.

+vfpv4−d16
> The VFPv4 floating−point instructions, with 16 double−precision registers.

+vfpv4
> The VFPv4 floating−point instructions, with 32 double−precision registers.

+neon
> The Advanced SIMD (Neon) v1 and the VFPv3 floating−point instructions. The extension **+neon−vfpv3** can be used as an alias for this extension.

+neon−fp16
> The Advanced SIMD (Neon) v1 and the VFPv3 floating−point instructions, with the half−precision floating−point conversion operations.

+nosimd
> Disable the Advanced SIMD instructions (does not disable floating point).

+nofp
> Disable the floating−point and Advanced SIMD instructions.

armv8−a
    +crc
> The Cyclic Redundancy Check (CRC) instructions.

    +simd
> The ARMv8−A Advanced SIMD and floating−point instructions.

    +crypto
> The cryptographic instructions.

    +nocrypto
> Disable the cryptographic instructions.

    +nofp
> Disable the floating−point, Advanced SIMD and cryptographic instructions.

    +sb
> Speculation Barrier Instruction.

    +predres
> Execution and Data Prediction Restriction Instructions.

armv8.1−a
    +simd
> The ARMv8.1−A Advanced SIMD and floating−point instructions.

    +crypto
> The cryptographic instructions. This also enables the Advanced SIMD and floating−point instructions.

+nocrypto

Disable the cryptographic instructions.

+nofp

Disable the floating−point, Advanced SIMD and cryptographic instructions.

+sb

Speculation Barrier Instruction.

+predres

Execution and Data Prediction Restriction Instructions.

armv8.2−a
armv8.3−a

+fp16

The half−precision floating−point data processing instructions. This also enables the Advanced SIMD and floating−point instructions.

+fp16fml

The half−precision floating−point fmla extension. This also enables the half−precision floating−point extension and Advanced SIMD and floating−point instructions.

+simd

The ARMv8.1−A Advanced SIMD and floating−point instructions.

+crypto

The cryptographic instructions. This also enables the Advanced SIMD and floating−point instructions.

+dotprod

Enable the Dot Product extension. This also enables Advanced SIMD instructions.

+nocrypto

Disable the cryptographic extension.

+nofp

Disable the floating−point, Advanced SIMD and cryptographic instructions.

+sb

Speculation Barrier Instruction.

+predres

Execution and Data Prediction Restriction Instructions.

+i8mm

8−bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating−point instructions.

+bf16

Brain half−precision floating−point instructions. This also enables Advanced SIMD and floating−point instructions.

armv8.4−a

+fp16

The half−precision floating−point data processing instructions. This also enables the Advanced SIMD and floating−point instructions as well as the Dot Product extension and the half−precision floating−point fmla extension.

+simd

The ARMv8.3−A Advanced SIMD and floating−point instructions as well as the Dot Product extension.

+crypto

The cryptographic instructions. This also enables the Advanced SIMD and floating−point instructions as well as the Dot Product extension.

+nocrypto

Disable the cryptographic extension.

+nofp

Disable the floating−point, Advanced SIMD and cryptographic instructions.

+sb

Speculation Barrier Instruction.

+predres

Execution and Data Prediction Restriction Instructions.

+i8mm

8−bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating−point instructions.

+bf16

Brain half−precision floating−point instructions. This also enables Advanced SIMD and floating−point instructions.

armv8.5−a
    +fp16

The half−precision floating−point data processing instructions. This also enables the Advanced SIMD and floating−point instructions as well as the Dot Product extension and the half−precision floating−point fmla extension.

    +simd

The ARMv8.3−A Advanced SIMD and floating−point instructions as well as the Dot Product extension.

    +crypto

The cryptographic instructions. This also enables the Advanced SIMD and floating−point instructions as well as the Dot Product extension.

    +nocrypto

Disable the cryptographic extension.

    +nofp

Disable the floating−point, Advanced SIMD and cryptographic instructions.

    +i8mm

8−bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating−point instructions.

    +bf16

Brain half−precision floating−point instructions. This also enables Advanced SIMD and floating−point instructions.

armv8.6−a
    +fp16

The half−precision floating−point data processing instructions. This also enables the Advanced SIMD and floating−point instructions as well as the Dot Product extension and the half−precision floating−point fmla extension.

    +simd

The ARMv8.3−A Advanced SIMD and floating−point instructions as well as the Dot Product extension.

    +crypto

The cryptographic instructions. This also enables the Advanced SIMD and floating−point instructions as well as the Dot Product extension.

+nocrypto

Disable the cryptographic extension.

+nofp

Disable the floating−point, Advanced SIMD and cryptographic instructions.

+i8mm

8−bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating−point instructions.

+bf16

Brain half−precision floating−point instructions. This also enables Advanced SIMD and floating−point instructions.

armv7−r

+fp.sp

The single−precision VFPv3 floating−point instructions. The extension **+vfpv3xd** can be used as an alias for this extension.

+fp The VFPv3 floating−point instructions with 16 double−precision registers. The extension +vfpv3−d16 can be used as an alias for this extension.

+vfpv3xd−d16−fp16

The single−precision VFPv3 floating−point instructions with 16 double−precision registers and the half−precision floating−point conversion operations.

+vfpv3−d16−fp16

The VFPv3 floating−point instructions with 16 double−precision registers and the half−precision floating−point conversion operations.

+nofp

Disable the floating−point extension.

+idiv

The ARM−state integer division instructions.

+noidiv

Disable the ARM−state integer division extension.

armv7e−m

+fp The single-precision VFPv4 floating−point instructions.

+fpv5

The single−precision FPv5 floating−point instructions.

+fp.dp

The single− and double−precision FPv5 floating−point instructions.

+nofp

Disable the floating−point extensions.

armv8.1−m.main

+dsp

The DSP instructions.

+mve

The M−Profile Vector Extension (MVE) integer instructions.

+mve.fp

The M−Profile Vector Extension (MVE) integer and single precision floating−point instructions.

+fp The single−precision floating−point instructions.

+fp.dp
> The single− and double−precision floating−point instructions.

+nofp
> Disable the floating−point extension.

+cdecp0, +cdecp1, ... , +cdecp7
> Enable the Custom Datapath Extension (CDE) on selected coprocessors according to the numbers given in the options in the range 0 to 7.

+pacbti
> Enable the Pointer Authentication and Branch Target Identification Extension.

armv8−m.main
> **+dsp**
> > The DSP instructions.
>
> **+nodsp**
> > Disable the DSP extension.
>
> **+fp** The single−precision floating−point instructions.
>
> **+fp.dp**
> > The single− and double−precision floating−point instructions.
>
> **+nofp**
> > Disable the floating−point extension.
>
> **+cdecp0, +cdecp1, ... , +cdecp7**
> > Enable the Custom Datapath Extension (CDE) on selected coprocessors according to the numbers given in the options in the range 0 to 7.

armv8−r
> **+crc**
> > The Cyclic Redundancy Check (CRC) instructions.
>
> **+fp.sp**
> > The single−precision FPv5 floating−point instructions.
>
> **+simd**
> > The ARMv8−A Advanced SIMD and floating−point instructions.
>
> **+crypto**
> > The cryptographic instructions.
>
> **+nocrypto**
> > Disable the cryptographic instructions.
>
> **+nofp**
> > Disable the floating−point, Advanced SIMD and cryptographic instructions.

−march=native causes the compiler to auto−detect the architecture of the build computer. At present, this feature is only supported on GNU/Linux, and not all architectures are recognized. If the auto−detect is unsuccessful the option has no effect.

−march=unset causes the compiler to ignore any **−march=...** options that appear earlier on the command line and behave as if the option was never passed. This is useful to avoid warnings about conflicting CPU and architecture options when the two produce different architecture specifications.

−mtune=*name*
> This option specifies the name of the target ARM processor for which GCC should tune the performance of the code. For some ARM implementations better performance can be obtained by using this option. Permissible names are: **arm7tdmi**, **arm7tdmi−s**, **arm710t**, **arm720t**, **arm740t**, **strongarm**, **strongarm110**, **strongarm1100**, **strongarm1110**, **arm8**, **arm810**, **arm9**, **arm9e**, **arm920**, **arm920t**, **arm922t**, **arm946e−s**, **arm966e−s**, **arm968e−s**, **arm926ej−s**, **arm940t**,

arm9tdmi, **arm10tdmi**, **arm1020t**, **arm1026ej−s**, **arm10e**, **arm1020e**, **arm1022e**, **arm1136j−s**, **arm1136jf−s**, **mpcore**, **mpcorenovfp**, **arm1156t2−s**, **arm1156t2f−s**, **arm1176jz−s**, **arm1176jzf−s**, **generic−armv7−a**, **cortex−a5**, **cortex−a7**, **cortex−a8**, **cortex−a9**, **cortex−a12**, **cortex−a15**, **cortex−a17**, **cortex−a32**, **cortex−a35**, **cortex−a53**, **cortex−a55**, **cortex−a57**, **cortex−a72**, **cortex−a73**, **cortex−a75**, **cortex−a76**, **cortex−a76ae**, **cortex−a77**, **cortex−a78**, **cortex−a78ae**, **cortex−a78c**, **cortex−a710**, **ares**, **cortex−r4**, **cortex−r4f**, **cortex−r5**, **cortex−r7**, **cortex−r8**, **cortex−r52**, **cortex−r52plus**, **cortex−m0**, **cortex−m0plus**, **cortex−m1**, **cortex−m3**, **cortex−m4**, **cortex−m7**, **cortex−m23**, **cortex−m33**, **cortex−m35p**, **cortex−m52**, **cortex−m55**, **cortex−m85**, **cortex−x1**, **cortex−x1c**, **cortex−m1.small−multiply**, **cortex−m0.small−multiply**, **cortex−m0plus.small−multiply**, **exynos−m1**, **marvell−pj4**, **neoverse−n1**, **neoverse−n2**, **neoverse−v1**, **xscale**, **iwmmxt**, **iwmmxt2**, **ep9312**, **fa526**, **fa626**, **fa606te**, **fa626te**, **fmp626**, **fa726te**, **star−mc1**, **xgene1**.

Additionally, this option can specify that GCC should tune the performance of the code for a big.LITTLE system. Permissible names are: **cortex−a15.cortex−a7**, **cortex−a17.cortex−a7**, **cortex−a57.cortex−a53**, **cortex−a72.cortex−a53**, **cortex−a72.cortex−a35**, **cortex−a73.cortex−a53**, **cortex−a75.cortex−a55**, **cortex−a76.cortex−a55**.

−mtune=generic−*arch* specifies that GCC should tune the performance for a blend of processors within architecture *arch*. The aim is to generate code that run well on the current most popular processors, balancing between optimizations that benefit some CPUs in the range, and avoiding performance pitfalls of other CPUs. The effects of this option may change in future GCC versions as CPU models come and go.

−mtune permits the same extension options as **−mcpu**, but the extension options do not affect the tuning of the generated code.

−mtune=native causes the compiler to auto−detect the CPU of the build computer. At present, this feature is only supported on GNU/Linux, and not all architectures are recognized. If the auto−detect is unsuccessful the option has no effect.

−mcpu=*name*[**+extension...**]

This specifies the name of the target ARM processor. GCC uses this name to derive the name of the target ARM architecture (as if specified by **−march**) and the ARM processor type for which to tune for performance (as if specified by **−mtune**). Where this option is used in conjunction with **−march** or **−mtune**, those options take precedence over the appropriate part of this option.

Many of the supported CPUs implement optional architectural extensions. Where this is so the architectural extensions are normally enabled by default. If implementations that lack the extension exist, then the extension syntax can be used to disable those extensions that have been omitted. For floating−point and Advanced SIMD (Neon) instructions, the settings of the options **−mfloat−abi** and **−mfpu** must also be considered: floating−point and Advanced SIMD instructions will only be used if **−mfloat−abi** is not set to **soft**; and any setting of **−mfpu** other than **auto** will override the available floating−point and SIMD extension instructions.

For example, **cortex−a9** can be found in three major configurations: integer only, with just a floating−point unit or with floating−point and Advanced SIMD. The default is to enable all the instructions, but the extensions **+nosimd** and **+nofp** can be used to disable just the SIMD or both the SIMD and floating−point instructions respectively.

Permissible names for this option are the same as those for **−mtune**.

The following extension options are common to the listed CPUs:

+nodsp

Disable the DSP instructions on **cortex−m33**, **cortex−m35p**, **cortex−m52**, **cortex−m55** and **cortex−m85**. Also disable the M−Profile Vector Extension (MVE) integer and single precision floating−point instructions on **cortex−m52**, **cortex−m55** and **cortex−m85**.

+nopacbti

　　Disable the Pointer Authentication and Branch Target Identification Extension on **cortex−m52** and **cortex−m85**.

+nomve

　　Disable the M−Profile Vector Extension (MVE) integer and single precision floating−point instructions on **cortex−m52**, **cortex−m55** and **cortex−m85**.

+nomve.fp

　　Disable the M−Profile Vector Extension (MVE) single precision floating−point instructions on **cortex−m52**, **cortex−m55** and **cortex−m85**.

+cdecp0, +cdecp1, ... , +cdecp7

　　Enable the Custom Datapath Extension (CDE) on selected coprocessors according to the numbers given in the options in the range 0 to 7 on **cortex−m52**, **cortex−m55** and **star−mc1**.

+nofp

　　Disables the floating−point instructions on **arm9e**, **arm946e−s**, **arm966e−s**, **arm968e−s**, **arm10e**, **arm1020e**, **arm1022e**, **arm926ej−s**, **arm1026ej−s**, **cortex−r5**, **cortex−r7**, **cortex−r8**, **cortex−m4**, **cortex−m7**, **cortex−m33**, **cortex−m35p**, **cortex−m52**, **cortex−m55** and **cortex−m85**. Disables the floating−point and SIMD instructions on **generic−armv7−a**, **cortex−a5**, **cortex−a7**, **cortex−a8**, **cortex−a9**, **cortex−a12**, **cortex−a15**, **cortex−a17**, **cortex−a15.cortex−a7**, **cortex−a17.cortex−a7**, **cortex−a32**, **cortex−a35**, **cortex−a53** and **cortex−a55**.

+nofp.dp

　　Disables the double−precision component of the floating−point instructions on **cortex−r5**, **cortex−r7**, **cortex−r8**, **cortex−r52**, **cortex−r52plus** and **cortex−m7**.

+nosimd

　　Disables the SIMD (but not floating−point) instructions on **generic−armv7−a**, **cortex−a5**, **cortex−a7** and **cortex−a9**.

+crypto

　　Enables the cryptographic instructions on **cortex−a32**, **cortex−a35**, **cortex−a53**, **cortex−a55**, **cortex−a57**, **cortex−a72**, **cortex−a73**, **cortex−a75**, **exynos−m1**, **xgene1**, **cortex−a57.cortex−a53**, **cortex−a72.cortex−a53**, **cortex−a73.cortex−a35**, **cortex−a73.cortex−a53** and **cortex−a75.cortex−a55**.

Additionally the **generic−armv7−a** pseudo target defaults to VFPv3 with 16 double−precision registers. It supports the following extension options: **mp**, **sec**, **vfpv3−d16**, **vfpv3**, **vfpv3−d16−fp16**, **vfpv3−fp16**, **vfpv4−d16**, **vfpv4**, **neon**, **neon−vfpv3**, **neon−fp16**, **neon−vfpv4**. The meanings are the same as for the extensions to **−march=armv7−a**.

−mcpu=generic−*arch* is also permissible, and is equivalent to **−march=***arch* **−mtune=generic−***arch*. See **−mtune** for more information.

−mcpu=native causes the compiler to auto−detect the CPU of the build computer. At present, this feature is only supported on GNU/Linux, and not all architectures are recognized. If the auto−detect is unsuccessful the option has no effect.

−mcpu=unset causes the compiler to ignore any **−mcpu=...** options that appear earlier on the command line and behave as if the option was never passed. This is useful to avoid warnings about conflicting CPU and architecture options when the two produce different architecture specifications.

−mfpu=*name*

　　This specifies what floating−point hardware (or hardware emulation) is available on the target. Permissible names are: **auto**, **vfpv2**, **vfpv3**, **vfpv3−fp16**, **vfpv3−d16**, **vfpv3−d16−fp16**, **vfpv3xd**, **vfpv3xd−fp16**, **neon−vfpv3**, **neon−fp16**, **vfpv4**, **vfpv4−d16**, **fpv4−sp−d16**, **neon−vfpv4**, **fpv5−d16**, **fpv5−sp−d16**, **fp−armv8**, **neon−fp−armv8** and **crypto−neon−fp−armv8**. Note that **neon** is an alias for **neon−vfpv3** and **vfp** is an alias for **vfpv2**.

The setting **auto** is the default and is special. It causes the compiler to select the floating−point and Advanced SIMD instructions based on the settings of **−mcpu** and **−march**.

If the selected floating−point hardware includes the NEON extension (e.g. **−mfpu=neon**), note that floating−point operations are not generated by GCC's auto−vectorization pass unless **−funsafe−math−optimizations** is also specified. This is because NEON hardware does not fully implement the IEEE 754 standard for floating−point arithmetic (in particular denormal values are treated as zero), so the use of NEON instructions may lead to a loss of precision.

You can also set the fpu name at function level by using the `target("fpu=")` function attributes or pragmas.

−mfp16−format=_name_
> Specify the format of the `__fp16` half−precision floating−point type. Permissible names are **none**, **ieee**, and **alternative**; the default is **none**, in which case the `__fp16` type is not defined.

−mstructure−size−boundary=_n_
> The sizes of all structures and unions are rounded up to a multiple of the number of bits set by this option. Permissible values are 8, 32 and 64. The default value varies for different toolchains. For the COFF targeted toolchain the default value is 8. A value of 64 is only allowed if the underlying ABI supports it.
>
> Specifying a larger number can produce faster, more efficient code, but can also increase the size of the program. Different values are potentially incompatible. Code compiled with one value cannot necessarily expect to work with code or libraries compiled with another value, if they exchange information using structures or unions.
>
> This option is deprecated.

−mabort−on−noreturn
> Generate a call to the function `abort` at the end of a `noreturn` function. It is executed if the function tries to return.

−mlong−calls
−mno−long−calls
> Tells the compiler to perform function calls by first loading the address of the function into a register and then performing a subroutine call on this register. This switch is needed if the target function lies outside of the 64−megabyte addressing range of the offset−based version of subroutine call instruction.
>
> Even if this switch is enabled, not all function calls are turned into long calls. The heuristic is that static functions, functions that have the `short_call` attribute, functions that are inside the scope of a `#pragma no_long_calls` directive, and functions whose definitions have already been compiled within the current compilation unit are not turned into long calls. The exceptions to this rule are that weak function definitions, functions with the `long_call` attribute or the `section` attribute, and functions that are within the scope of a `#pragma long_calls` directive are always turned into long calls.
>
> This feature is not enabled by default. Specifying **−mno−long−calls** restores the default behavior, as does placing the function calls within the scope of a `#pragma long_calls_off` directive. Note these switches have no effect on how the compiler generates code to handle function calls via function pointers.

−msingle−pic−base
> Treat the register used for PIC addressing as read−only, rather than loading it in the prologue for each function. The runtime system is responsible for initializing this register with an appropriate value before execution begins.

−mpic−register=_reg_
> Specify the register to be used for PIC addressing. For standard PIC base case, the default is any suitable register determined by compiler. For single PIC base case, the default is **R9** if target is EABI

based or stack−checking is enabled, otherwise the default is **R10**.

−mpic−data−is−text−relative

Assume that the displacement between the text and data segments is fixed at static link time. This permits using PC−relative addressing operations to access data known to be in the data segment. For non−VxWorks RTP targets, this option is enabled by default. When disabled on such targets, it will enable **−msingle−pic−base** by default.

−mpoke−function−name

Write the name of each function into the text section, directly preceding the function prologue. The generated code is similar to this:

```
t0
    .ascii "arm_poke_function_name", 0
    .align
t1
    .word 0xff000000 + (t1 − t0)
arm_poke_function_name
    mov     ip, sp
    stmfd   sp!, {fp, ip, lr, pc}
    sub     fp, ip, #4
```

When performing a stack backtrace, code can inspect the value of `pc` stored at `fp + 0`. If the trace function then looks at location `pc − 12` and the top 8 bits are set, then we know that there is a function name embedded immediately preceding this location and has length (`(pc[-3]) & 0xff000000`).

−mthumb
−marm

Select between generating code that executes in ARM and Thumb states. The default for most configurations is to generate code that executes in ARM state, but the default can be changed by configuring GCC with the **−−with−mode=**_state_ configure option.

You can also override the ARM and Thumb mode for each function by using the `target("thumb")` and `target("arm")` function attributes or pragmas.

−mflip−thumb

Switch ARM/Thumb modes on alternating functions. This option is provided for regression testing of mixed Thumb/ARM code generation, and is not intended for ordinary use in compiling code.

−mtpcs−frame

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all non−leaf functions. (A leaf function is one that does not call any other functions.) The default is **−mno−tpcs−frame**.

−mtpcs−leaf−frame

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all leaf functions. (A leaf function is one that does not call any other functions.) The default is **−mno−apcs−leaf−frame**.

−mcallee−super−interworking

Gives all externally visible functions in the file being compiled an ARM instruction set header which switches to Thumb mode before executing the rest of the function. This allows these functions to be called from non−interworking code. This option is not valid in AAPCS configurations because interworking is enabled by default.

−mcaller−super−interworking

Allows calls via function pointers (including virtual functions) to execute correctly regardless of whether the target code has been compiled for interworking or not. There is a small overhead in the cost of executing a function pointer if this option is enabled. This option is not valid in AAPCS configurations because interworking is enabled by default.

−mtp=*name*

Specify the access model for the thread local storage pointer. The model **soft** generates calls to `__aeabi_read_tp`. Other accepted models are **tpidrurw**, **tpidruro** and **tpidrprw** which fetch the thread pointer from the corresponding system register directly (supported from the arm6k architecture and later). These system registers are accessed through the CP15 co−processor interface and the argument **cp15** is also accepted as a convenience alias of **tpidruro**. The argument **auto** uses the best available method for the selected processor. The default setting is **auto**.

−mtls−dialect=*dialect*

Specify the dialect to use for accessing thread local storage. Two *dialect*s are supported−−−**gnu** and **gnu2**. The **gnu** dialect selects the original GNU scheme for supporting local and global dynamic TLS models. The **gnu2** dialect selects the GNU descriptor scheme, which provides better performance for shared libraries. The GNU descriptor scheme is compatible with the original scheme, but does require new assembler, linker and library support. Initial and local exec TLS models are unaffected by this option and always use the original scheme.

−mword−relocations

Only generate absolute relocations on word−sized values (i.e. R_ARM_ABS32). This is enabled by default on targets (uClinux, SymbianOS) where the runtime loader imposes this restriction, and when **−fpic** or **−fPIC** is specified. This option conflicts with **−mslow−flash−data**.

−mfix−cortex−m3−ldrd

Some Cortex−M3 cores can cause data corruption when `ldrd` instructions with overlapping destination and base registers are used. This option avoids generating these instructions. This option is enabled by default when **−mcpu=cortex−m3** is specified.

−mfix−cortex−a57−aes−1742098
−mno−fix−cortex−a57−aes−1742098
−mfix−cortex−a72−aes−1655431
−mno−fix−cortex−a72−aes−1655431

Enable (disable) mitigation for an erratum on Cortex−A57 and Cortex−A72 that affects the AES cryptographic instructions. This option is enabled by default when either **−mcpu=cortex−a57** or **−mcpu=cortex−a72** is specified.

−munaligned−access
−mno−unaligned−access

Enables (or disables) reading and writing of 16− and 32− bit values from addresses that are not 16− or 32− bit aligned. By default unaligned access is disabled for all pre−ARMv6, all ARMv6−M and for ARMv8−M Baseline architectures, and enabled for all other architectures. If unaligned access is not enabled then words in packed data structures are accessed a byte at a time.

The ARM attribute `Tag_CPU_unaligned_access` is set in the generated object file to either true or false, depending upon the setting of this option. If unaligned access is enabled then the preprocessor symbol `__ARM_FEATURE_UNALIGNED` is also defined.

−mneon−for−64bits

This option is deprecated and has no effect.

−mslow−flash−data

Assume loading data from flash is slower than fetching instruction. Therefore literal load is minimized for better performance. This option is only supported when compiling for ARMv7 M−profile and off by default. It conflicts with **−mword−relocations**.

−masm−syntax−unified

Assume inline assembler is using unified asm syntax. The default is currently off which implies divided syntax. This option has no impact on Thumb2. However, this may change in future releases of GCC. Divided syntax should be considered deprecated.

−mrestrict−it

Restricts generation of IT blocks to conform to the rules of ARMv8−A. IT blocks can only contain a single 16−bit instruction from a select set of instructions. This option is on by default for ARMv8−A Thumb mode.

−mprint−tune−info

Print CPU tuning information as comment in assembler file. This is an option used only for regression testing of the compiler and not intended for ordinary use in compiling code. This option is disabled by default.

−mverbose−cost−dump

Enable verbose cost model dumping in the debug dump files. This option is provided for use in debugging the compiler.

−mpure−code

Do not allow constant data to be placed in code sections. Additionally, when compiling for ELF object format give all text sections the ELF processor−specific section attribute `SHF_ARM_PURECODE`. This option is only available when generating non−pic code for M−profile targets.

−mcmse

Generate secure code as per the "ARMv8−M Security Extensions: Requirements on Development Tools Engineering Specification", which can be found on **<https://developer.arm.com/documentation/ecm0359818/latest/>**.

−mfix−cmse−cve−2021−35465

Mitigate against a potential security issue with the `VLLDM` instruction in some M−profile devices when using CMSE (CVE−2021−365465). This option is enabled by default when the option **−mcpu=** is used with `cortex-m33`, `cortex-m35p`, `cortex-m52`, `cortex-m55`, `cortex-m85` or `star-mc1`. The option **−mno−fix−cmse−cve−2021−35465** can be used to disable the mitigation.

−mstack−protector−guard=*guard*
−mstack−protector−guard−offset=*offset*

Generate stack protection code using canary at *guard*. Supported locations are **global** for a global canary or **tls** for a canary accessible via the TLS register. The option **−mstack−protector−guard−offset=** is for use with **−fstack−protector−guard=tls** and not for use in user−land code.

−mfdpic
−mno−fdpic

Select the FDPIC ABI, which uses 64−bit function descriptors to represent pointers to functions. When the compiler is configured for `arm-*-uclinuxfdpiceabi` targets, this option is on by default and implies **−fPIE** if none of the PIC/PIE−related options is provided. On other targets, it only enables the FDPIC−specific code generation features, and the user should explicitly provide the PIC/PIE−related options as needed.

Note that static linking is not supported because it would still involve the dynamic linker when the program self−relocates. If such behavior is acceptable, use −static and −Wl,−dynamic−linker options.

The opposite **−mno−fdpic** option is useful (and required) to build the Linux kernel using the same (`arm-*-uclinuxfdpiceabi`) toolchain as the one used to build the userland programs.

−mbranch−protection=*none|standard|pac−ret***[+***leaf***][+***bti***]|*bti***[+***pac−ret***[+***leaf***]]**

Enable branch protection features (armv8.1−m.main only). **none** generate code without branch protection or return address signing. **standard[+***leaf***]** generate code with all branch protection features enabled at their standard level. **pac−ret[+***leaf***]** generate code with return address signing set to its standard level, which is to sign all functions that save the return address to memory. **leaf** When return address signing is enabled, also sign leaf functions even if they do not write the return address to memory. **+bti** Add landing−pad instructions at the permitted targets of indirect branch instructions.

If the **+pacbti** architecture extension is not enabled, then all branch protection and return address

signing operations are constrained to use only the instructions defined in the architectural−NOP space. The generated code will remain backwards−compatible with earlier versions of the architecture, but the additional security can be enabled at run time on processors that support the **PACBTI** extension.

Branch target enforcement using BTI can only be enabled at runtime if all code in the application has been compiled with at least **−mbranch−protection=bti**.

Any setting other than **none** is supported only on armv8−m.main or later.

The default is to generate code without branch protection or return address signing.

AVR Options

These options are defined for AVR implementations:

−mmcu=*mcu*

Specify the AVR instruction set architecture (ISA) or device type. The default for this option is `avr2`.

The following AVR devices and ISAs are supported. *Note:* A complete device support consists of startup code `crt`*mcu*`.o`, a device header `avr/io*.h`, a device library `lib`*mcu*`.a` and a device−specs (`https://gcc.gnu.org/wiki/avr-gcc#spec-files`) file `specs-`*mcu*. Only the latter is provided by the compiler according the supported *mcu*s below. The rest is supported by AVR−LibC (`https://github.com/avrdudes/avr-libc/`), or by means of `atpack` (`https://gcc.gnu.org/wiki/avr-gcc#atpack`) files from the hardware manufacturer.

`avr2`

"Classic" devices with up to 8 KiB of program memory. *mcu* = `attiny22`, `attiny26`, `at90s2313`, `at90s2323`, `at90s2333`, `at90s2343`, `at90s4414`, `at90s4433`, `at90s4434`, `at90c8534`, `at90s8515`, `at90s8535`.

`avr25`

"Classic" devices with up to 8 KiB of program memory and with the `MOVW` instruction. *mcu* = `attiny13`, `attiny13a`, `attiny24`, `attiny24a`, `attiny25`, `attiny261`, `attiny261a`, `attiny2313`, `attiny2313a`, `attiny43u`, `attiny44`, `attiny44a`, `attiny45`, `attiny48`, `attiny441`, `attiny461`, `attiny461a`, `attiny4313`, `attiny84`, `attiny84a`, `attiny85`, `attiny87`, `attiny88`, `attiny828`, `attiny841`, `attiny861`, `attiny861a`, `ata5272`, `ata6616c`, `at86rf401`.

`avr3`

"Classic" devices with 16 KiB up to 64 KiB of program memory. *mcu* = `at76c711`, `at43usb355`.

`avr31`

"Classic" devices with 128 KiB of program memory. *mcu* = `atmega103`, `at43usb320`.

`avr35`

"Classic" devices with 16 KiB up to 64 KiB of program memory and with the `MOVW` instruction. *mcu* = `attiny167`, `attiny1634`, `atmega8u2`, `atmega16u2`, `atmega32u2`, `ata5505`, `ata6617c`, `ata664251`, `at90usb82`, `at90usb162`.

`avr4`

"Enhanced" devices with up to 8 KiB of program memory. *mcu* = `atmega48`, `atmega48a`, `atmega48p`, `atmega48pa`, `atmega48pb`, `atmega8`, `atmega8a`, `atmega8hva`, `atmega88`, `atmega88a`, `atmega88p`, `atmega88pa`, `atmega88pb`, `atmega8515`, `atmega8535`, `ata5795`, `ata6285`, `ata6286`, `ata6289`, `ata6612c`, `at90pwm1`, `at90pwm2`, `at90pwm2b`, `at90pwm3`, `at90pwm3b`, `at90pwm81`.

`avr5`

"Enhanced" devices with 16 KiB up to 64 KiB of program memory. *mcu* = `atmega16`, `atmega16a`, `atmega16hva`, `atmega16hva2`, `atmega16hvb`, `atmega16hvbrevb`, `atmega16m1`, `atmega16u4`, `atmega161`, `atmega162`, `atmega163`, `atmega164a`, `atmega164p`, `atmega164pa`, `atmega165`, `atmega165a`, `atmega165p`, `atmega165pa`, `atmega168`, `atmega168a`, `atmega168p`, `atmega168pa`,

atmega168pb, atmega169, atmega169a, atmega169p, atmega169pa, atmega32,
atmega32a, atmega32c1, atmega32hvb, atmega32hvbrevb, atmega32m1,
atmega32u4, atmega32u6, atmega323, atmega324a, atmega324p, atmega324pa,
atmega324pb, atmega325, atmega325a, atmega325p, atmega325pa, atmega328,
atmega328p, atmega328pb, atmega329, atmega329a, atmega329p,
atmega329pa, atmega3250, atmega3250a, atmega3250p, atmega3250pa,
atmega3290, atmega3290a, atmega3290p, atmega3290pa, atmega406,
atmega64, atmega64a, atmega64c1, atmega64hve, atmega64hve2, atmega64m1,
atmega64rfr2, atmega640, atmega644, atmega644a, atmega644p,
atmega644pa, atmega644rfr2, atmega645, atmega645a, atmega645p,
atmega649, atmega649a, atmega649p, atmega6450, atmega6450a,
atmega6450p, atmega6490, atmega6490a, atmega6490p, ata5790, ata5790n,
ata5791, ata6613c, ata6614q, ata5782, ata5831, ata8210, ata8510, ata5787,
ata5835, ata5700m322, ata5702m322, at90pwm161, at90pwm216, at90pwm316,
at90can32, at90can64, at90scr100, at90usb646, at90usb647, at94k, m3000.

avr51
"Enhanced" devices with 128 KiB of program memory. *mcu* = atmega128, atmega128a,
atmega128rfa1, atmega128rfr2, atmega1280, atmega1281, atmega1284,
atmega1284p, atmega1284rfr2, at90can128, at90usb1286, at90usb1287.

avr6
"Enhanced" devices with 3−byte PC, i.e. with more than 128 KiB of program memory. *mcu* =
atmega256rfr2, atmega2560, atmega2561, atmega2564rfr2.

avrxmega2
"XMEGA" devices with more than 8 KiB and up to 64 KiB of program memory. *mcu* =
atxmega8e5, atxmega16a4, atxmega16a4u, atxmega16c4, atxmega16d4,
atxmega16e5, atxmega32a4, atxmega32a4u, atxmega32c3, atxmega32c4,
atxmega32d3, atxmega32d4, atxmega32e5, avr64da28, avr64da32, avr64da48,
avr64da64, avr64db28, avr64db32, avr64db48, avr64db64, avr64dd14,
avr64dd20, avr64dd28, avr64dd32, avr64du28, avr64du32, avr64ea28,
avr64ea32, avr64ea48, avr64sd28, avr64sd32, avr64sd48.

avrxmega3
"XMEGA" devices with up to 64 KiB of combined program memory and RAM, and with
program memory visible in the RAM address space. *mcu* = attiny202, attiny204,
attiny212, attiny214, attiny402, attiny404, attiny406, attiny412,
attiny414, attiny416, attiny416auto, attiny417, attiny424, attiny426,
attiny427, attiny804, attiny806, attiny807, attiny814, attiny816,
attiny817, attiny824, attiny826, attiny827, attiny1604, attiny1606,
attiny1607, attiny1614, attiny1616, attiny1617, attiny1624, attiny1626,
attiny1627, attiny3214, attiny3216, attiny3217, attiny3224, attiny3226,
attiny3227, atmega808, atmega809, atmega1608, atmega1609, atmega3208,
atmega3209, atmega4808, atmega4809, avr16dd14, avr16dd20, avr16dd28,
avr16dd32, avr16du14, avr16du20, avr16du28, avr16du32, avr16ea28,
avr16ea32, avr16ea48, avr16eb14, avr16eb20, avr16eb28, avr16eb32,
avr32da28, avr32da32, avr32da48, avr32db28, avr32db32, avr32db48,
avr32dd14, avr32dd20, avr32dd28, avr32dd32, avr32du14, avr32du20,
avr32du28, avr32du32, avr32ea28, avr32ea32, avr32ea48, avr32sd20,
avr32sd28, avr32sd32.

avrxmega4
"XMEGA" devices with more than 64 KiB and up to 128 KiB of program memory. *mcu* =
atxmega64a3, atxmega64a3u, atxmega64a4u, atxmega64b1, atxmega64b3,
atxmega64c3, atxmega64d3, atxmega64d4, avr128da28, avr128da32,
avr128da48, avr128da64, avr128db28, avr128db32, avr128db48, avr128db64.

avrxmega5
    "XMEGA" devices with more than 64 KiB and up to 128 KiB of program memory and more than 64 KiB of RAM. *mcu* = `atxmega64a1`, `atxmega64a1u`.

avrxmega6
    "XMEGA" devices with more than 128 KiB of program memory. *mcu* = `atxmega128a3`, `atxmega128a3u`, `atxmega128b1`, `atxmega128b3`, `atxmega128c3`, `atxmega128d3`, `atxmega128d4`, `atxmega192a3`, `atxmega192a3u`, `atxmega192c3`, `atxmega192d3`, `atxmega256a3`, `atxmega256a3b`, `atxmega256a3bu`, `atxmega256a3u`, `atxmega256c3`, `atxmega256d3`, `atxmega384c3`, `atxmega384d3`.

avrxmega7
    "XMEGA" devices with more than 128 KiB of program memory and more than 64 KiB of RAM. *mcu* = `atxmega128a1`, `atxmega128a1u`, `atxmega128a4u`.

avrtiny
    "Reduced Tiny" Tiny core devices with only 16 general purpose registers and 512 B up to 4 KiB of program memory. *mcu* = `attiny4`, `attiny5`, `attiny9`, `attiny10`, `attiny102`, `attiny104`, `attiny20`, `attiny40`.

avr1
    This ISA is implemented by the minimal AVR core and supported for assembler only. *mcu* = `attiny11`, `attiny12`, `attiny15`, `attiny28`, `at90s1200`.

−mabsdata
    Assume that all data in static storage can be accessed by LDS / STS instructions. This option has only an effect on reduced Tiny devices like ATtiny40. See also the `absdata` **AVR Variable Attributes,variable attribute**.

−mcvt
    Use a *compact vector table*. Some devices support a CVT with only four entries: 0=Reset, 1=NMI, 2=Prio1 IRQ, 3=Prio0 IRQs. This option will link startup code from `crt`*mcu*`-cvt.o` instead of the usual `crt`*mcu*`.o`. Apart from providing a compact vector table, the startup code will set bit `CPUINT_CTRLA.CPUINT_CVT` which enables the CVT on the device.

    When you do not want the startup code to set `CPUINT_CTRLA.CPUINT_CVT`, then you can satisfy symbol `__init_cvt` so that the respective code is no more pulled in from `lib`*mcu*`.a`. For example, you can link with `-Wl,--defsym,__init_cvt=0`.

    The CVT startup code is available since AVR−LibC v2.3 (`https://github.com/avrdudes/avr-libc/issues/1010`).

−mdouble=bits
−mlong−double=bits
    Set the size (in bits) of the `double` or `long double` type, respectively. Possible values for *bits* are 32 and 64. Whether or not a specific value for *bits* is allowed depends on the `--with-double=` and `--with-long-double=` configure options (`https://gcc.gnu.org/install/configure.html#avr`), and the same applies for the default values of the options.

−mgas−isr−prologues
    Interrupt service routines (ISRs) may use the `__gcc_isr` pseudo instruction supported by GNU Binutils. If this option is on, the feature can still be disabled for individual ISRs by means of the **AVR Function Attributes,no_gccisr** function attribute. This feature is activated per default if optimization is on (but not with **−Og**, `@pxref{Optimize Options}`), and if GNU Binutils support PR21683 (`https://sourceware.org/PR21683`).

−mint8
    Assume `int` to be 8−bit integer. This affects the sizes of all types: a `char` is 1 byte, an `int` is 1 byte, a `long` is 2 bytes, and `long long` is 4 bytes. Please note that this option does not conform to the C standards, but it results in smaller code size.

−mmain−is−OS_task

Do not save registers in main. The effect is the same like attaching attribute **AVR Function Attributes„OS_task** to main. It is activated per default if optimization is on.

−mno−call−main

Don't run main by means of

```
        XCALL   main
        XJMP    exit
```

Instead, put main in section .init9 (https://avrdudes.github.io/avr-libc/avr-libc-user-manual/mem_sections.html#sec_d so that no call is required. By setting this option the user asserts that main will not return.

This option can be used for devices with very limited resources in order to save a few bytes of code and stack space. It will work as expected since AVR−LibC v2.3 (https://github.com/avrdudes/avr-libc/issues/1012). With older versions, there will be no performance gain.

−mno−interrupts

Generated code is not compatible with hardware interrupts. Code size is smaller.

−mrelax

Try to replace CALL resp. JMP instruction by the shorter RCALL resp. RJMP instruction if applicable. Setting **−mrelax** just adds the **−−mlink−relax** option to the assembler's command line and the **−−relax** option to the linker's command line.

Jump relaxing is performed by the linker because jump offsets are not known before code is located. Therefore, the assembler code generated by the compiler is the same, but the instructions in the executable may differ from instructions in the assembler code.

Relaxing must be turned on if linker stubs are needed, see the section on EIND and linker stubs below.

−mrodata−in−ram
−mno−rodata−in−ram

Locate the .rodata sections for read−only data in RAM resp. in program memory. For most devices, there is no choice and this option acts rather like an assertion.

Since v14 and for the AVR64* and AVR128* devices, .rodata is located in flash memory per default, provided the required GNU Binutils support (PR31124 (https://sourceware.org/PR31124)) is available. In that case, **−mrodata−in−ram** can be used to return to the old layout with .rodata in RAM.

−mtiny−stack

Only change the lower 8 bits of the stack pointer.

−mfract−convert−truncate

Allow to use truncation instead of rounding towards zero for fractional fixed−point types.

−nodevicelib

Don't link against AVR−LibC's device specific library lib*mcu*.a.

Notice that since AVR−LibC v2.3, that library contains code that is essential for the correct functioning of a program. In particular, it contains parts of the startup code like: __init_sp (https://github.com/avrdudes/avr-libc/issues/1011) to initialize the stack pointer with symbol __stack, __init_cvt (https://github.com/avrdudes/avr-libc/issues/1010) to set up the hardware to use a compact vector table with **−mcvt**, __call_main (https://github.com/avrdudes/avr-libc/issues/1012) to call main and exit, and __do_flmap_init (https://github.com/avrdudes/avr-libc/issues/931) to set up FLMAP according to symbol __flmap.

−nodevicespecs

Don't add **−specs=device−specs/specs−***mcu* to the compiler driver's command line. The user takes responsibility for supplying the sub−processes like compiler proper, assembler and linker with appropriate command line options. This means that the user has to supply her private device specs file by means of **−specs=***path−to−specs−file*. There is no more need for option **−mmcu=***mcu*.

This option can also serve as a replacement for the older way of specifying custom device−specs files that needed **−B** *some−path* to point to a directory which contains a folder named `device-specs` which contains a specs file named `specs-mcu`, where *mcu* was specified by **−mmcu=***mcu*.

−Waddr−space−convert

Warn about conversions between address spaces in the case where the resulting address space is not contained in the incoming address space.

−Wmisspelled−isr

Warn if the ISR is misspelled, i.e. without `__vector` prefix. Enabled by default.

AVR Optimization Options

The following options are pure optimization options. Options **−mgas−isr−prologues**, **−mmain−is−OS_task**, **−mno−call−main** and **−mrelax** from above are only *almost* optimization options, since there are rare occasions where their different code generation matters.

−maccumulate−args

Accumulate outgoing function arguments and acquire/release the needed stack space for outgoing function arguments once in function prologue/epilogue. Without this option, outgoing arguments are pushed before calling a function and popped afterwards. See also the **−fdefer−pop Optimize Options,,optimization option**.

Popping the arguments after the function call can be expensive on AVR so that accumulating the stack space might lead to smaller executables because arguments need not be removed from the stack after such a function call.

This option can lead to reduced code size for functions that perform several calls to functions that get their arguments on the stack like calls to printf−like functions.

−mbranch−cost=*cost*

Set the branch costs for conditional branch instructions to *cost*. Reasonable values for *cost* are small, non−negative integers. The default branch cost is 0.

−mcall−prologues

Functions prologues/epilogues are expanded as calls to appropriate subroutines. Code size is smaller.

−mfuse−add
−mno−fuse−add
−mfuse−add=*level*

Optimize indirect memory accesses on reduced Tiny devices. The default uses `level=1` for optimizations **−Og** and **−O1**, and `level=2` for higher optimizations. Valid values for *level* are `0`, `1` and `2`.

−mfuse−move
−mno−fuse−move
−mfuse−move=*level*

Run a post reload optimization pass that tries to fuse move instructions and to split multi−byte instructions into 8−bit operations. The default uses `level=3` for optimization **−O1**, and `level=23` for higher optimizations. Valid values for *level* are in the range `0 ... 23` which is a 3:2:2:2 mixed radix value. Each digit controls some aspect of the optimization.

−mstrict−X

Use address register `X` in a way proposed by the hardware. This means that `X` is only used in indirect, post−increment or pre−decrement addressing.

Without this option, the X register may be used in the same way as Y or Z which then is emulated by additional instructions. For example, loading a value with X+const addressing with a small non−negative const < 64 to a register *Rn* is performed as

```
adiw r26, const   ; X += const
ld   <Rn>, X      ; <Rn> = *X
sbiw r26, const   ; X -= const
```

−msplit−bit−shift

Split multi−byte shifts with a constant offset into a shift with a byte offset and a residual shift with a non−byte offset. This optimization is turned on per default for **−O2** and higher, including **−Os** but excluding **−Oz**. Splitting of shifts with a constant offset that is a multiple of 8 is controlled by **−mfuse−move**.

−msplit−ldst

Split multi−byte loads and stores into several byte loads and stores. This optimization is turned on per default for **−O2** and higher.

−muse−nonzero−bits

Enable optimizations that are only possible when some bits in a register are always zero. This optimization is turned on per default for **−O2** and higher.

EIND and Devices with More Than 128 Ki Bytes of Flash

Pointers in the implementation are 16 bits wide. The address of a function or label is represented as word address so that indirect jumps and calls can target any code address in the range of 64 Ki words.

In order to facilitate indirect jump on devices with more than 128 Ki bytes of program memory space, there is a special function register called EIND that serves as most significant part of the target address when EICALL or EIJMP instructions are used.

Indirect jumps and calls on these devices are handled as follows by the compiler and are subject to some limitations:

*   The compiler never sets EIND.

*   The compiler uses EIND implicitly in EICALL/EIJMP instructions or might read EIND directly in order to emulate an indirect call/jump by means of a RET instruction.

*   The compiler assumes that EIND never changes during the startup code or during the application. In particular, EIND is not saved/restored in function or interrupt service routine prologue/epilogue.

*   For indirect calls to functions and computed goto, the linker generates *stubs*. Stubs are jump pads sometimes also called *trampolines*. Thus, the indirect call/jump jumps to such a stub. The stub contains a direct jump to the desired address.

*   Linker relaxation must be turned on so that the linker generates the stubs correctly in all situations. See the compiler option **−mrelax** and the linker option **−−relax**. There are corner cases where the linker is supposed to generate stubs but aborts without relaxation and without a helpful error message.

*   The default linker script is arranged for code with EIND = 0. If code is supposed to work for a setup with EIND != 0, a custom linker script has to be used in order to place the sections whose name start with .trampolines into the segment where EIND points to.

*   The startup code from libgcc never sets EIND. Notice that startup code is a blend of code from libgcc and AVR−LibC. For the impact of AVR−LibC on EIND, see the AVR−LibC user manual (https://avrdudes.github.io/avr-libc/avr-libc-user-manual/).

*   It is legitimate for user−specific startup code to set up EIND early, for example by means of initialization code located in section .init3. Such code runs prior to general startup code that initializes RAM and calls constructors, but after the bit of startup code from AVR−LibC that sets EIND to the segment where the vector table is located.

```
#include <avr/io.h>
```

```
static void
__attribute__((section(".init3"),naked,used,no_instrument_function))
init3_set_eind (void)
{
  __asm volatile ("ldi r24,pm_hh8(__trampolines_start)\n\t"
                  "out %i0,r24" :: "n" (&EIND) : "r24","memory");
}
```

The `__trampolines_start` symbol is defined in the linker script.

* Stubs are generated automatically by the linker if the following two conditions are met:

   −<The address of a label is taken by means of the gs modifier>
      (short for *generate stubs*) like so:

```
LDI r24, lo8(gs(<func>))
LDI r25, hi8(gs(<func>))
```

   −<The final location of that label is in a code segment>
      outside the segment where the stubs are located.

* The compiler emits such gs modifiers for code labels in the following situations:

   −<Taking address of a function or code label.>
   −<Computed goto.>
   −<If prologue−save function is used, see **−mcall−prologues**>
      command−line option.

   −<Switch/case dispatch tables. If you do not want such dispatch>
      tables you can specify the **−fno−jump−tables** command−line option.

   −<C and C++ constructors/destructors called during startup/shutdown.>
   −<If the tools hit a gs () modifier explained above.>

* Jumping to non−symbolic addresses like so is *not* supported:

```
int main (void)
{
    /* Call function at word address 0x2 */
    return ((int(*)(void)) 0x2)();
}
```

Instead, a stub has to be set up, i.e. the function has to be called through a symbol (func_4 in the example):

```
int main (void)
{
    extern int func_4 (void);

    /* Call function at byte address 0x4 */
    return func_4();
}
```

and the application be linked with **−Wl,−−defsym,func_4=0x4**. Alternatively, func_4 can be defined in the linker script.

Handling of the RAMPD, RAMPX, RAMPY and RAMPZ Special Function Registers

Some AVR devices support memories larger than the 64 KiB range that can be accessed with 16−bit pointers. To access memory locations outside this 64 KiB range, the content of a RAMP register is used as high part of the address: The X, Y, Z address register is concatenated with the RAMPX, RAMPY, RAMPZ special function register, respectively, to get a wide address. Similarly, RAMPD is used together with direct

addressing.

* The startup code initializes the RAMP special function registers with zero.

* If a **AVR Named Address Spaces,named address space** other than generic or `__flash` is used, then `RAMPZ` is set as needed before the operation.

* If the device supports RAM larger than 64 KiB and the compiler needs to change `RAMPZ` to accomplish an operation, `RAMPZ` is reset to zero after the operation.

* If the device comes with a specific `RAMP` register, the ISR prologue/epilogue saves/restores that SFR and initializes it with zero in case the ISR code might (implicitly) use it.

* RAM larger than 64 KiB is not supported by GCC for AVR targets. If you use inline assembler to read from locations outside the 16−bit address range and change one of the `RAMP` registers, you must reset it to zero after the access.

AVR Built−in Macros

GCC defines several built−in macros so that the user code can test for the presence or absence of features. Almost any of the following built−in macros are deduced from device capabilities and thus triggered by the **−mmcu=** command−line option.

For even more AVR−specific built−in macros see **AVR Named Address Spaces** and **AVR Built−in Functions**.

`__AVR_ARCH__`
> Build−in macro that resolves to a decimal number that identifies the architecture and depends on the **−mmcu=***mcu* option. Possible values are:
>
> 2, 25, 3, 31, 35, 4, 5, 51, 6
>
> for *mcu*=`avr2, avr25, avr3, avr31, avr35, avr4, avr5, avr51, avr6`,
>
> respectively and
>
> 100, 102, 103, 104, 105, 106, 107
>
> for *mcu*=`avrtiny, avrxmega2, avrxmega3, avrxmega4, avrxmega5, avrxmega6, avrxmega7`, respectively. If *mcu* specifies a device, this built−in macro is set accordingly. For example, with **−mmcu=atmega8** the macro is defined to 4.

`__AVR_`*Device*`__`
> Setting **−mmcu=***device* defines this built−in macro which reflects the device's name. For example, **−mmcu=atmega8** defines the built−in macro `__AVR_ATmega8__`, **−mmcu=attiny261a** defines `__AVR_ATtiny261A__`, etc.
>
> The built−in macros' names follow the scheme `__AVR_`*Device*`__` where *Device* is the device name as from the AVR user manual. The difference between *Device* in the built−in macro and *device* in **−mmcu=***device* is that the latter is always lowercase.
>
> If *device* is not a device but only a core architecture like **avr51**, this macro is not defined.

`__AVR_DEVICE_NAME__`
> Setting **−mmcu=***device* defines this built−in macro to the device's name. For example, with **−mmcu=atmega8** the macro is defined to `atmega8`.
>
> If *device* is not a device but only a core architecture like **avr51**, this macro is not defined.

`__AVR_CVT__`
> The code is being compiled with option −mcvt to use a *compact vector table*.

`__AVR_XMEGA__`
> The device / architecture belongs to the XMEGA family of devices.

`__AVR_HAVE_ADIW__`
    The device has the `ADIW` and `SBIW` instructions.

`__AVR_HAVE_ELPM__`
    The device has the `ELPM` instruction.

`__AVR_HAVE_ELPMX__`
    The device has the `ELPM R`*n*`,Z` and `ELPM R`*n*`,Z+` instructions.

`__AVR_HAVE_LPMX__`
    The device has the `LPM R`*n*`,Z` and `LPM R`*n*`,Z+` instructions.

`__AVR_HAVE_MOVW__`
    The device has the `MOVW` instruction to perform 16−bit register−register moves.

`__AVR_HAVE_MUL__`
    The device has a hardware multiplier.

`__AVR_HAVE_JMP_CALL__`
    The device has the `JMP` and `CALL` instructions. This is the case for devices with more than 8 KiB of
    program memory.

`__AVR_HAVE_EIJMP_EICALL__`
`__AVR_3_BYTE_PC__`
    The device has the `EIJMP` and `EICALL` instructions. This is the case for devices with more than 128
    KiB of program memory. This also means that the program counter (PC) is 3 bytes wide.

`__AVR_2_BYTE_PC__`
    The program counter (PC) is 2 bytes wide. This is the case for devices with up to 128 KiB of program
    memory.

`__AVR_HAVE_8BIT_SP__`
`__AVR_HAVE_16BIT_SP__`
    The stack pointer (SP) register is treated as 8−bit respectively 16−bit register by the compiler. The
    definition of these macros is affected by **−mtiny−stack**.

`__AVR_HAVE_SPH__`
`__AVR_SP8__`
    The device has the SPH (high part of stack pointer) special function register or has an 8−bit stack
    pointer, respectively. The definition of these macros is affected by **−mmcu=** and in the cases of
    −mmcu=avr2 and **−mmcu=avr25** also by **−msp8**.

`__AVR_HAVE_RAMPD__`
`__AVR_HAVE_RAMPX__`
`__AVR_HAVE_RAMPY__`
`__AVR_HAVE_RAMPZ__`
    The device has the `RAMPD`, `RAMPX`, `RAMPY`, `RAMPZ` special function register, respectively.

`__NO_INTERRUPTS__`
    This macro reflects the **−mno−interrupts** command−line option.

`__AVR_ERRATA_SKIP__`
`__AVR_ERRATA_SKIP_JMP_CALL__`
    Some AVR devices (AT90S8515, ATmega103) must not skip 32−bit instructions because of a
    hardware erratum. Skip instructions are `SBRS`, `SBRC`, `SBIS`, `SBIC` and `CPSE`. The second macro is
    only defined if `__AVR_HAVE_JMP_CALL__` is also set.

`__AVR_ISA_RMW__`
    The device has Read−Modify−Write instructions (XCH, LAC, LAS and LAT).

`__AVR_SFR_OFFSET__=`*offset*
    Instructions that can address I/O special function registers directly like `IN`, `OUT`, `SBI`, etc. may use a
    different address as if addressed by an instruction to access RAM like `LD` or `STS`. This offset depends

on the device architecture and has to be subtracted from the RAM address in order to get the respective I/O address.

`__AVR_SHORT_CALLS__`
The **−mshort−calls** command line option is set.

`__AVR_PM_BASE_ADDRESS__=`*addr*
Some devices support reading from flash memory by means of `LD*` instructions. The flash memory is seen in the data address space at an offset of `__AVR_PM_BASE_ADDRESS__`. If this macro is not defined, this feature is not available. If defined, the address space is linear and there is no need to put `.rodata` into RAM. This is handled by the default linker description file, and is currently available for `avrtiny` and `avrxmega3`. Even more convenient, there is no need to use address spaces like `__flash` or features like attribute `progmem` and `pgm_read_*`.

`__AVR_HAVE_FLMAP__`
This macro is defined provided the following conditions are met:

*<The device has the `NVMCTRL_CTRLB.FLMAP` bitfield.>
This applies to the AVR64* and AVR128* devices.

*<It's not known at assembler−time which emulation will be used.>

This implies the compiler was configured with GNU Binutils that implement PR31124 (`https://sourceware.org/PR31124`).

`__AVR_RODATA_IN_RAM__`
This macro is undefined when the code is compiled for a core architecture.

When the code is compiled for a device, the macro is defined to 1 when the `.rodata` sections for read−only data is located in RAM; and defined to 0, otherwise.

`__WITH_AVRLIBC__`
The compiler is configured to be used together with AVR−LibC. See the **−−with−avrlibc** configure option.

`__HAVE_SIGNAL_N__`
The compiler supports the `signal(`*num*`)` and `interrupt(`*num*`)` **AVR Function Attributes,,function attributes** with an argument *num* that specifies the number of the interrupt service routine.

`__HAVE_DOUBLE_MULTILIB__`
Defined if **−mdouble=** acts as a multilib option.

`__HAVE_DOUBLE32__`
`__HAVE_DOUBLE64__`
Defined if the compiler supports 32−bit double resp. 64−bit double. The actual layout is specified by option **−mdouble=**.

`__DEFAULT_DOUBLE__`
The size in bits of `double` if **−mdouble=** is not set. To test the layout of `double` in a program, use the built−in macro `__SIZEOF_DOUBLE__`.

`__HAVE_LONG_DOUBLE32__`
`__HAVE_LONG_DOUBLE64__`
`__HAVE_LONG_DOUBLE_MULTILIB__`
`__DEFAULT_LONG_DOUBLE__`
Same as above, but for `long double` instead of `double`.

`__WITH_DOUBLE_COMPARISON__`
Reflects the `--with-double-comparison={tristate|bool|libf7}` configure option (`https://gcc.gnu.org/install/configure.html#avr`) and is defined to 2 or 3.

```
__WITH_LIBF7_LIBGCC__
__WITH_LIBF7_MATH__
__WITH_LIBF7_MATH_SYMBOLS__
```
> Reflects the `--with-libf7={libgcc|math|math-symbols}` configure option (`https://gcc.gnu.org/install/configure.html#avr`).

AVR Internal Options

The following options are used internally by the compiler and to communicate between device specs files and the compiler proper. You don't need to set these options by hand, in particular they are not optimization options. Using these options in the wrong way may lead to sub−optimal or wrong code. They are documented for completeness, and in order to get a better understanding of device specs (`https://gcc.gnu.org/wiki/avr-gcc#spec-files`) files.

−mn−flash=*num*
> Assume that the flash memory has a size of *num* times 64 KiB. This determines which `__flash`*N* address spaces are available.

−mflmap
> The device has the `FLMAP` bit field located in special function register `NVMCTRL_CTRLB`.

−mrmw
> Assume that the device supports the Read−Modify−Write instructions `XCH`, `LAC`, `LAS` and `LAT`.

−mshort−calls
> Assume that `RJMP` and `RCALL` can target the whole program memory. This option is used for multilib generation and selection for the devices from architecture `avrxmega3`.

−mskip−bug
> Generate code without skips (`CPSE`, `SBRS`, `SBRC`, `SBIS`, `SBIC`) over 32−bit instructions.

−msp8
> Treat the stack pointer register as an 8−bit register, i.e. assume the high byte of the stack pointer is zero. This option is used by the compiler to select and build multilibs for architectures `avr2` and `avr25`. These architectures mix devices with and without `SPH`.

Blackfin Options

−mcpu=*cpu*[*−sirevision*]
> Specifies the name of the target Blackfin processor. Currently, *cpu* can be one of **bf512**, **bf514**, **bf516**, **bf518**, **bf522**, **bf523**, **bf524**, **bf525**, **bf526**, **bf527**, **bf531**, **bf532**, **bf533**, **bf534**, **bf536**, **bf537**, **bf538**, **bf539**, **bf542**, **bf544**, **bf547**, **bf548**, **bf549**, **bf542m**, **bf544m**, **bf547m**, **bf548m**, **bf549m**, **bf561**, **bf592**.
>
> The optional *sirevision* specifies the silicon revision of the target Blackfin processor. Any workarounds available for the targeted silicon revision are enabled. If *sirevision* is **none**, no workarounds are enabled. If *sirevision* is **any**, all workarounds for the targeted processor are enabled. The `__SILICON_REVISION__` macro is defined to two hexadecimal digits representing the major and minor numbers in the silicon revision. If *sirevision* is **none**, the `__SILICON_REVISION__` is not defined. If *sirevision* is **any**, the `__SILICON_REVISION__` is defined to be `0xffff`. If this optional *sirevision* is not used, GCC assumes the latest known silicon revision of the targeted Blackfin processor.
>
> GCC defines a preprocessor macro for the specified *cpu*. For the **bfin−elf** toolchain, this option causes the hardware BSP provided by libgloss to be linked in if **−msim** is not given.
>
> Without this option, **bf532** is used as the processor by default.
>
> Note that support for **bf561** is incomplete. For **bf561**, only the preprocessor macro is defined.

−msim
> Specifies that the program will be run on the simulator. This causes the simulator BSP provided by libgloss to be linked in. This option has effect only for **bfin−elf** toolchain. Certain other options, such

as **−mid−shared−library** and **−mfdpic**, imply **−msim**.

−momit−leaf−frame−pointer

Don't keep the frame pointer in a register for leaf functions. This avoids the instructions to save, set up and restore frame pointers and makes an extra register available in leaf functions.

−mspecld−anomaly

When enabled, the compiler ensures that the generated code does not contain speculative loads after jump instructions. If this option is used, `__WORKAROUND_SPECULATIVE_LOADS` is defined.

−mno−specld−anomaly

Don't generate extra code to prevent speculative loads from occurring.

−mcsync−anomaly

When enabled, the compiler ensures that the generated code does not contain CSYNC or SSYNC instructions too soon after conditional branches. If this option is used, `__WORKAROUND_SPECULATIVE_SYNCS` is defined.

−mno−csync−anomaly

Don't generate extra code to prevent CSYNC or SSYNC instructions from occurring too soon after a conditional branch.

−mlow64k

When enabled, the compiler is free to take advantage of the knowledge that the entire program fits into the low 64k of memory.

−mno−low64k

Assume that the program is arbitrarily large. This is the default.

−mstack−check−l1

Do stack checking using information placed into L1 scratchpad memory by the uClinux kernel.

−mid−shared−library

Generate code that supports shared libraries via the library ID method. This allows for execute in place and shared libraries in an environment without virtual memory management. This option implies **−fPIC**. With a **bfin−elf** target, this option implies **−msim**.

−mno−id−shared−library

Generate code that doesn't assume ID−based shared libraries are being used. This is the default.

−mleaf−id−shared−library

Generate code that supports shared libraries via the library ID method, but assumes that this library or executable won't link against any other ID shared libraries. That allows the compiler to use faster code for jumps and calls.

−mno−leaf−id−shared−library

Do not assume that the code being compiled won't link against any ID shared libraries. Slower code is generated for jump and call insns.

−mshared−library−id=n

Specifies the identification number of the ID−based shared library being compiled. Specifying a value of 0 generates more compact code; specifying other values forces the allocation of that number to the current library but is no more space− or time−efficient than omitting this option.

−msep−data

Generate code that allows the data segment to be located in a different area of memory from the text segment. This allows for execute in place in an environment without virtual memory management by eliminating relocations against the text section.

−mno−sep−data

Generate code that assumes that the data segment follows the text segment. This is the default.

−mlong−calls
−mno−long−calls
> Tells the compiler to perform function calls by first loading the address of the function into a register and then performing a subroutine call on this register. This switch is needed if the target function lies outside of the 24−bit addressing range of the offset−based version of subroutine call instruction.
>
> This feature is not enabled by default. Specifying **−mno−long−calls** restores the default behavior. Note these switches have no effect on how the compiler generates code to handle function calls via function pointers.

−mfast−fp
> Link with the fast floating−point library. This library relaxes some of the IEEE floating−point standard's rules for checking inputs against Not−a−Number (NAN), in the interest of performance.

−minline−plt
> Enable inlining of PLT entries in function calls to functions that are not known to bind locally. It has no effect without **−mfdpic**.

−mmulticore
> Build a standalone application for multicore Blackfin processors. This option causes proper start files and link scripts supporting multicore to be used, and defines the macro `__BFIN_MULTICORE`. It can only be used with **−mcpu=bf561**[−*sirevision*].
>
> This option can be used with **−mcorea** or **−mcoreb**, which selects the one−application−per−core programming model. Without **−mcorea** or **−mcoreb**, the single−application/dual−core programming model is used. In this model, the main function of Core B should be named as `coreb_main`.
>
> If this option is not used, the single−core application programming model is used.

−mcorea
> Build a standalone application for Core A of BF561 when using the one−application−per−core programming model. Proper start files and link scripts are used to support Core A, and the macro `__BFIN_COREA` is defined. This option can only be used in conjunction with **−mmulticore**.

−mcoreb
> Build a standalone application for Core B of BF561 when using the one−application−per−core programming model. Proper start files and link scripts are used to support Core B, and the macro `__BFIN_COREB` is defined. When this option is used, `coreb_main` should be used instead of `main`. This option can only be used in conjunction with **−mmulticore**.

−msdram
> Build a standalone application for SDRAM. Proper start files and link scripts are used to put the application into SDRAM, and the macro `__BFIN_SDRAM` is defined. The loader should initialize SDRAM before loading the application.

−micplb
> Assume that ICPLBs are enabled at run time. This has an effect on certain anomaly workarounds. For Linux targets, the default is to assume ICPLBs are enabled; for standalone applications the default is off.

C6X Options
−march=ance_name_
> This specifies the name of the target architecture. GCC uses this name to determine what kind of instructions it can emit when generating assembly code. Permissible names are: **c62x**, **c64x**, **c64x+**, **c67x**, **c67x+**, **c674x**.

−mbig−endian
> Generate code for a big−endian target.

−mlittle−endian
> Generate code for a little−endian target. This is the default.

−msim

Choose startup files and linker script suitable for the simulator.

−msdata=default

Put small global and static data in the `.neardata` section, which is pointed to by register `B14`. Put small uninitialized global and static data in the `.bss` section, which is adjacent to the `.neardata` section. Put small read−only data into the `.rodata` section. The corresponding sections used for large pieces of data are `.fardata`, `.far` and `.const`.

−msdata=all

Put all data, not just small objects, into the sections reserved for small data, and use addressing relative to the `B14` register to access them.

−msdata=none

Make no use of the sections reserved for small data, and use absolute addresses to access all data. Put all initialized global and static data in the `.fardata` section, and all uninitialized data in the `.far` section. Put all constant data into the `.const` section.

CRIS Options

These options are defined specifically for the CRIS ports.

−march=*architecture−type*
−mcpu=*architecture−type*

Generate code for the specified architecture. The choices for *architecture−type* are **v3**, **v8** and **v10** for respectively ETRAX 4, ETRAX 100, and ETRAX 100 LX. Default is **v0**.

−mtune=*architecture−type*

Tune to *architecture−type* everything applicable about the generated code, except for the ABI and the set of available instructions. The choices for *architecture−type* are the same as for **−march=***architecture−type*.

−mmax−stack−frame=*n*

Warn when the stack frame of a function exceeds *n* bytes.

−metrax4
−metrax100

The options **−metrax4** and **−metrax100** are synonyms for **−march=v3** and **−march=v8** respectively.

−mmul−bug−workaround
−mno−mul−bug−workaround

Work around a bug in the `muls` and `mulu` instructions for CPU models where it applies. This option is disabled by default.

−mpdebug

Enable CRIS−specific verbose debug−related information in the assembly code. This option also has the effect of turning off the **#NO_APP** formatted−code indicator to the assembler at the beginning of the assembly file.

−mcc−init

Do not use condition−code results from previous instruction; always emit compare and test instructions before use of condition codes.

−mno−side−effects

Do not emit instructions with side effects in addressing modes other than post−increment.

−mstack−align
−mno−stack−align
−mdata−align
−mno−data−align
−mconst−align

−mno−const−align

These options (**no−** options) arrange (eliminate arrangements) for the stack frame, individual data and constants to be aligned for the maximum single data access size for the chosen CPU model. The default is to arrange for 32−bit alignment. ABI details such as structure layout are not affected by these options.

−m32−bit
−m16−bit
−m8−bit

Similar to the stack− data− and const−align options above, these options arrange for stack frame, writable data and constants to all be 32−bit, 16−bit or 8−bit aligned. The default is 32−bit alignment.

−mno−prologue−epilogue
−mprologue−epilogue

With **−mno−prologue−epilogue**, the normal function prologue and epilogue which set up the stack frame are omitted and no return instructions or return sequences are generated in the code. Use this option only together with visual inspection of the compiled code: no warnings or errors are generated when call−saved registers must be saved, or storage for local variables needs to be allocated.

−melf

Legacy no−op option.

−sim

This option arranges to link with input−output functions from a simulator library. Code, initialized data and zero−initialized data are allocated consecutively.

−sim2

Like **−sim**, but pass linker options to locate initialized data at 0x40000000 and zero−initialized data at 0x80000000.

C−SKY Options

GCC supports these options when compiling for C−SKY V2 processors.

−march=*arch*

Specify the C−SKY target architecture. Valid values for *arch* are: **ck801**, **ck802**, **ck803**, **ck807**, and **ck810**. The default is **ck810**.

−mcpu=*cpu*

Specify the C−SKY target processor. Valid values for *cpu* are: **ck801**, **ck801t**, **ck802**, **ck802t**, **ck802j**, **ck803**, **ck803h**, **ck803t**, **ck803ht**, **ck803f**, **ck803fh**, **ck803e**, **ck803eh**, **ck803et**, **ck803eht**, **ck803ef**, **ck803efh**, **ck803ft**, **ck803eft**, **ck803efht**, **ck803r1**, **ck803hr1**, **ck803tr1**, **ck803htr1**, **ck803fr1**, **ck803fhr1**, **ck803er1**, **ck803ehr1**, **ck803etr1**, **ck803ehtr1**, **ck803efr1**, **ck803efhr1**, **ck803ftr1**, **ck803eftr1**, **ck803efhtr1**, **ck803s**, **ck803st**, **ck803se**, **ck803sf**, **ck803sef**, **ck803seft**, **ck807e**, **ck807ef**, **ck807**, **ck807f**, **ck810e**, **ck810et**, **ck810ef**, **ck810eft**, **ck810**, **ck810v**, **ck810f**, **ck810t**, **ck810fv**, **ck810tv**, **ck810ft**, and **ck810ftv**.

−mbig−endian
−EB
−mlittle−endian
−EL

Select big− or little−endian code. The default is little−endian.

−mfloat−abi=*name*

Specifies which floating−point ABI to use. Permissible values are: **soft**, **softfp** and **hard**.

Specifying **soft** causes GCC to generate output containing library calls for floating−point operations. **softfp** allows the generation of code using hardware floating−point instructions, but still uses the soft−float calling conventions. **hard** allows generation of floating−point instructions and uses FPU−specific calling conventions.

The default depends on the specific target configuration. Note that the hard−float and soft−float ABIs

are not link−compatible; you must compile your entire program with the same ABI, and link with a compatible set of libraries.

−mhard−float
−msoft−float
Select hardware or software floating−point implementations. The default is soft float.

−mdouble−float
−mno−double−float
When **−mhard−float** is in effect, enable generation of double−precision float instructions. This is the default except when compiling for CK803.

−mfdivdu
−mno−fdivdu
When **−mhard−float** is in effect, enable generation of `frecipd`, `fsqrtd`, and `fdivd` instructions. This is the default except when compiling for CK803.

−mfpu=*fpu*
Select the floating−point processor. This option can only be used with **−mhard−float**. Values for *fpu* are **fpv2_sf** (equivalent to **−mno−double−float −mno−fdivdu**), **fpv2** (**−mdouble−float −mno−divdu**), and **fpv2_divd** (**−mdouble−float −mdivdu**).

−melrw
−mno−elrw
Enable the extended `lrw` instruction. This option defaults to on for CK801 and off otherwise.

−mistack
−mno−istack
Enable interrupt stack instructions; the default is off.

The **−mistack** option is required to handle the `interrupt` and `isr` function attributes.

−mmp
Enable multiprocessor instructions; the default is off.

−mcp
Enable coprocessor instructions; the default is off.

−mcache
Enable coprocessor instructions; the default is off.

−msecurity
Enable C−SKY security instructions; the default is off.

−mtrust
Enable C−SKY trust instructions; the default is off.

−mdsp
−medsp
−mvdsp
Enable C−SKY DSP, Enhanced DSP, or Vector DSP instructions, respectively. All of these options default to off.

−mdiv
−mno−div
Generate divide instructions. Default is off.

−msmart
−mno−smart
Generate code for Smart Mode, using only registers numbered 0−7 to allow use of 16−bit instructions. This option is ignored for CK801 where this is the required behavior, and it defaults to on for CK802. For other targets, the default is off.

−mhigh−registers
−mno−high−registers
> Generate code using the high registers numbered 16–31. This option is not supported on CK801, CK802, or CK803, and is enabled by default for other processors.

−manchor
−mno−anchor
> Generate code using global anchor symbol addresses.

−mpushpop
−mno−pushpop
> Generate code using `push` and `pop` instructions. This option defaults to on.

−mmultiple−stld
−mstm
−mno−multiple−stld
−mno−stm
> Generate code using `stm` and `ldm` instructions. This option isn't supported on CK801 but is enabled by default on other processors.

−mconstpool
−mno−constpool
> Create constant pools in the compiler instead of deferring it to the assembler. This option is the default and required for correct code generation on CK801 and CK802, and is optional on other processors.

−mstack−size
−mno−stack−size
> Emit `.stack_size` directives for each function in the assembly output. This option defaults to off.

−mccrt
−mno−ccrt
> Generate code for the C−SKY compiler runtime instead of libgcc. This option defaults to off.

−mbranch−cost=*n*
> Set the branch costs to roughly `n` instructions. The default is 1.

−msched−prolog
−mno−sched−prolog
> Permit scheduling of function prologue and epilogue sequences. Using this option can result in code that is not compliant with the C−SKY V2 ABI prologue requirements and that cannot be debugged or backtraced. It is disabled by default.

−msim
> Links the library libsemi.a which is in compatible with simulator. Applicable to ELF compiler only.

Cygwin and MinGW Options
> These additional options are available for Microsoft Windows targets:

−mconsole
> This option specifies that a console application is to be generated, by instructing the linker to set the PE header subsystem type required for console applications. This option is available for Cygwin and MinGW targets and is enabled by default on those targets.

−mcrtdll=*library*
> Preprocess, compile or link with specified C RunTime DLL *library*. This option adjust predefined macros `__CRTDLL__`, `__MSVCRT__`, `_UCRT` and `__MSVCRT_VERSION__` for specified CRT *library*, choose start file for CRT *library* and link with CRT *library*. Recognized CRT library names for proprocessor are: `crtdll*`, `msvcrt10*`, `msvcrt20*`, `msvcrt40*`, `msvcr40*`, `msvcrtd*`, `msvcrt-os*`, `msvcr70*`, `msvcr71*`, `msvcr80*`, `msvcr90*`, `msvcr100*`, `msvcr110*`, `msvcr120*` and `ucrt*`. If this options is not specified then the default MinGW import library `msvcrt` is used for linking and no other adjustment for preprocessor is done. MinGW

import library `msvcrt` is just a symlink to (or a copy of) another MinGW CRT import library chosen during MinGW compilation. MinGW import library `msvcrt-os` is for Windows system CRT DLL library `msvcrt.dll` and in most cases is the default MinGW import library. Generally speaking, changing the CRT DLL requires recompiling the entire MinGW CRT. This option is for experimental and testing purposes only. This option is available for MinGW targets.

−mdll

This option is available for Cygwin and MinGW targets. It specifies that a DLL−−−a dynamic link library−−−is to be generated, enabling the selection of the required runtime startup object and entry point.

−mnop−fun−dllimport

This option is available for Cygwin and MinGW targets. It specifies that the `dllimport` attribute should be ignored.

−mthreads

This option is available for MinGW targets. It specifies that MinGW−specific thread support is to be used.

−municode

This option is available for MinGW−w64 targets. It causes the `UNICODE` preprocessor macro to be predefined, and chooses Unicode−capable runtime startup code.

−mwin32

This option is available for Cygwin and MinGW targets. It specifies that the typical Microsoft Windows predefined macros are to be set in the pre−processor, but does not influence the choice of runtime library/startup code.

−mwindows

This option is available for Cygwin and MinGW targets. It specifies that a GUI application is to be generated by instructing the linker to set the PE header subsystem type appropriately.

−fno−set−stack−executable

This option is available for MinGW targets. It specifies that the executable flag for the stack used by nested functions isn't set. This is necessary for binaries running in kernel mode of Microsoft Windows, as there the User32 API, which is used to set executable privileges, isn't available.

−fwritable−relocated−rdata

This option is available for MinGW and Cygwin targets. It specifies that relocated−data in read−only section is put into the `.data` section. This is a necessary for older runtimes not supporting modification of `.rdata` sections for pseudo−relocation.

−mpe−aligned−commons

This option is available for Cygwin and MinGW targets. It specifies that the GNU extension to the PE file format that permits the correct alignment of COMMON variables should be used when generating code. It is enabled by default if GCC detects that the target assembler found during configuration supports the feature.

See also under **x86 Options** for standard options.

Darwin Options

These options are defined for all architectures running the Darwin operating system.

FSF GCC on Darwin does not create "fat" object files; it creates an object file for the single architecture that GCC was built to target. Apple's GCC on Darwin does create "fat" files if multiple **−arch** options are used; it does so by running the compiler or linker multiple times and joining the results together with *lipo*.

The subtype of the file created (like **ppc7400** or **ppc970** or **i686**) is determined by the flags that specify the ISA that GCC is targeting, like **−mcpu** or **−march**. The **−force_cpusubtype_ALL** option can be used to override this.

The Darwin tools vary in their behavior when presented with an ISA mismatch. The assembler, *as*, only permits instructions to be used that are valid for the subtype of the file it is generating, so you cannot put

64−bit instructions in a **ppc750** object file. The linker for shared libraries, */usr/bin/libtool*, fails and prints an error if asked to create a shared library with a less restrictive subtype than its input files (for instance, trying to put a **ppc970** object file in a **ppc7400** library). The linker for executables, **ld**, quietly gives the executable the most restrictive subtype of any of its input files.

−F*dir*

>   Add the framework directory *dir* to the head of the list of directories to be searched for header files. These directories are interleaved with those specified by **−I** options and are scanned in a left−to−right order.

>   A framework directory is a directory with frameworks in it. A framework is a directory with a *Headers* and/or *PrivateHeaders* directory contained directly in it that ends in *.framework*. The name of a framework is the name of this directory excluding the *.framework*. Headers associated with the framework are found in one of those two directories, with *Headers* being searched first. A subframework is a framework directory that is in a framework's *Frameworks* directory. Includes of subframework headers can only appear in a header of a framework that contains the subframework, or in a sibling subframework header. Two subframeworks are siblings if they occur in the same framework. A subframework should not have the same name as a framework; a warning is issued if this is violated. Currently a subframework cannot have subframeworks; in the future, the mechanism may be extended to support this. The standard frameworks can be found in */System/Library/Frameworks* and */Library/Frameworks*. An example include looks like `#include <Framework/header.h>`, where *Framework* denotes the name of the framework and *header.h* is found in the *PrivateHeaders* or *Headers* directory.

−iframework*dir*

>   Like **−F** except the directory is a treated as a system directory. The main difference between this **−iframework** and **−F** is that with **−iframework** the compiler does not warn about constructs contained within header files found via *dir*. This option is valid only for the C family of languages.

−gused

>   Emit debugging information for symbols that are used. For stabs debugging format, this enables **−feliminate−unused−debug−symbols**. This is by default ON.

−gfull

>   Emit debugging information for all symbols and types.

−fconstant−cfstrings

>   The **−fconstant−cfstrings** is an alias for **−mconstant−cfstrings**.

−mconstant−cfstrings

>   When the NeXT runtime is being used (the default on these systems), override any **−fconstant−string−class** setting and cause `@"..."` literals to be laid out as constant CoreFoundation strings.

−mmacosx−version−min=*version*

>   The earliest version of MacOS X that this executable will run on is *version*. Typical values supported for *version* include `12`, `10.12`, and `10.5.8`.

>   If the compiler was built to use the system's headers by default, then the default for this option is the system version on which the compiler is running, otherwise the default is to make choices that are compatible with as many systems and code bases as possible.

−mkernel

>   Enable kernel development mode. The **−mkernel** option sets **−static**, **−fno−common**, **−fno−use−cxa−atexit**, **−fno−exceptions**, **−fno−non−call−exceptions**, **−fapple−kext**, **−fno−weak** and **−fno−rtti** where applicable. This mode also sets **−mno−altivec**, **−msoft−float**, **−fno−builtin** and **−mlong−branch** for PowerPC targets.

−mone−byte−bool

>   Override the defaults for `bool` so that `sizeof(bool)==1`. By default `sizeof(bool)` is 4 when compiling for Darwin/PowerPC and `1` when compiling for Darwin/x86, so this option has no effect on

x86.

Warning: The **−mone−byte−bool** switch causes GCC to generate code that is not binary compatible with code generated without that switch. Using this switch may require recompiling all other modules in a program, including system libraries. Use this switch to conform to a non−default data model.

−mfix−and−continue
−ffix−and−continue
−findirect−data

Generate code suitable for fast turnaround development, such as to allow GDB to dynamically load *.o* files into already−running programs. **−findirect−data** and **−ffix−and−continue** are provided for backwards compatibility.

−all_load

Loads all members of static archive libraries. See man **ld** (1) for more information.

−arch_errors_fatal

Cause the errors having to do with files that have the wrong architecture to be fatal.

−bind_at_load

Causes the output file to be marked such that the dynamic linker will bind all undefined references when the file is loaded or launched.

−bundle

Produce a Mach−o bundle format file. See man **ld** (1) for more information.

−bundle_loader *executable*

This option specifies the *executable* that will load the build output file being linked. See man **ld** (1) for more information.

−dynamiclib

When passed this option, GCC produces a dynamic library instead of an executable when linking, using the Darwin *libtool* command.

−force_cpusubtype_ALL

This causes GCC's output file to have the **ALL** subtype, instead of one controlled by the **−mcpu** or **−march** option.

−nodefaultrpaths

Do not add default run paths for the compiler library directories to executables, modules or dynamic libraries. On macOS 10.5 and later, the embedded runpath is added by default unless the user adds **−nodefaultrpaths** to the link line. Run paths are needed (and therefore enforced) to build on macOS version 10.11 or later.

−allowable_client *client_name*
−client_name
−compatibility_version
−current_version
−dead_strip
−dependency−file
−dylib_file
−dylinker_install_name
−dynamic
−exported_symbols_list
−filelist
−flat_namespace
−force_flat_namespace
−headerpad_max_install_names
−image_base

–init
–install_name
–keep_private_externs
–multi_module
–multiply_defined
–multiply_defined_unused
–noall_load
–no_dead_strip_inits_and_terms
–nofixprebinding
–nomultidefs
–noprebind
–noseglinkedit
–pagezero_size
–prebind
–prebind_all_twolevel_modules
–private_bundle
–read_only_relocs
–sectalign
–sectobjectsymbols
–whyload
–seg1addr
–sectcreate
–sectobjectsymbols
–sectorder
–segaddr
–segs_read_only_addr
–segs_read_write_addr
–seg_addr_table
–seg_addr_table_filename
–seglinkedit
–segprot
–segs_read_only_addr
–segs_read_write_addr
–single_module
–static
–sub_library
–sub_umbrella
–twolevel_namespace
–umbrella
–undefined
–unexported_symbols_list
–weak_reference_mismatches
–whatsloaded
> These options are passed to the Darwin linker. The Darwin linker man page describes them in detail.

DEC Alpha Options
> These **–m** options are defined for the DEC Alpha implementations:

–mno−soft−float
–msoft−float
> Use (do not use) the hardware floating−point instructions for floating−point operations. When **–msoft−float** is specified, functions in *libgcc.a* are 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 issue floating−point operations. If you are compiling for an Alpha without floating−point operations, you must ensure that the library is built so as not to call them.

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

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

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

−mieee
The Alpha architecture implements floating−point hardware optimized for maximum performance. It is mostly compliant with the IEEE floating−point standard. However, for full compliance, software assistance is required. This option generates code fully IEEE−compliant code *except* that the *inexact−flag* is not maintained (see below). If this option is turned on, the preprocessor macro `_IEEE_FP` is defined during compilation. The resulting code is less efficient but is able to correctly support denormalized numbers and exceptional IEEE values such as not−a−number and plus/minus infinity. Other Alpha compilers call this option **−ieee_with_no_inexact**.

−mieee−with−inexact
This is like **−mieee** except the generated code also maintains the IEEE *inexact−flag*. Turning on this option causes the generated code to implement fully−compliant IEEE math. In addition to `_IEEE_FP`, `_IEEE_FP_EXACT` is defined as a preprocessor macro. On some Alpha implementations the resulting code may execute significantly slower than the code generated by default. Since there is very little code that depends on the *inexact−flag*, you should normally not specify this option. Other Alpha compilers call this option **−ieee_with_inexact**.

−mfp−trap−mode=_trap−mode_
This option controls what floating−point related traps are enabled. Other Alpha compilers call this option **−fptm** *trap−mode*. The trap mode can be set to one of four values:

n   This is the default (normal) setting. The only traps that are enabled are the ones that cannot be disabled in software (e.g., division by zero trap).

u   In addition to the traps enabled by **n**, underflow traps are enabled as well.

su   Like **u**, but the instructions are marked to be safe for software completion (see Alpha architecture manual for details).

sui Like **su**, but inexact traps are enabled as well.

−mfp−rounding−mode=_rounding−mode_
Selects the IEEE rounding mode. Other Alpha compilers call this option **−fprm** *rounding−mode*. The *rounding−mode* can be one of:

n   Normal IEEE rounding mode. Floating−point numbers are rounded towards the nearest machine number or towards the even machine number in case of a tie.

m   Round towards minus infinity.

c   Chopped rounding mode. Floating−point numbers are rounded towards zero.

d   Dynamic rounding mode. A field in the floating−point control register (*fpcr*, see Alpha architecture reference manual) controls the rounding mode in effect. The C library initializes this register for rounding towards plus infinity. Thus, unless your program modifies the *fpcr*, **d** corresponds to round towards plus infinity.

−mtrap−precision=*trap−precision*

    In the Alpha architecture, floating−point traps are imprecise. This means without software assistance it is impossible to recover from a floating trap and program execution normally needs to be terminated. GCC can generate code that can assist operating system trap handlers in determining the exact location that caused a floating−point trap. Depending on the requirements of an application, different levels of precisions can be selected:

    p   Program precision. This option is the default and means a trap handler can only identify which program caused a floating−point exception.

    f   Function precision. The trap handler can determine the function that caused a floating−point exception.

    i   Instruction precision. The trap handler can determine the exact instruction that caused a floating−point exception.

    Other Alpha compilers provide the equivalent options called **−scope_safe** and **−resumption_safe**.

−mieee−conformant

    This option marks the generated code as IEEE conformant. You must not use this option unless you also specify **−mtrap−precision=i** and either **−mfp−trap−mode=su** or **−mfp−trap−mode=sui**. Its only effect is to emit the line **.eflag 48** in the function prologue of the generated assembly file.

−mbuild−constants

    Normally GCC examines a 32− or 64−bit integer constant to see if it can construct it from smaller constants in two or three instructions. If it cannot, it outputs the constant as a literal and generates code to load it from the data segment at run time.

    Use this option to require GCC to construct *all* integer constants using code, even if it takes more instructions (the maximum is six).

    You typically use this option to build a shared library dynamic loader. Itself a shared library, it must relocate itself in memory before it can find the variables and constants in its own data segment.

−mbwx
−mno−bwx
−mcix
−mno−cix
−mfix
−mno−fix
−mmax
−mno−max

    Indicate whether GCC should generate code to use the optional BWX, CIX, FIX and MAX instruction sets. The default is to use the instruction sets supported by the CPU type specified via **−mcpu=** option or that of the CPU on which GCC was built if none is specified.

−msafe−bwa
−mno−safe−bwa

    Indicate whether in the absence of the optional BWX instruction set GCC should generate multi−thread and async−signal safe code for byte and aligned word memory accesses.

−msafe−partial
−mno−safe−partial

    Indicate whether GCC should generate multi−thread and async−signal safe code for partial memory accesses, including piecemeal accesses to unaligned data as well as block accesses to leading and trailing parts of aggregate types or other objects in memory that do not respectively start and end on an aligned 64−bit data boundary.

−mfloat−vax

−mfloat−ieee
    Generate code that uses (does not use) VAX F and G floating−point arithmetic instead of IEEE single
    and double precision.

−mexplicit−relocs
−mno−explicit−relocs
    Older Alpha assemblers provided no way to generate symbol relocations except via assembler macros.
    Use of these macros does not allow optimal instruction scheduling. GNU binutils as of version 2.12
    supports a new syntax that allows the compiler to explicitly mark which relocations should apply to
    which instructions. This option is mostly useful for debugging, as GCC detects the capabilities of the
    assembler when it is built and sets the default accordingly.

−msmall−data
−mlarge−data
    When **−mexplicit−relocs** is in effect, static data is accessed via *gp−relative* relocations. When
    −msmall−data is used, objects 8 bytes long or smaller are placed in a *small data area* (the `.sdata`
    and `.sbss` sections) and are accessed via 16−bit relocations off of the `$gp` register. This limits the
    size of the small data area to 64KB, but allows the variables to be directly accessed via a single
    instruction.

    The default is **−mlarge−data**. With this option the data area is limited to just below 2GB. Programs
    that require more than 2GB of data must use `malloc` or `mmap` to allocate the data in the heap instead
    of in the program's data segment.

    When generating code for shared libraries, **−fpic** implies **−msmall−data** and **−fPIC** implies
    −mlarge−data.

−msmall−text
−mlarge−text
    When **−msmall−text** is used, the compiler assumes that the code of the entire program (or shared
    library) fits in 4MB, and is thus reachable with a branch instruction. When **−msmall−data** is used, the
    compiler can assume that all local symbols share the same `$gp` value, and thus reduce the number of
    instructions required for a function call from 4 to 1.

    The default is **−mlarge−text**.

−mcpu=*cpu_type*
    Set the instruction set and instruction scheduling parameters for machine type *cpu_type*. You can
    specify either the **EV** style name or the corresponding chip number. GCC supports scheduling
    parameters for the EV4, EV5 and EV6 family of processors and chooses the default values for the
    instruction set from the processor you specify. If you do not specify a processor type, GCC defaults to
    the processor on which the compiler was built.

    Supported values for *cpu_type* are

    ev4
    ev45
    21064
        Schedules as an EV4 and has no instruction set extensions.

    ev5
    21164
        Schedules as an EV5 and has no instruction set extensions.

    ev56
    21164a
        Schedules as an EV5 and supports the BWX extension.

    pca56

21164pc
21164PC
        Schedules as an EV5 and supports the BWX and MAX extensions.

ev6
21264
        Schedules as an EV6 and supports the BWX, FIX, and MAX extensions.

ev67
21264a
        Schedules as an EV6 and supports the BWX, CIX, FIX, and MAX extensions.

Native toolchains also support the value **native**, which selects the best architecture option for the host processor. **−mcpu=native** has no effect if GCC does not recognize the processor.

−mtune=*cpu_type*
        Set only the instruction scheduling parameters for machine type *cpu_type*. The instruction set is not changed.

        Native toolchains also support the value **native**, which selects the best architecture option for the host processor. **−mtune=native** has no effect if GCC does not recognize the processor.

−mmemory−latency=*time*
        Sets the latency the scheduler should assume for typical memory references as seen by the application. This number is highly dependent on the memory access patterns used by the application and the size of the external cache on the machine.

        Valid options for *time* are

        number
                A decimal number representing clock cycles.

        L1
        L2
        L3
        main
                The compiler contains estimates of the number of clock cycles for "typical" EV4 & EV5 hardware for the Level 1, 2 & 3 caches (also called Dcache, Scache, and Bcache), as well as to main memory. Note that L3 is only valid for EV5.

eBPF Options
−mframe−limit=*bytes*
        This specifies the hard limit for frame sizes, in bytes. Currently, the value that can be specified should be less than or equal to **32767**. Defaults to whatever limit is imposed by the version of the Linux kernel targeted.

−mbig−endian
        Generate code for a big−endian target.

−mlittle−endian
        Generate code for a little−endian target. This is the default.

−mjmpext
−mno−jmpext
        Enable or disable generation of extra conditional−branch instructions. Enabled for CPU v2 and above.

−mjmp32
−mno−jmp32
        Enable or disable generation of 32−bit jump instructions. Enabled for CPU v3 and above.

−malu32

−mno−alu32

> Enable or disable generation of 32−bit ALU instructions. Enabled for CPU v3 and above.

−mv3−atomics
−mno−v3−atomics

> Enable or disable instructions for general atomic operations introduced in CPU v3. Enabled for CPU v3 and above.

−mbswap
−mno−bswap

> Enable or disable byte swap instructions. Enabled for CPU v4 and above.

−msdiv
−mno−sdiv

> Enable or disable signed division and modulus instructions. Enabled for CPU v4 and above.

−msmov
−mno−smov

> Enable or disable sign−extending move and memory load instructions. Enabled for CPU v4 and above.

−mcpu=_version_

> This specifies which version of the eBPF ISA to target. Newer versions may not be supported by all kernels. The default is **v4**.
>
> Supported values for _version_ are:
>
> **v1** The first stable eBPF ISA with no special features or extensions.
>
> **v2** Supports the jump extensions, as in **−mjmpext**.
>
> **v3** All features of v2, plus:
>
> > −<32−bit jump operations, as in **−mjmp32>**
> > −<32−bit ALU operations, as in **−malu32>**
> > −<general atomic operations, as in **−mv3−atomics>**
>
> **v4** All features of v3, plus:
>
> > −<Byte swap instructions, as in **−mbswap>**
> > −<Signed division and modulus instructions, as in **−msdiv>**
> > −<Sign−extending move and memory load instructions, as in **−msmov>**

−mco−re

> Enable BPF Compile Once − Run Everywhere (CO−RE) support. Requires and is implied by **−gbtf**.

−mno−co−re

> Disable BPF Compile Once − Run Everywhere (CO−RE) support. BPF CO−RE support is enabled by default when generating BTF debug information for the BPF target.

−mxbpf

> Generate code for an expanded version of BPF, which relaxes some of the restrictions imposed by the BPF architecture:
>
> −<Save and restore callee−saved registers at function entry and>
> > exit, respectively.

−masm=_dialect_

> Outputs assembly instructions using eBPF selected _dialect_. The default is **pseudoc**.
>
> Supported values for _dialect_ are:
>
> **normal**
> > Outputs normal assembly dialect.

pseudoc

Outputs pseudo−c assembly dialect.

−minline−memops−threshold=*bytes*

Specifies a size threshold in bytes at or below which memmove, memcpy and memset shall always be expanded inline. Operations dealing with sizes larger than this threshold would have to be implemented using a library call instead of being expanded inline, but since BPF doesn't allow libcalls, exceeding this threshold results in a compile−time error. The default is **1024** bytes.

FR30 Options

These options are defined specifically for the FR30 port.

−msmall−model

Use the small address space model. This can produce smaller code, but it does assume that all symbolic values and addresses fit into a 20−bit range.

−mno−lsim

Assume that runtime support has been provided and so there is no need to include the simulator library (*libsim.a*) on the linker command line.

FT32 Options

These options are defined specifically for the FT32 port.

−msim

Specifies that the program will be run on the simulator. This causes an alternate runtime startup and library to be linked. You must not use this option when generating programs that will run on real hardware; you must provide your own runtime library for whatever I/O functions are needed.

−mlra

Does nothing. Preserved for backward compatibility.

−mnodiv

Do not use div and mod instructions.

−mft32b

Enable use of the extended instructions of the FT32B processor.

−mcompress

Compress all code using the Ft32B code compression scheme.

−mnopm

Do not generate code that reads program memory.

FRV Options

−mgpr−32

Only use the first 32 general−purpose registers.

−mgpr−64

Use all 64 general−purpose registers.

−mfpr−32

Use only the first 32 floating−point registers.

−mfpr−64

Use all 64 floating−point registers.

−mhard−float

Use hardware instructions for floating−point operations.

−msoft−float

Use library routines for floating−point operations.

−malloc−cc

Dynamically allocate condition code registers.

−mfixed−cc

Do not try to dynamically allocate condition code registers, only use `icc0` and `fcc0`.

−mdword

Change ABI to use double word insns.

−mno−dword

Do not use double word instructions.

−mdouble

Use floating−point double instructions.

−mno−double

Do not use floating−point double instructions.

−mmedia

Use media instructions.

−mno−media

Do not use media instructions.

−mmuladd

Use multiply and add/subtract instructions.

−mno−muladd

Do not use multiply and add/subtract instructions.

−mfdpic

Select the FDPIC ABI, which uses function descriptors to represent pointers to functions. Without any PIC/PIE−related options, it implies **−fPIE**. With **−fpic** or **−fpie**, it assumes GOT entries and small data are within a 12−bit range from the GOT base address; with **−fPIC** or **−fPIE**, GOT offsets are computed with 32 bits. With a **bfin−elf** target, this option implies **−msim**.

−minline−plt

Enable inlining of PLT entries in function calls to functions that are not known to bind locally. It has no effect without **−mfdpic**. It's enabled by default if optimizing for speed and compiling for shared libraries (i.e., **−fPIC** or **−fpic**), or when an optimization option such as **−O3** or above is present in the command line.

−mTLS

Assume a large TLS segment when generating thread−local code.

−mtls

Do not assume a large TLS segment when generating thread−local code.

−mgprel−ro

Enable the use of `GPREL` relocations in the FDPIC ABI for data that is known to be in read−only sections. It's enabled by default, except for **−fpic** or **−fpie**: even though it may help make the global offset table smaller, it trades 1 instruction for 4. With **−fPIC** or **−fPIE**, it trades 3 instructions for 4, one of which may be shared by multiple symbols, and it avoids the need for a GOT entry for the referenced symbol, so it's more likely to be a win. If it is not, **−mno−gprel−ro** can be used to disable it.

−multilib−library−pic

Link with the (library, not FD) pic libraries. It's implied by **−mlibrary−pic**, as well as by **−fPIC** and **−fpic** without **−mfdpic**. You should never have to use it explicitly.

−mlinked−fp

Follow the EABI requirement of always creating a frame pointer whenever a stack frame is allocated. This option is enabled by default and can be disabled with **−mno−linked−fp**.

−mlong−calls

Use indirect addressing to call functions outside the current compilation unit. This allows the functions to be placed anywhere within the 32−bit address space.

−malign−labels
Try to align labels to an 8−byte boundary by inserting NOPs into the previous packet. This option only has an effect when VLIW packing is enabled. It doesn't create new packets; it merely adds NOPs to existing ones.

−mlibrary−pic
Generate position−independent EABI code.

−macc−4
Use only the first four media accumulator registers.

−macc−8
Use all eight media accumulator registers.

−mpack
Pack VLIW instructions.

−mno−pack
Do not pack VLIW instructions.

−mno−eflags
Do not mark ABI switches in e_flags.

−mcond−move
Enable the use of conditional−move instructions (default).

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mno−cond−move
Disable the use of conditional−move instructions.

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mscc
Enable the use of conditional set instructions (default).

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mno−scc
Disable the use of conditional set instructions.

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mcond−exec
Enable the use of conditional execution (default).

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mno−cond−exec
Disable the use of conditional execution.

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mvliw−branch
Run a pass to pack branches into VLIW instructions (default).

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mno−vliw−branch
Do not run a pass to pack branches into VLIW instructions.

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mmulti−cond−exec
Enable optimization of `&&` and `||` in conditional execution (default).

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mno−multi−cond−exec

Disable optimization of `&&` and `||` in conditional execution.

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mnested−cond−exec

Enable nested conditional execution optimizations (default).

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−mno−nested−cond−exec

Disable nested conditional execution optimizations.

This switch is mainly for debugging the compiler and will likely be removed in a future version.

−moptimize−membar

This switch removes redundant `membar` instructions from the compiler−generated code. It is enabled by default.

−mno−optimize−membar

This switch disables the automatic removal of redundant `membar` instructions from the generated code.

−mtomcat−stats

Cause gas to print out tomcat statistics.

−mcpu=*cpu*

Select the processor type for which to generate code. Possible values are **frv**, **fr550**, **tomcat**, **fr500**, **fr450**, **fr405**, **fr400**, **fr300** and **simple**.

GNU/Linux Options

These **−m** options are defined for GNU/Linux targets:

−mglibc

Use the GNU C library. This is the default except on ***−*−linux−*uclibc***, ***−*−linux−*musl*** and ***−*−linux−*android*** targets.

−muclibc

Use uClibc C library. This is the default on ***−*−linux−*uclibc*** targets.

−mmusl

Use the musl C library. This is the default on ***−*−linux−*musl*** targets.

−mbionic

Use Bionic C library. This is the default on ***−*−linux−*android*** targets.

−mandroid

Compile code compatible with Android platform. This is the default on ***−*−linux−*android*** targets.

When compiling, this option enables **−mbionic**, **−fPIC**, **−fno−exceptions** and **−fno−rtti** by default. When linking, this option makes the GCC driver pass Android−specific options to the linker. Finally, this option causes the preprocessor macro `__ANDROID__` to be defined.

−tno−android−cc

Disable compilation effects of **−mandroid**, i.e., do not enable **−mbionic**, **−fPIC**, **−fno−exceptions** and **−fno−rtti** by default.

−tno−android−ld

Disable linking effects of **−mandroid**, i.e., pass standard Linux linking options to the linker.

H8/300 Options

These **−m** options are defined for the H8/300 implementations:

−mrelax

Shorten some address references at link time, when possible; uses the linker option **−relax**.

−mh

Generate code for the H8/300H.

−ms

Generate code for the H8S.

−mn

Generate code for the H8S and H8/300H in the normal mode. This switch must be used either with **−mh** or **−ms**.

−ms2600

Generate code for the H8S/2600. This switch must be used with **−ms**.

−mexr

Extended registers are stored on stack before execution of function with monitor attribute. Default option is **−mexr**. This option is valid only for H8S targets.

−mno−exr

Extended registers are not stored on stack before execution of function with monitor attribute. Default option is **−mno−exr**. This option is valid only for H8S targets.

−mint32

Make `int` data 32 bits by default.

−malign−300

On the H8/300H and H8S, use the same alignment rules as for the H8/300. The default for the H8/300H and H8S is to align longs and floats on 4−byte boundaries. **−malign−300** causes them to be aligned on 2−byte boundaries. This option has no effect on the H8/300.

HPPA Options

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

−march=*architecture−type*

Generate code for the specified architecture. The choices for *architecture−type* are **1.0** for PA 1.0, **1.1** for PA 1.1, and **2.0** for PA 2.0 processors. Refer to */usr/lib/sched.models* on an HP−UX system to determine the proper architecture option for your machine. Code compiled for lower numbered architectures runs on higher numbered architectures, but not the other way around.

−mpa−risc−1−0
−mpa−risc−1−1
−mpa−risc−2−0

Synonyms for **−march=1.0**, **−march=1.1**, and **−march=2.0** respectively.

−matomic−libcalls

Generate libcalls for atomic loads and stores when sync libcalls are disabled. This option is enabled by default. It only affects the generation of atomic libcalls by the HPPA backend.

Both the sync and *libatomic* libcall implementations use locking. As a result, processor stores are not atomic with respect to other atomic operations. Processor loads up to DImode are atomic with respect to other atomic operations provided they are implemented as a single access.

The PA−RISC architecture does not support any atomic operations in hardware except for the `ldcw` instruction. Thus, all atomic support is implemented using sync and atomic libcalls. Sync libcall support is in *libgcc.a*. Atomic libcall support is in *libatomic*.

This option generates `__atomic_exchange` calls for atomic stores. It also provides special handling for atomic DImode accesses on 32−bit targets.

−mbig−switch

Does nothing. Preserved for backward compatibility.

−mcaller−copies

The caller copies function arguments passed by hidden reference. This option should be used with care as it is not compatible with the default 32−bit runtime. However, only aggregates larger than eight bytes are passed by hidden reference and the option provides better compatibility with OpenMP.

−mcoherent−ldcw

Use ldcw/ldcd coherent cache−control hint.

−mdisable−fpregs

Disable floating−point registers. Equivalent to `−msoft−float`.

−mdisable−indexing

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

−mfast−indirect−calls

Generate code that assumes calls never cross space boundaries. This allows GCC to emit code that performs faster indirect calls.

This option does not work in the presence of shared libraries or nested functions.

−mfixed−range=*register−range*

Generate code treating the given register range as fixed registers. A fixed register is one that the register allocator cannot use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

−mgas

Enable the use of assembler directives only GAS understands.

−mgnu−ld

Use options specific to GNU **ld**. This passes **−shared** to **ld** when building a shared library. It is the default when GCC is configured, explicitly or implicitly, with the GNU linker. This option does not affect which **ld** is called; it only changes what parameters are passed to that **ld**. The **ld** that is called is determined by the **−−with−ld** configure option, GCC's program search path, and finally by the user's **PATH**. The linker used by GCC can be printed using **which `gcc −print−prog−name=ld`**. This option is only available on the 64−bit HP−UX GCC, i.e. configured with **hppa*64*−*−hpux***.

−mhp−ld

Use options specific to HP **ld**. This passes **−b** to **ld** when building a shared library and passes **+Accept TypeMismatch** to **ld** on all links. It is the default when GCC is configured, explicitly or implicitly, with the HP linker. This option does not affect which **ld** is called; it only changes what parameters are passed to that **ld**. The **ld** that is called is determined by the **−−with−ld** configure option, GCC's program search path, and finally by the user's **PATH**. The linker used by GCC can be printed using **which `gcc −print−prog−name=ld`**. This option is only available on the 64−bit HP−UX GCC, i.e. configured with **hppa*64*−*−hpux***.

−mlinker−opt

Enable the optimization pass in the HP−UX linker. Note this makes symbolic debugging impossible.

−mlong−calls

Generate code that uses long call sequences. This ensures that a call is always able to reach linker generated stubs. The default is to generate long calls only when the distance from the call site to the beginning of the function or translation unit, as the case may be, exceeds a predefined limit set by the branch type being used. The limits for normal calls are 7,600,000 and 240,000 bytes, respectively for the PA 2.0 and PA 1.X architectures. Sibcalls are always limited at 240,000 bytes.

Distances are measured from the beginning of functions when using the **−ffunction−sections** option, or when using the **−mgas** and **−mno−portable−runtime** options together under HP−UX with the SOM linker.

It is normally not desirable to use this option as it degrades performance. However, it may be useful in large applications, particularly when partial linking is used to build the application.

The types of long calls used depends on the capabilities of the assembler and linker, and the type of code being generated. The impact on systems that support long absolute calls, and long pic symbol−difference or pc−relative calls should be relatively small. However, an indirect call is used on 32−bit ELF systems in pic code and it is quite long.

−mlong−load−store
Generate 3−instruction load and store sequences as sometimes required by the HP−UX 10 linker. This is equivalent to the **+k** option to the HP compilers.

−mjump−in−delay
This option is ignored and provided for compatibility purposes only.

−mno−space−regs
Generate code that assumes the target has no space registers. This allows GCC to generate faster indirect calls and use unscaled index address modes.

Such code is suitable for level 0 PA systems and kernels.

−mordered
Assume memory references are ordered and barriers are not needed.

−mportable−runtime
Use the portable calling conventions proposed by HP for ELF systems.

−mschedule=*cpu−type*
Schedule code according to the constraints for the machine type *cpu−type*. The choices for *cpu−type* are **700 7100**, **7100LC**, **7200**, **7300** and **8000**. Refer to */usr/lib/sched.models* on an HP−UX system to determine the proper scheduling option for your machine. The default scheduling is **8000**.

−msio
Generate the predefine, `_SIO`, for server IO. The default is **−mwsio**. This generates the predefines, `__hp9000s700`, `__hp9000s700__` and `_WSIO`, for workstation IO. These options are available under HP−UX and HI−UX.

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

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

−msoft−mult
Use software integer multiplication.

This disables the use of the `xmpyu` instruction.

−munix=*unix−std*
Generate compiler predefines and select a startfile for the specified UNIX standard. The choices for *unix−std* are **93**, **95** and **98**. **93** is supported on all HP−UX versions. **95** is available on HP−UX 10.10 and later. **98** is available on HP−UX 11.11 and later. The default values are **93** for HP−UX 10.00, **95** for HP−UX 10.10 though to 11.00, and **98** for HP−UX 11.11 and later.

−munix=93 provides the same predefines as GCC 3.3 and 3.4. **−munix=95** provides additional predefines for `XOPEN_UNIX` and `_XOPEN_SOURCE_EXTENDED`, and the startfile *unix95.o*. **−munix=98** provides additional predefines for `_XOPEN_UNIX`, `_XOPEN_SOURCE_EXTENDED`, `_INCLUDE__STDC_A1_SOURCE` and `_INCLUDE_XOPEN_SOURCE_500`, and the startfile *unix98.o*.

It is *important* to note that this option changes the interfaces for various library routines. It also affects the operational behavior of the C library. Thus, *extreme* care is needed in using this option.

Library code that is intended to operate with more than one UNIX standard must test, set and restore the variable `__xpg4_extended_mask` as appropriate. Most GNU software doesn't provide this capability.

−nolibdld

Suppress the generation of link options to search libdld.sl when the **−static** option is specified on HP−UX 10 and later.

−static

The HP−UX implementation of setlocale in libc has a dependency on libdld.sl. There isn't an archive version of libdld.sl. Thus, when the **−static** option is specified, special link options are needed to resolve this dependency.

On HP−UX 10 and later, the GCC driver adds the necessary options to link with libdld.sl when the **−static** option is specified. This causes the resulting binary to be dynamic. On the 64−bit port, the linkers generate dynamic binaries by default in any case. The **−nolibdld** option can be used to prevent the GCC driver from adding these link options.

−threads

Add support for multithreading with the *dce thread* library under HP−UX. This option sets flags for both the preprocessor and linker.

IA−64 Options

These are the **−m** options defined for the Intel IA−64 architecture.

−mbig−endian

Generate code for a big−endian target. This is the default for HP−UX.

−mlittle−endian

Generate code for a little−endian target. This is the default for AIX5 and GNU/Linux.

−mgnu−as
−mno−gnu−as

Generate (or don't) code for the GNU assembler. This is the default.

−mgnu−ld
−mno−gnu−ld

Generate (or don't) code for the GNU linker. This is the default.

−mno−pic

Generate code that does not use a global pointer register. The result is not position independent code, and violates the IA−64 ABI.

−mvolatile−asm−stop
−mno−volatile−asm−stop

Generate (or don't) a stop bit immediately before and after volatile asm statements.

−mregister−names
−mno−register−names

Generate (or don't) **in**, **loc**, and **out** register names for the stacked registers. This may make assembler output more readable.

−mno−sdata
−msdata

Disable (or enable) optimizations that use the small data section. This may be useful for working around optimizer bugs.

−mconstant−gp

Generate code that uses a single constant global pointer value. This is useful when compiling kernel code.

−mauto−pic

> Generate code that is self−relocatable. This implies **−mconstant−gp**. This is useful when compiling firmware code.

−minline−float−divide−min−latency

> Generate code for inline divides of floating−point values using the minimum latency algorithm.

−minline−float−divide−max−throughput

> Generate code for inline divides of floating−point values using the maximum throughput algorithm.

−mno−inline−float−divide

> Do not generate inline code for divides of floating−point values.

−minline−int−divide−min−latency

> Generate code for inline divides of integer values using the minimum latency algorithm.

−minline−int−divide−max−throughput

> Generate code for inline divides of integer values using the maximum throughput algorithm.

−mno−inline−int−divide

> Do not generate inline code for divides of integer values.

−minline−sqrt−min−latency

> Generate code for inline square roots using the minimum latency algorithm.

−minline−sqrt−max−throughput

> Generate code for inline square roots using the maximum throughput algorithm.

−mno−inline−sqrt

> Do not generate inline code for `sqrt`.

−mfused−madd
−mno−fused−madd

> Do (don't) generate code that uses the fused multiply/add or multiply/subtract instructions. The default is to use these instructions.

−mno−dwarf2−asm
−mdwarf2−asm

> Don't (or do) generate assembler code for the DWARF line number debugging info. This may be useful when not using the GNU assembler.

−mearly−stop−bits
−mno−early−stop−bits

> Allow stop bits to be placed earlier than immediately preceding the instruction that triggered the stop bit. This can improve instruction scheduling, but does not always do so.

−mfixed−range=*register−range*

> Generate code treating the given register range as fixed registers. A fixed register is one that the register allocator cannot use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

−mtls−size=*tls−size*

> Specify bit size of immediate TLS offsets. Valid values are 14, 22, and 64.

−mtune=*cpu−type*

> Tune the instruction scheduling for a particular CPU, Valid values are **itanium**, **itanium1**, **merced**, **itanium2**, and **mckinley**.

−milp32
−mlp64

> Generate code for a 32−bit or 64−bit environment. The 32−bit environment sets int, long and pointer to 32 bits. The 64−bit environment sets int to 32 bits and long and pointer to 64 bits. These are HP−UX specific flags.

−mno−sched−br−data−spec
−msched−br−data−spec
> (Dis/En)able data speculative scheduling before reload. This results in generation of `ld.a` instructions and the corresponding check instructions (`ld.c` / `chk.a`). The default setting is disabled.

−msched−ar−data−spec
−mno−sched−ar−data−spec
> (En/Dis)able data speculative scheduling after reload. This results in generation of `ld.a` instructions and the corresponding check instructions (`ld.c` / `chk.a`). The default setting is enabled.

−mno−sched−control−spec
−msched−control−spec
> (Dis/En)able control speculative scheduling. This feature is available only during region scheduling (i.e. before reload). This results in generation of the `ld.s` instructions and the corresponding check instructions `chk.s`. The default setting is disabled.

−msched−br−in−data−spec
−mno−sched−br−in−data−spec
> (En/Dis)able speculative scheduling of the instructions that are dependent on the data speculative loads before reload. This is effective only with **−msched−br−data−spec** enabled. The default setting is enabled.

−msched−ar−in−data−spec
−mno−sched−ar−in−data−spec
> (En/Dis)able speculative scheduling of the instructions that are dependent on the data speculative loads after reload. This is effective only with **−msched−ar−data−spec** enabled. The default setting is enabled.

−msched−in−control−spec
−mno−sched−in−control−spec
> (En/Dis)able speculative scheduling of the instructions that are dependent on the control speculative loads. This is effective only with **−msched−control−spec** enabled. The default setting is enabled.

−mno−sched−prefer−non−data−spec−insns
−msched−prefer−non−data−spec−insns
> If enabled, data−speculative instructions are chosen for schedule only if there are no other choices at the moment. This makes the use of the data speculation much more conservative. The default setting is disabled.

−mno−sched−prefer−non−control−spec−insns
−msched−prefer−non−control−spec−insns
> If enabled, control−speculative instructions are chosen for schedule only if there are no other choices at the moment. This makes the use of the control speculation much more conservative. The default setting is disabled.

−mno−sched−count−spec−in−critical−path
−msched−count−spec−in−critical−path
> If enabled, speculative dependencies are considered during computation of the instructions priorities. This makes the use of the speculation a bit more conservative. The default setting is disabled.

−msched−spec−ldc
> Use a simple data speculation check. This option is on by default.

−msched−control−spec−ldc
> Use a simple check for control speculation. This option is on by default.

−msched−stop−bits−after−every−cycle
> Place a stop bit after every cycle when scheduling. This option is on by default.

−msched−fp−mem−deps−zero−cost
 Assume that floating−point stores and loads are not likely to cause a conflict when placed into the same instruction group. This option is disabled by default.

−msel−sched−dont−check−control−spec
 Generate checks for control speculation in selective scheduling. This flag is disabled by default.

−msched−max−memory−insns=_max−insns_
 Limit on the number of memory insns per instruction group, giving lower priority to subsequent memory insns attempting to schedule in the same instruction group. Frequently useful to prevent cache bank conflicts. The default value is 1.

−msched−max−memory−insns−hard−limit
 Makes the limit specified by **msched−max−memory−insns** a hard limit, disallowing more than that number in an instruction group. Otherwise, the limit is "soft", meaning that non−memory operations are preferred when the limit is reached, but memory operations may still be scheduled.

LM32 Options
 These **−m** options are defined for the LatticeMico32 architecture:

−mbarrel−shift−enabled
 Enable barrel−shift instructions.

−mdivide−enabled
 Enable divide and modulus instructions.

−mmultiply−enabled
 Enable multiply instructions.

−msign−extend−enabled
 Enable sign extend instructions.

−muser−enabled
 Enable user−defined instructions.

LoongArch Options
 These command−line options are defined for LoongArch targets:

−march=_arch−type_
 Generate instructions for the machine type _arch−type_. **−march=**_arch−type_ allows GCC to generate code that may not run at all on processors other than the one indicated.

 The choices for _arch−type_ are:

native
 Local processor type detected by the native compiler.

loongarch64
 Generic LoongArch 64−bit processor.

la464
 LoongArch LA464−based processor with LSX, LASX.

la664
 LoongArch LA664−based processor with LSX, LASX and all LoongArch v1.1 instructions.

la64v1.0
 LoongArch64 ISA version 1.0.

la64v1.1
 LoongArch64 ISA version 1.1.

 More information about LoongArch ISA versions can be found at **<https://github.com/loongson/la−toolchain−conventions>**.

−mtune=*tune−type*

Optimize the generated code for the given processor target.

The choices for *tune−type* are:

native

Local processor type detected by the native compiler.

generic

Generic LoongArch processor.

loongarch64

Generic LoongArch 64−bit processor.

la464

LoongArch LA464 core.

la664

LoongArch LA664 core.

−mabi=*base−abi−type*

Generate code for the specified calling convention. *base−abi−type* can be one of:

lp64d

Uses 64−bit general purpose registers and 32/64−bit floating−point registers for parameter passing. Data model is LP64, where **int** is 32 bits, while **long int** and pointers are 64 bits.

lp64f

Uses 64−bit general purpose registers and 32−bit floating−point registers for parameter passing. Data model is LP64, where **int** is 32 bits, while **long int** and pointers are 64 bits.

lp64s

Uses 64−bit general purpose registers and no floating−point registers for parameter passing. Data model is LP64, where **int** is 32 bits, while **long int** and pointers are 64 bits.

−mfpu=*fpu−type*

Generate code for the specified FPU type, which can be one of:

64   Allow the use of hardware floating−point instructions for 32−bit and 64−bit operations.

32   Allow the use of hardware floating−point instructions for 32−bit operations.

none

0   Prevent the use of hardware floating−point instructions.

−msimd=*simd−type*

Enable generation of LoongArch SIMD instructions for vectorization and via builtin functions. The value can be one of:

lasx

Enable generating instructions from the 256−bit LoongArch Advanced SIMD Extension (LASX) and the 128−bit LoongArch SIMD Extension (LSX).

lsx   Enable generating instructions from the 128−bit LoongArch SIMD Extension (LSX).

none

No LoongArch SIMD instruction may be generated.

−msoft−float

Force **−mfpu=none** and prevents the use of floating−point registers for parameter passing. This option may change the target ABI.

−msingle−float

Force **−mfpu=32** and allow the use of 32−bit floating−point registers for parameter passing. This option may change the target ABI.

−mdouble−float

> Force **−mfpu=64** and allow the use of 32/64−bit floating−point registers for parameter passing. This option may change the target ABI.

−mlasx
−mno−lasx
−mlsx
−mno−lsx

> Incrementally adjust the scope of the SIMD extensions (none / LSX / LASX) that can be used by the compiler for code generation. Enabling LASX with **mlasx** automatically enables LSX, and diabling LSX with **mno−lsx** automatically disables LASX. These driver−only options act upon the final **msimd** configuration state and make incremental changes in the order they appear on the GCC driver's command line, deriving the final / canonicalized **msimd** option that is passed to the compiler proper.

−mbranch−cost=*n*

> Set the cost of branches to roughly *n* instructions.

−maddr−reg−reg−cost=*n*

> Set the cost of ADDRESS_REG_REG to the value calculated by *n*.

−mcheck−zero−division
−mno−check−zero−divison

> Trap (do not trap) on integer division by zero. The default is **−mcheck−zero−division** for **−O0** or **−Og**, and **−mno−check−zero−division** for other optimization levels.

−mcond−move−int
−mno−cond−move−int

> Conditional moves for integral data in general−purpose registers are enabled (disabled). The default is **−mcond−move−int**.

−mcond−move−float
−mno−cond−move−float

> Conditional moves for floating−point registers are enabled (disabled). The default is **−mcond−move−float**.

−mmemcpy
−mno−memcpy

> Force (do not force) the use of memcpy for non−trivial block moves. The default is **−mno−memcpy**, which allows GCC to inline most constant−sized copies. Setting optimization level to **−Os** also forces the use of memcpy, but **−mno−memcpy** may override this behavior if explicitly specified, regardless of the order these options on the command line.

−mstrict−align
−mno−strict−align

> Avoid or allow generating memory accesses that may not be aligned on a natural object boundary as described in the architecture specification. The default is **−mno−strict−align**.

−G *num*

> Put global and static data smaller than *num* bytes into a small data section. The default value is 0.

−mmax−inline−memcpy−size=*n*

> Inline all block moves (such as calls to memcpy or structure copies) less than or equal to *n* bytes. The default value of *n* is 1024.

−mcmodel=*code−model*

> Set the code model to one of:

> **tiny−static (Not implemented yet)**
> **tiny (Not implemented yet)**
> **normal**
>> The text segment must be within 128MB addressing space. The data segment must be within 2GB addressing space.

medium

   The text segment and data segment must be within 2GB addressing space.

large (Not implemented yet)
extreme

   This mode does not limit the size of the code segment and data segment. The **−mcmodel=extreme** option is incompatible with **−fplt** and/or **−mexplicit−relocs=none**.

The default code model is normal.

−mexplicit−relocs=_style_

   Set when to use assembler relocation operators when dealing with symbolic addresses. The alternative is to use assembler macros instead, which may limit instruction scheduling but allow linker relaxation. with **−mexplicit−relocs=none** the assembler macros are always used, with **−mexplicit−relocs=always** the assembler relocation operators are always used, with **−mexplicit−relocs=auto** the compiler will use the relocation operators where the linker relaxation is impossible to improve the code quality, and macros elsewhere. The default value for the option is determined with the assembler capability detected during GCC build−time and the setting of **−mrelax**: **−mexplicit−relocs=none** if the assembler does not support relocation operators at all, **−mexplicit−relocs=always** if the assembler supports relocation operators but **−mrelax** is not enabled, **−mexplicit−relocs=auto** if the assembler supports relocation operators and **−mrelax** is enabled.

−mexplicit−relocs

   An alias of **−mexplicit−relocs=always** for backward compatibility.

−mno−explicit−relocs

   An alias of **−mexplicit−relocs=none** for backward compatibility.

−mdirect−extern−access
−mno−direct−extern−access

   Do not use or use GOT to access external symbols. The default is **−mno−direct−extern−access**: GOT is used for external symbols with default visibility, but not used for other external symbols.

   With **−mdirect−extern−access**, GOT is not used and all external symbols are PC−relatively addressed. It is **only** suitable for environments where no dynamic link is performed, like firmwares, OS kernels, executables linked with **−static** or **−static−pie**. **−mdirect−extern−access** is not compatible with **−fPIC** or **−fpic**.

−mrelax
−mno−relax

   Take (do not take) advantage of linker relaxations. If **−mpass−mrelax−to−as** is enabled, this option is also passed to the assembler. The default is determined during GCC build−time by detecting corresponding assembler support: **−mrelax** if the assembler supports both the **−mrelax** option and the conditional branch relaxation (it's required or the .align directives and conditional branch instructions in the assembly code outputted by GCC may be rejected by the assembler because of a relocation overflow), **−mno−relax** otherwise.

−mpass−mrelax−to−as
−mno−pass−mrelax−to−as

   Pass (do not pass) the **−mrelax** or **−mno−relax** option to the assembler. The default is determined during GCC build−time by detecting corresponding assembler support: **−mpass−mrelax−to−as** if the assembler supports the **−mrelax** option, **−mno−pass−mrelax−to−as** otherwise. This option is mostly useful for debugging, or interoperation with assemblers different from the build−time one.

−mrecip

   This option enables use of the reciprocal estimate and reciprocal square root estimate instructions with additional Newton−Raphson steps to increase precision instead of doing a divide or square root and divide for floating−point arguments. These instructions are generated only when **−funsafe−math−optimizations** is enabled together with **−ffinite−math−only** and **−fno−trapping−math**. This option is off by default. Before you can use this option, you must sure

the target CPU supports frecipe and frsqrte instructions. Note that while the throughput of the sequence is higher than the throughput of the non−reciprocal instruction, the precision of the sequence can be decreased by up to 2 ulp (i.e. the inverse of 1.0 equals 0.99999994).

−mrecip=*opt*

This option controls which reciprocal estimate instructions may be used. *opt* is a comma−separated list of options, which may be preceded by a **!** to invert the option:

all   Enable all estimate instructions.

default
Enable the default instructions, equivalent to **−mrecip**.

none
Disable all estimate instructions, equivalent to **−mno−recip**.

div   Enable the approximation for scalar division.

vec−div
Enable the approximation for vectorized division.

sqrt
Enable the approximation for scalar square root.

vec−sqrt
Enable the approximation for vectorized square root.

rsqrt
Enable the approximation for scalar reciprocal square root.

vec−rsqrt
Enable the approximation for vectorized reciprocal square root.

So, for example, **−mrecip=all,!sqrt** enables all of the reciprocal approximations, except for scalar square root.

−mfrecipe
−mno−frecipe
Use (do not use) frecipe.{s/d} and frsqrte.{s/d} instructions. When build with **−march=la664**, it is enabled by default. The default is **−mno−frecipe**.

−mdiv32
−mno−div32
Use (do not use) div.w[u] and mod.w[u] instructions with input not sign−extended. When build with **−march=la664**, it is enabled by default. The default is **−mno−div32**.

−mlam−bh
−mno−lam−bh
Use (do not use) am{swap/add}[_db].{b/h} instructions. When build with **−march=la664**, it is enabled by default. The default is **−mno−lam−bh**.

−mlamcas
−mno−lamcas
Use (do not use) amcas[_db].{b/h/w/d} instructions. When build with **−march=la664**, it is enabled by default. The default is **−mno−lamcas**.

−mld−seq−sa
−mno−ld−seq−sa
Whether a same−address load−load barrier (dbar 0x700) is needed. When build with **−march=la664**, it is enabled by default. The default is **−mno−ld−seq−sa**, the load−load barrier is needed.

−mtls−dialect=*opt*
This option controls which tls dialect may be used for general dynamic and local dynamic TLS models.

trad
> Use traditional TLS. This is the default.

desc
> Use TLS descriptors.

−mannotate−tablejump
−mno−annotate−tablejump
> Create an annotation section `.discard.tablejump_annotate` to correlate the `jirl` instruction and the jump table when a jump table is used to optimize the `switch` statement. Some external tools, for example *objtool* of the Linux kernel building system, need the annotation to analysis the control flow. The default is **−mno−annotate−tablejump**.

−−param loongarch−vect−unroll−limit=_n_
> The vectorizer will use available tuning information to determine whether it would be beneficial to unroll the main vectorized loop and by how much. This parameter set's the upper bound of how much the vectorizer will unroll the main loop. The default value is six.

M32C Options
−mcpu=_name_
> Select the CPU for which code is generated. *name* may be one of **r8c** for the R8C/Tiny series, **m16c** for the M16C (up to /60) series, **m32cm** for the M16C/80 series, or **m32c** for the M32C/80 series.

−msim
> Specifies that the program will be run on the simulator. This causes an alternate runtime library to be linked in which supports, for example, file I/O. You must not use this option when generating programs that will run on real hardware; you must provide your own runtime library for whatever I/O functions are needed.

−memregs=_number_
> Specifies the number of memory−based pseudo−registers GCC uses during code generation. These pseudo−registers are used like real registers, so there is a tradeoff between GCC's ability to fit the code into available registers, and the performance penalty of using memory instead of registers. Note that all modules in a program must be compiled with the same value for this option. Because of that, you must not use this option with GCC's default runtime libraries.

M32R/D Options
These **−m** options are defined for Renesas M32R/D architectures:

−m32r2
> Generate code for the M32R/2.

−m32rx
> Generate code for the M32R/X.

−m32r
> Generate code for the M32R. This is the default.

−mmodel=small
> Assume all objects live in the lower 16MB of memory (so that their addresses can be loaded with the `ld24` instruction), and assume all subroutines are reachable with the `bl` instruction. This is the default.

> The addressability of a particular object can be set with the `model` attribute.

−mmodel=medium
> Assume objects may be anywhere in the 32−bit address space (the compiler generates `seth/add3` instructions to load their addresses), and assume all subroutines are reachable with the `bl` instruction.

−mmodel=large
> Assume objects may be anywhere in the 32−bit address space (the compiler generates `seth/add3` instructions to load their addresses), and assume subroutines may not be reachable with the `bl` instruction (the compiler generates the much slower `seth/add3/jl` instruction sequence).

−msdata=none

> Disable use of the small data area. Variables are put into one of `.data`, `.bss`, or `.rodata` (unless the `section` attribute has been specified). This is the default.
>
> The small data area consists of sections `.sdata` and `.sbss`. Objects may be explicitly put in the small data area with the `section` attribute using one of these sections.

−msdata=sdata

> Put small global and static data in the small data area, but do not generate special code to reference them.

−msdata=use

> Put small global and static data in the small data area, and generate special instructions to reference them.

−G *num*

> Put global and static objects less than or equal to *num* bytes into the small data or BSS sections instead of the normal data or BSS sections. The default value of *num* is 8. The **−msdata** option must be set to one of **sdata** or **use** for this option to have any effect.
>
> All modules should be compiled with the same **−G** *num* value. Compiling with different values of *num* may or may not work; if it doesn't the linker gives an error message−−−incorrect code is not generated.

−mdebug

> Makes the M32R−specific code in the compiler display some statistics that might help in debugging programs.

−malign−loops

> Align all loops to a 32−byte boundary.

−mno−align−loops

> Do not enforce a 32−byte alignment for loops. This is the default.

−missue−rate=*number*

> Issue *number* instructions per cycle. *number* can only be 1 or 2.

−mbranch−cost=*number*

> *number* can only be 1 or 2. If it is 1 then branches are preferred over conditional code, if it is 2, then the opposite applies.

−mflush−trap=*number*

> Specifies the trap number to use to flush the cache. The default is 12. Valid numbers are between 0 and 15 inclusive.

−mno−flush−trap

> Specifies that the cache cannot be flushed by using a trap.

−mflush−func=*name*

> Specifies the name of the operating system function to call to flush the cache. The default is **_flush_cache**, but a function call is only used if a trap is not available.

−mno−flush−func

> Indicates that there is no OS function for flushing the cache.

M680x0 Options

These are the **−m** options defined for M680x0 and ColdFire processors. The default settings depend on which architecture was selected when the compiler was configured; the defaults for the most common choices are given below.

−march=*arch*

> Generate code for a specific M680x0 or ColdFire instruction set architecture. Permissible values of *arch* for M680x0 architectures are: **68000**, **68010**, **68020**, **68030**, **68040**, **68060** and **cpu32**. ColdFire architectures are selected according to Freescale's ISA classification and the permissible values are:

isaa, **isaaplus**, **isab** and **isac**.

GCC defines a macro `__mcf`*arch*`__` whenever it is generating code for a ColdFire target. The *arch* in this macro is one of the **−march** arguments given above.

When used together, **−march** and **−mtune** select code that runs on a family of similar processors but that is optimized for a particular microarchitecture.

−mcpu=*cpu*

Generate code for a specific M680x0 or ColdFire processor. The M680x0 *cpu*s are: **68000**, **68010**, **68020**, **68030**, **68040**, **68060**, **68302**, **68332** and **cpu32**. The ColdFire *cpu*s are given by the table below, which also classifies the CPUs into families:

Family : **−mcpu** *arguments*
51 : **51 51ac 51ag 51cn 51em 51je 51jf 51jg 51jm 51mm 51qe 51qm**
5206 : **5202 5204 5206**
5206e : **5206e**
5208 : **5207 5208**
5211a : **5210a 5211a**
5213 : **5211 5212 5213**
5216 : **5214 5216**
52235 : **52230 52231 52232 52233 52234 52235**
5225 : **5224 5225**
52259 : **52252 52254 52255 52256 52258 52259**
5235 : **5232 5233 5234 5235 523x**
5249 : **5249**
5250 : **5250**
5271 : **5270 5271**
5272 : **5272**
5275 : **5274 5275**
5282 : **5280 5281 5282 528x**
53017 : **53011 53012 53013 53014 53015 53016 53017**
5307 : **5307**
5329 : **5327 5328 5329 532x**
5373 : **5372 5373 537x**
5407 : **5407**
5475 : **5470 5471 5472 5473 5474 5475 547x 5480 5481 5482 5483 5484 5485**

−mcpu=*cpu* overrides **−march=***arch* if *arch* is compatible with *cpu*. Other combinations of **−mcpu** and **−march** are rejected.

GCC defines the macro `__mcf_cpu_`*cpu* when ColdFire target *cpu* is selected. It also defines `__mcf_family_`*family*, where the value of *family* is given by the table above.

−mtune=*tune*

Tune the code for a particular microarchitecture within the constraints set by **−march** and **−mcpu**. The M680x0 microarchitectures are: **68000**, **68010**, **68020**, **68030**, **68040**, **68060** and **cpu32**. The ColdFire microarchitectures are: **cfv1**, **cfv2**, **cfv3**, **cfv4** and **cfv4e**.

You can also use **−mtune=68020−40** for code that needs to run relatively well on 68020, 68030 and 68040 targets. **−mtune=68020−60** is similar but includes 68060 targets as well. These two options select the same tuning decisions as **−m68020−40** and **−m68020−60** respectively.

GCC defines the macros `__mc`*arch* and `__mc`*arch*`__` when tuning for 680x0 architecture *arch*. It also defines `mc`*arch* unless either **−ansi** or a non−GNU **−std** option is used. If GCC is tuning for a range of architectures, as selected by **−mtune=68020−40** or **−mtune=68020−60**, it defines the macros for every architecture in the range.

GCC also defines the macro `__mu`*arch*`__` when tuning for ColdFire microarchitecture *uarch*, where *uarch* is one of the arguments given above.

–m68000
–mc68000

>   Generate output for a 68000. This is the default when the compiler is configured for 68000–based systems. It is equivalent to **–march=68000**.

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

–m68010

>   Generate output for a 68010. This is the default when the compiler is configured for 68010–based systems. It is equivalent to **–march=68010**.

–m68020
–mc68020

>   Generate output for a 68020. This is the default when the compiler is configured for 68020–based systems. It is equivalent to **–march=68020**.

–m68030

>   Generate output for a 68030. This is the default when the compiler is configured for 68030–based systems. It is equivalent to **–march=68030**.

–m68040

>   Generate output for a 68040. This is the default when the compiler is configured for 68040–based systems. It is equivalent to **–march=68040**.

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

–m68060

>   Generate output for a 68060. This is the default when the compiler is configured for 68060–based systems. It is equivalent to **–march=68060**.

>   This option inhibits the use of 68020 and 68881/68882 instructions that have to be emulated by software on the 68060. Use this option if your 68060 does not have code to emulate those instructions.

–mcpu32

>   Generate output for a CPU32. This is the default when the compiler is configured for CPU32–based systems. It is equivalent to **–march=cpu32**.

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

–m5200

>   Generate output for a 520X ColdFire CPU. This is the default when the compiler is configured for 520X–based systems. It is equivalent to **–mcpu=5206**, and is now deprecated in favor of that option.

>   Use this option for microcontroller with a 5200 core, including the MCF5202, MCF5203, MCF5204 and MCF5206.

–m5206e

>   Generate output for a 5206e ColdFire CPU. The option is now deprecated in favor of the equivalent **–mcpu=5206e**.

–m528x

>   Generate output for a member of the ColdFire 528X family. The option is now deprecated in favor of the equivalent **–mcpu=528x**.

–m5307

>   Generate output for a ColdFire 5307 CPU. The option is now deprecated in favor of the equivalent **–mcpu=5307**.

−m5407

Generate output for a ColdFire 5407 CPU. The option is now deprecated in favor of the equivalent **−mcpu=5407**.

−mcfv4e

Generate output for a ColdFire V4e family CPU (e.g. 547x/548x). This includes use of hardware floating−point instructions. The option is equivalent to **−mcpu=547x**, and is now deprecated in favor of that option.

−m68020−40

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

The option is equivalent to **−march=68020 −mtune=68020−40**.

−m68020−60

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

The option is equivalent to **−march=68020 −mtune=68020−60**.

−mhard−float
−m68881

Generate floating−point instructions. This is the default for 68020 and above, and for ColdFire devices that have an FPU. It defines the macro `__HAVE_68881__` on M680x0 targets and `__mcffpu__` on ColdFire targets.

−msoft−float

Do not generate floating−point instructions; use library calls instead. This is the default for 68000, 68010, and 68832 targets. It is also the default for ColdFire devices that have no FPU.

−mdiv
−mno−div

Generate (do not generate) ColdFire hardware divide and remainder instructions. If **−march** is used without **−mcpu**, the default is "on" for ColdFire architectures and "off" for M680x0 architectures. Otherwise, the default is taken from the target CPU (either the default CPU, or the one specified by **−mcpu**). For example, the default is "off" for **−mcpu=5206** and "on" for **−mcpu=5206e**.

GCC defines the macro `__mcfhwdiv__` when this option is enabled.

−mshort

Consider type `int` to be 16 bits wide, like `short int`. Additionally, parameters passed on the stack are also aligned to a 16−bit boundary even on targets whose API mandates promotion to 32−bit.

−mno−short

Do not consider type `int` to be 16 bits wide. This is the default.

−mnobitfield
−mno−bitfield

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

−mbitfield

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

−mrtd

Use a different function−calling convention, in which functions that take a fixed number of arguments return with the `rtd` instruction, which pops their arguments while returning. This saves one instruction in the caller since there is no need to pop the arguments there.

This calling convention is incompatible with the one normally used on Unix, so you cannot use it if you need to call libraries compiled with the Unix compiler.

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

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

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

The default is **−mno−rtd**.

−malign−int
−mno−align−int

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

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

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

−mno−strict−align
−mstrict−align

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

−msep−data

Generate code that allows the data segment to be located in a different area of memory from the text segment. This allows for execute−in−place in an environment without virtual memory management. This option implies **−fPIC**.

−mno−sep−data

Generate code that assumes that the data segment follows the text segment. This is the default.

−mid−shared−library

Generate code that supports shared libraries via the library ID method. This allows for execute−in−place and shared libraries in an environment without virtual memory management. This option implies **−fPIC**.

−mno−id−shared−library

Generate code that doesn't assume ID−based shared libraries are being used. This is the default.

−mshared−library−id=n

Specifies the identification number of the ID−based shared library being compiled. Specifying a value of 0 generates more compact code; specifying other values forces the allocation of that number to the current library, but is no more space− or time−efficient than omitting this option.

−mxgot
−mno−xgot

When generating position−independent code for ColdFire, generate code that works if the GOT has more than 8192 entries. This code is larger and slower than code generated without this option. On M680x0 processors, this option is not needed; **−fPIC** suffices.

GCC normally uses a single instruction to load values from the GOT. While this is relatively efficient, it only works if the GOT is smaller than about 64k. Anything larger causes the linker to report an error such as:

```
relocation truncated to fit: R_68K_GOT16O foobar
```

If this happens, you should recompile your code with **−mxgot**. It should then work with very large GOTs. However, code generated with **−mxgot** is less efficient, since it takes 4 instructions to fetch the value of a global symbol.

Note that some linkers, including newer versions of the GNU linker, can create multiple GOTs and sort GOT entries. If you have such a linker, you should only need to use **−mxgot** when compiling a single object file that accesses more than 8192 GOT entries. Very few do.

These options have no effect unless GCC is generating position−independent code.

−mlong−jump−table−offsets
Use 32−bit offsets in `switch` tables. The default is to use 16−bit offsets.

MCore Options
These are the **−m** options defined for the Motorola M*Core processors.

−mhardlit
−mno−hardlit
Inline constants into the code stream if it can be done in two instructions or less.

−mdiv
−mno−div
Use the divide instruction. (Enabled by default).

−mrelax−immediate
−mno−relax−immediate
Allow arbitrary−sized immediates in bit operations.

−mwide−bitfields
−mno−wide−bitfields
Always treat bit−fields as `int`−sized.

−m4byte−functions
−mno−4byte−functions
Force all functions to be aligned to a 4−byte boundary.

−mcallgraph−data
−mno−callgraph−data
Emit callgraph information.

−mslow−bytes
−mno−slow−bytes
Prefer word access when reading byte quantities.

−mlittle−endian
−mbig−endian
Generate code for a little−endian target.

−m210
−m340
Generate code for the 210 processor.

−mno−lsim
Assume that runtime support has been provided and so omit the simulator library (*libsim.a)* from the linker command line.

−mstack−increment=*size*
Set the maximum amount for a single stack increment operation. Large values can increase the speed of programs that contain functions that need a large amount of stack space, but they can also trigger a segmentation fault if the stack is extended too much. The default value is 0x1000.

MicroBlaze Options

−msoft−float

Use software emulation for floating point (default).

−mhard−float

Use hardware floating−point instructions.

−mmemcpy

Do not optimize block moves, use memcpy.

−mno−clearbss

This option is deprecated. Use **−fno−zero−initialized−in−bss** instead.

−mcpu=*cpu−type*

Use features of, and schedule code for, the given CPU. Supported values are in the format **v***X***.***YY***.***Z*,
where *X* is a major version, *YY* is the minor version, and *Z* is compatibility code. Example values are
v3.00.a, **v4.00.b**, **v5.00.a**, **v5.00.b**, **v6.00.a**.

−mxl−soft−mul

Use software multiply emulation (default).

−mxl−soft−div

Use software emulation for divides (default).

−mxl−barrel−shift

Use the hardware barrel shifter.

−mxl−pattern−compare

Use pattern compare instructions.

−msmall−divides

Use table lookup optimization for small signed integer divisions.

−mxl−stack−check

This option is deprecated. Use **−fstack−check** instead.

−mxl−gp−opt

Use GP−relative .sdata/.sbss sections.

−mxl−multiply−high

Use multiply high instructions for high part of 32x32 multiply.

−mxl−float−convert

Use hardware floating−point conversion instructions.

−mxl−float−sqrt

Use hardware floating−point square root instruction.

−mbig−endian

Generate code for a big−endian target.

−mlittle−endian

Generate code for a little−endian target.

−mxl−reorder

Use reorder instructions (swap and byte reversed load/store).

−mxl−mode−*app−model*

Select application model *app−model*. Valid models are

executable

normal executable (default), uses startup code *crt0.o*.

xmdstub

for use with Xilinx Microprocessor Debugger (XMD) based software intrusive debug agent called
xmdstub. This uses startup file *crt1.o* and sets the start address of the program to 0x800.

bootstrap

for applications that are loaded using a bootloader. This model uses startup file *crt2.o* which does not contain a processor reset vector handler. This is suitable for transferring control on a processor reset to the bootloader rather than the application.

novectors

for applications that do not require any of the MicroBlaze vectors. This option may be useful for applications running within a monitoring application. This model uses *crt3.o* as a startup file.

Option **−xl−mode−***app−model* is a deprecated alias for **−mxl−mode−***app−model*.

−mpic−data−is−text−relative

Assume that the displacement between the text and data segments is fixed at static link time. This allows data to be referenced by offset from start of text address instead of GOT since PC−relative addressing is not supported.

MIPS Options

−EB

Generate big−endian code.

−EL

Generate little−endian code. This is the default for **mips*el−*−*** configurations.

−march=*arch*

Generate code that runs on *arch*, which can be the name of a generic MIPS ISA, or the name of a particular processor. The ISA names are: **mips1**, **mips2**, **mips3**, **mips4**, **mips32**, **mips32r2**, **mips32r3**, **mips32r5**, **mips32r6**, **mips64**, **mips64r2**, **mips64r3**, **mips64r5** and **mips64r6**. The processor names are: **4kc**, **4km**, **4kp**, **4ksc**, **4kec**, **4kem**, **4kep**, **4ksd**, **5kc**, **5kf**, **20kc**, **24kc**, **24kf2_1**, **24kf1_1**, **24kec**, **24kef2_1**, **24kef1_1**, **34kc**, **34kf2_1**, **34kf1_1**, **34kn**, **74kc**, **74kf2_1**, **74kf1_1**, **74kf3_2**, **1004kc**, **1004kf2_1**, **1004kf1_1**, **i6400**, **i6500**, **interaptiv**, **loongson2e**, **loongson2f**, **loongson3a**, **gs464**, **gs464e**, **gs264e**, **m4k**, **m14k**, **m14kc**, **m14ke**, **m14kec**, **m5100**, **m5101**, **octeon**, **octeon+**, **octeon2**, **octeon3**, **orion**, **p5600**, **p6600**, **r2000**, **r3000**, **r3900**, **r4000**, **r4400**, **r4600**, **r4650**, **r4700**, **r5900**, **r6000**, **r8000**, **rm7000**, **rm9000**, **r10000**, **r12000**, **r14000**, **r16000**, **sb1**, **sr71000**, **vr4100**, **vr4111**, **vr4120**, **vr4130**, **vr4300**, **vr5000**, **vr5400**, **vr5500**, **xlr** and **xlp**. The special value **from−abi** selects the most compatible architecture for the selected ABI (that is, **mips1** for 32−bit ABIs and **mips3** for 64−bit ABIs).

The native Linux/GNU toolchain also supports the value **native**, which selects the best architecture option for the host processor. **−march=native** has no effect if GCC does not recognize the processor.

In processor names, a final **000** can be abbreviated as **k** (for example, **−march=r2k**). Prefixes are optional, and **vr** may be written **r**.

Names of the form *n***f2_1** refer to processors with FPUs clocked at half the rate of the core, names of the form *n***f1_1** refer to processors with FPUs clocked at the same rate as the core, and names of the form *n***f3_2** refer to processors with FPUs clocked a ratio of 3:2 with respect to the core. For compatibility reasons, *n***f** is accepted as a synonym for *n***f2_1** while *n***x** and *b***fx** are accepted as synonyms for *n***f1_1**.

GCC defines two macros based on the value of this option. The first is _MIPS_ARCH, which gives the name of target architecture, as a string. The second has the form _MIPS_ARCH_*foo*, where *foo* is the capitalized value of _MIPS_ARCH. For example, **−march=r2000** sets _MIPS_ARCH to "r2000" and defines the macro _MIPS_ARCH_R2000.

Note that the _MIPS_ARCH macro uses the processor names given above. In other words, it has the full prefix and does not abbreviate **000** as **k**. In the case of **from−abi**, the macro names the resolved architecture (either "mips1" or "mips3"). It names the default architecture when no **−march** option is given.

−mtune=_arch_
>   Optimize for _arch_. Among other things, this option controls the way instructions are scheduled, and the perceived cost of arithmetic operations. The list of _arch_ values is the same as for **−march**.
>
>   When this option is not used, GCC optimizes for the processor specified by **−march**. By using **−march** and **−mtune** together, it is possible to generate code that runs on a family of processors, but optimize the code for one particular member of that family.
>
>   **−mtune** defines the macros _MIPS_TUNE and _MIPS_TUNE_`foo`, which work in the same way as the **−march** ones described above.

−mips1
>   Equivalent to **−march=mips1**.

−mips2
>   Equivalent to **−march=mips2**.

−mips3
>   Equivalent to **−march=mips3**.

−mips4
>   Equivalent to **−march=mips4**.

−mips32
>   Equivalent to **−march=mips32**.

−mips32r3
>   Equivalent to **−march=mips32r3**.

−mips32r5
>   Equivalent to **−march=mips32r5**.

−mips32r6
>   Equivalent to **−march=mips32r6**.

−mips64
>   Equivalent to **−march=mips64**.

−mips64r2
>   Equivalent to **−march=mips64r2**.

−mips64r3
>   Equivalent to **−march=mips64r3**.

−mips64r5
>   Equivalent to **−march=mips64r5**.

−mips64r6
>   Equivalent to **−march=mips64r6**.

−mips16
−mno−mips16
>   Generate (do not generate) MIPS16 code. If GCC is targeting a MIPS32 or MIPS64 architecture, it makes use of the MIPS16e ASE.
>
>   MIPS16 code generation can also be controlled on a per−function basis by means of `mips16` and `nomips16` attributes.

−mmips16e2
−mno−mips16e2
>   Use (do not use) the MIPS16e2 ASE. This option modifies the behavior of the **−mips16** option such that it targets the MIPS16e2 ASE.

−mflip−mips16
>   Generate MIPS16 code on alternating functions. This option is provided for regression testing of mixed MIPS16/non−MIPS16 code generation, and is not intended for ordinary use in compiling user

code.

−minterlink−compressed
−mno−interlink−compressed
>   Require (do not require) that code using the standard (uncompressed) MIPS ISA be link−compatible with MIPS16 and microMIPS code, and vice versa.
>
>   For example, code using the standard ISA encoding cannot jump directly to MIPS16 or microMIPS code; it must either use a call or an indirect jump. **−minterlink−compressed** therefore disables direct jumps unless GCC knows that the target of the jump is not compressed.

−minterlink−mips16
−mno−interlink−mips16
>   Aliases of **−minterlink−compressed** and **−mno−interlink−compressed**. These options predate the microMIPS ASE and are retained for backwards compatibility.

−mabi=32
−mabi=o64
−mabi=n32
−mabi=64
−mabi=eabi
>   Generate code for the given ABI.
>
>   Note that the EABI has a 32−bit and a 64−bit variant. GCC normally generates 64−bit code when you select a 64−bit architecture, but you can use **−mgp32** to get 32−bit code instead.
>
>   For information about the O64 ABI, see **<https://gcc.gnu.org/projects/mipso64−abi.html>**.
>
>   GCC supports a variant of the o32 ABI in which floating−point registers are 64 rather than 32 bits wide. You can select this combination with **−mabi=32 −mfp64**. This ABI relies on the mthc1 and mfhc1 instructions and is therefore only supported for MIPS32R2, MIPS32R3 and MIPS32R5 processors.
>
>   The register assignments for arguments and return values remain the same, but each scalar value is passed in a single 64−bit register rather than a pair of 32−bit registers. For example, scalar floating−point values are returned in **$f0** only, not a **$f0/$f1** pair. The set of call−saved registers also remains the same in that the even−numbered double−precision registers are saved.
>
>   Two additional variants of the o32 ABI are supported to enable a transition from 32−bit to 64−bit registers. These are FPXX (**−mfpxx**) and FP64A (**−mfp64 −mno−odd−spreg**). The FPXX extension mandates that all code must execute correctly when run using 32−bit or 64−bit registers. The code can be interlinked with either FP32 or FP64, but not both. The FP64A extension is similar to the FP64 extension but forbids the use of odd−numbered single−precision registers. This can be used in conjunction with the FRE mode of FPUs in MIPS32R5 processors and allows both FP32 and FP64A code to interlink and run in the same process without changing FPU modes.

−mabicalls
−mno−abicalls
>   Generate (do not generate) code that is suitable for SVR4−style dynamic objects. **−mabicalls** is the default for SVR4−based systems.

−mshared
−mno−shared
>   Generate (do not generate) code that is fully position−independent, and that can therefore be linked into shared libraries. This option only affects **−mabicalls**.
>
>   All **−mabicalls** code has traditionally been position−independent, regardless of options like **−fPIC** and **−fpic**. However, as an extension, the GNU toolchain allows executables to use absolute accesses for locally−binding symbols. It can also use shorter GP initialization sequences and generate direct calls to locally−defined functions. This mode is selected by **−mno−shared**.

−mno−shared depends on binutils 2.16 or higher and generates objects that can only be linked by the GNU linker. However, the option does not affect the ABI of the final executable; it only affects the ABI of relocatable objects. Using **−mno−shared** generally makes executables both smaller and quicker.

−mshared is the default.

−mplt
−mno−plt

Assume (do not assume) that the static and dynamic linkers support PLTs and copy relocations. This option only affects **−mno−shared −mabicalls**. For the n64 ABI, this option has no effect without **−msym32**.

You can make **−mplt** the default by configuring GCC with **−−with−mips−plt**. The default is **−mno−plt** otherwise.

−mxgot
−mno−xgot

Lift (do not lift) the usual restrictions on the size of the global offset table.

GCC normally uses a single instruction to load values from the GOT. While this is relatively efficient, it only works if the GOT is smaller than about 64k. Anything larger causes the linker to report an error such as:

```
relocation truncated to fit: R_MIPS_GOT16 foobar
```

If this happens, you should recompile your code with **−mxgot**. This works with very large GOTs, although the code is also less efficient, since it takes three instructions to fetch the value of a global symbol.

Note that some linkers can create multiple GOTs. If you have such a linker, you should only need to use **−mxgot** when a single object file accesses more than 64k's worth of GOT entries. Very few do.

These options have no effect unless GCC is generating position independent code.

−mgp32

Assume that general−purpose registers are 32 bits wide.

−mgp64

Assume that general−purpose registers are 64 bits wide.

−mfp32

Assume that floating−point registers are 32 bits wide.

−mfp64

Assume that floating−point registers are 64 bits wide.

−mfpxx

Do not assume the width of floating−point registers.

−mhard−float

Use floating−point coprocessor instructions.

−msoft−float

Do not use floating−point coprocessor instructions. Implement floating−point calculations using library calls instead.

−mno−float

Equivalent to **−msoft−float**, but additionally asserts that the program being compiled does not perform any floating−point operations. This option is presently supported only by some bare−metal MIPS configurations, where it may select a special set of libraries that lack all floating−point support (including, for example, the floating−point `printf` formats). If code compiled with **−mno−float** accidentally contains floating−point operations, it is likely to suffer a link−time or run−time failure.

−msingle−float
>   Assume that the floating−point coprocessor only supports single−precision operations.

−mdouble−float
>   Assume that the floating−point coprocessor supports double−precision operations. This is the default.

−modd−spreg
−mno−odd−spreg
>   Enable the use of odd−numbered single−precision floating−point registers for the o32 ABI. This is the default for processors that are known to support these registers. When using the o32 FPXX ABI, **−mno−odd−spreg** is set by default.

−mabs=2008
−mabs=legacy
>   These options control the treatment of the special not−a−number (NaN) IEEE 754 floating−point data with the abs.*fmt* and neg.*fmt* machine instructions.

>   By default or when **−mabs=legacy** is used the legacy treatment is selected. In this case these instructions are considered arithmetic and avoided where correct operation is required and the input operand might be a NaN. A longer sequence of instructions that manipulate the sign bit of floating−point datum manually is used instead unless the **−ffinite−math−only** option has also been specified.

>   The **−mabs=2008** option selects the IEEE 754−2008 treatment. In this case these instructions are considered non−arithmetic and therefore operating correctly in all cases, including in particular where the input operand is a NaN. These instructions are therefore always used for the respective operations.

−mnan=2008
−mnan=legacy
>   These options control the encoding of the special not−a−number (NaN) IEEE 754 floating−point data.

>   The **−mnan=legacy** option selects the legacy encoding. In this case quiet NaNs (qNaNs) are denoted by the first bit of their trailing significand field being 0, whereas signaling NaNs (sNaNs) are denoted by the first bit of their trailing significand field being 1.

>   The **−mnan=2008** option selects the IEEE 754−2008 encoding. In this case qNaNs are denoted by the first bit of their trailing significand field being 1, whereas sNaNs are denoted by the first bit of their trailing significand field being 0.

>   The default is **−mnan=legacy** unless GCC has been configured with **−−with−nan=2008**.

−mllsc
−mno−llsc
>   Use (do not use) **ll**, **sc**, and **sync** instructions to implement atomic memory built−in functions. When neither option is specified, GCC uses the instructions if the target architecture supports them.

>   **−mllsc** is useful if the runtime environment can emulate the instructions and **−mno−llsc** can be useful when compiling for nonstandard ISAs. You can make either option the default by configuring GCC with **−−with−llsc** and **−−without−llsc** respectively. **−−with−llsc** is the default for some configurations; see the installation documentation for details.

−mdsp
−mno−dsp
>   Use (do not use) revision 1 of the MIPS DSP ASE.
>   This option defines the preprocessor macro `__mips_dsp`. It also defines `__mips_dsp_rev` to 1.

−mdspr2
−mno−dspr2
>   Use (do not use) revision 2 of the MIPS DSP ASE.
>   This option defines the preprocessor macros `__mips_dsp` and `__mips_dspr2`. It also defines `__mips_dsp_rev` to 2.

−msmartmips
−mno−smartmips
> Use (do not use) the MIPS SmartMIPS ASE.

−mpaired−single
−mno−paired−single
> Use (do not use) paired−single floating−point instructions.
> This option requires hardware floating−point support to be enabled.

−mdmx
−mno−mdmx
> Use (do not use) MIPS Digital Media Extension instructions. This option can only be used when generating 64−bit code and requires hardware floating−point support to be enabled.

−mips3d
−mno−mips3d
> Use (do not use) the MIPS−3D ASE. The option **−mips3d** implies **−mpaired−single**.

−mmicromips
−mno−micromips
> Generate (do not generate) microMIPS code.
>
> MicroMIPS code generation can also be controlled on a per−function basis by means of `micromips` and `nomicromips` attributes.

−mmt
−mno−mt
> Use (do not use) MT Multithreading instructions.

−mmcu
−mno−mcu
> Use (do not use) the MIPS MCU ASE instructions.

−meva
−mno−eva
> Use (do not use) the MIPS Enhanced Virtual Addressing instructions.

−mvirt
−mno−virt
> Use (do not use) the MIPS Virtualization (VZ) instructions.

−mxpa
−mno−xpa
> Use (do not use) the MIPS eXtended Physical Address (XPA) instructions.

−mcrc
−mno−crc
> Use (do not use) the MIPS Cyclic Redundancy Check (CRC) instructions.

−mginv
−mno−ginv
> Use (do not use) the MIPS Global INValidate (GINV) instructions.

−mloongson−mmi
−mno−loongson−mmi
> Use (do not use) the MIPS Loongson MultiMedia extensions Instructions (MMI).

−mloongson−ext
−mno−loongson−ext
> Use (do not use) the MIPS Loongson EXTensions (EXT) instructions.

−mloongson−ext2
−mno−loongson−ext2
> Use (do not use) the MIPS Loongson EXTensions r2 (EXT2) instructions.

−mlong64
> Force `long` types to be 64 bits wide. See **−mlong32** for an explanation of the default and the way that the pointer size is determined.

−mlong32
> Force `long`, `int`, and pointer types to be 32 bits wide.

> The default size of `ints`, `longs` and pointers depends on the ABI. All the supported ABIs use 32−bit `ints`. The n64 ABI uses 64−bit `longs`, as does the 64−bit EABI; the others use 32−bit `longs`. Pointers are the same size as `longs`, or the same size as integer registers, whichever is smaller.

−msym32
−mno−sym32
> Assume (do not assume) that all symbols have 32−bit values, regardless of the selected ABI. This option is useful in combination with **−mabi=64** and **−mno−abicalls** because it allows GCC to generate shorter and faster references to symbolic addresses.

−G *num*
> Put definitions of externally−visible data in a small data section if that data is no bigger than *num* bytes. GCC can then generate more efficient accesses to the data; see **−mgpopt** for details.

> The default **−G** option depends on the configuration.

−mlocal−sdata
−mno−local−sdata
> Extend (do not extend) the **−G** behavior to local data too, such as to static variables in C. **−mlocal−sdata** is the default for all configurations.

> If the linker complains that an application is using too much small data, you might want to try rebuilding the less performance−critical parts with **−mno−local−sdata**. You might also want to build large libraries with **−mno−local−sdata**, so that the libraries leave more room for the main program.

−mextern−sdata
−mno−extern−sdata
> Assume (do not assume) that externally−defined data is in a small data section if the size of that data is within the **−G** limit. **−mextern−sdata** is the default for all configurations.

> If you compile a module *Mod* with **−mextern−sdata −G** *num* **−mgpopt**, and *Mod* references a variable *Var* that is no bigger than *num* bytes, you must make sure that *Var* is placed in a small data section. If *Var* is defined by another module, you must either compile that module with a high−enough **−G** setting or attach a `section` attribute to *Var*'s definition. If *Var* is common, you must link the application with a high−enough **−G** setting.

> The easiest way of satisfying these restrictions is to compile and link every module with the same **−G** option. However, you may wish to build a library that supports several different small data limits. You can do this by compiling the library with the highest supported **−G** setting and additionally using **−mno−extern−sdata** to stop the library from making assumptions about externally−defined data.

−mgpopt
−mno−gpopt
> Use (do not use) GP−relative accesses for symbols that are known to be in a small data section; see **−G**, **−mlocal−sdata** and **−mextern−sdata**. **−mgpopt** is the default for all configurations.

> **−mno−gpopt** is useful for cases where the `$gp` register might not hold the value of `_gp`. For example, if the code is part of a library that might be used in a boot monitor, programs that call boot monitor routines pass an unknown value in `$gp`. (In such situations, the boot monitor itself is usually compiled with **−G0**.)

−mno−gpopt implies **−mno−local−sdata** and **−mno−extern−sdata**.

−membedded−data

−mno−embedded−data

> Allocate variables to the read−only data section first if possible, then next in the small data section if possible, otherwise in data. This gives slightly slower code than the default, but reduces the amount of RAM required when executing, and thus may be preferred for some embedded systems.

−muninit−const−in−rodata

−mno−uninit−const−in−rodata

> Put uninitialized const variables in the read−only data section. This option is only meaningful in conjunction with **−membedded−data**.

−mcode−readable=*setting*

> Specify whether GCC may generate code that reads from executable sections. There are three possible settings:

> **−mcode−readable=yes**

> > Instructions may freely access executable sections. This is the default setting.

> **−mcode−readable=pcrel**

> > MIPS16 PC−relative load instructions can access executable sections, but other instructions must not do so. This option is useful on 4KSc and 4KSd processors when the code TLBs have the Read Inhibit bit set. It is also useful on processors that can be configured to have a dual instruction/data SRAM interface and that, like the M4K, automatically redirect PC−relative loads to the instruction RAM.

> **−mcode−readable=no**

> > Instructions must not access executable sections. This option can be useful on targets that are configured to have a dual instruction/data SRAM interface but that (unlike the M4K) do not automatically redirect PC−relative loads to the instruction RAM.

−msplit−addresses

−mno−split−addresses

> Enable (disable) use of the %hi() and %lo() assembler relocation operators. This option has been superseded by **−mexplicit−relocs** but is retained for backwards compatibility.

−mexplicit−relocs=none

−mexplicit−relocs=base

−mexplicit−relocs=pcrel

−mexplicit−relocs

−mno−explicit−relocs

> These options control whether explicit relocs (such as %gp_rel) are used. The default value depends on the version of GAS when GCC itself was built.

> The base explicit−relocs support was introduced into GAS in 2001. The pcrel explicit−relocs support was introduced into GAS in 2014, which supports %pcrel_hi and %pcrel_lo.

−mcheck−zero−division

−mno−check−zero−division

> Trap (do not trap) on integer division by zero.

> The default is **−mcheck−zero−division**.

−mdivide−traps

−mdivide−breaks

> MIPS systems check for division by zero by generating either a conditional trap or a break instruction. Using traps results in smaller code, but is only supported on MIPS II and later. Also, some versions of the Linux kernel have a bug that prevents trap from generating the proper signal (SIGFPE). Use **−mdivide−traps** to allow conditional traps on architectures that support them and **−mdivide−breaks** to force the use of breaks.

The default is usually **−mdivide−traps**, but this can be overridden at configure time using **−−with−divide=breaks**. Divide−by−zero checks can be completely disabled using **−mno−check−zero−division**.

−mload−store−pairs
−mno−load−store−pairs
> Enable (disable) an optimization that pairs consecutive load or store instructions to enable load/store bonding. This option is enabled by default but only takes effect when the selected architecture is known to support bonding.

−mstrict−align
−mno−strict−align
−munaligned−access
−mno−unaligned−access
> Disable (enable) direct unaligned access for MIPS Release 6. MIPSr6 requires load/store unaligned−access support, by hardware or trap&emulate. So **−mstrict−align** may be needed by kernel. The options **−munaligned−access** and **−mno−unaligned−access** are obsoleted, and only for backward−compatible.

−mmemcpy
−mno−memcpy
> Force (do not force) the use of memcpy for non−trivial block moves. The default is **−mno−memcpy**, which allows GCC to inline most constant−sized copies.

−mlong−calls
−mno−long−calls
> Disable (do not disable) use of the jal instruction. Calling functions using jal is more efficient but requires the caller and callee to be in the same 256 megabyte segment.

> This option has no effect on abicalls code. The default is **−mno−long−calls**.

−mmad
−mno−mad
> Enable (disable) use of the mad, madu and mul instructions, as provided by the R4650 ISA.

−mimadd
−mno−imadd
> Enable (disable) use of the madd and msub integer instructions. The default is **−mimadd** on architectures that support madd and msub except for the 74k architecture where it was found to generate slower code.

−mfused−madd
−mno−fused−madd
> Enable (disable) use of the floating−point multiply−accumulate instructions, when they are available. The default is **−mfused−madd**.

> On the R8000 CPU when multiply−accumulate instructions are used, the intermediate product is calculated to infinite precision and is not subject to the FCSR Flush to Zero bit. This may be undesirable in some circumstances. On other processors the result is numerically identical to the equivalent computation using separate multiply, add, subtract and negate instructions.

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

−mfix−24k
−mno−fix−24k
> Work around the 24K E48 (lost data on stores during refill) errata. The workarounds are implemented by the assembler rather than by GCC.

−mfix−r4000

−mno−fix−r4000

Work around certain R4000 CPU errata:

−   A double−word or a variable shift may give an incorrect result if executed immediately after starting an integer division.

−   A double−word or a variable shift may give an incorrect result if executed while an integer multiplication is in progress.

−   An integer division may give an incorrect result if started in a delay slot of a taken branch or a jump.

−mfix−r4400

−mno−fix−r4400

Work around certain R4400 CPU errata:

−   A double−word or a variable shift may give an incorrect result if executed immediately after starting an integer division.

−mfix−r10000

−mno−fix−r10000

Work around certain R10000 errata:

−   `ll`/`sc` sequences may not behave atomically on revisions prior to 3.0. They may deadlock on revisions 2.6 and earlier.

This option can only be used if the target architecture supports branch−likely instructions. **−mfix−r10000** is the default when **−march=r10000** is used; **−mno−fix−r10000** is the default otherwise.

−mfix−r5900

−mno−fix−r5900

Do not attempt to schedule the preceding instruction into the delay slot of a branch instruction placed at the end of a short loop of six instructions or fewer and always schedule a `nop` instruction there instead. The short loop bug under certain conditions causes loops to execute only once or twice, due to a hardware bug in the R5900 chip. The workaround is implemented by the assembler rather than by GCC.

−mfix−rm7000

−mno−fix−rm7000

Work around the RM7000 `dmult`/`dmultu` errata. The workarounds are implemented by the assembler rather than by GCC.

−mfix−vr4120

−mno−fix−vr4120

Work around certain VR4120 errata:

−   `dmultu` does not always produce the correct result.

−   `div` and `ddiv` do not always produce the correct result if one of the operands is negative.

The workarounds for the division errata rely on special functions in *libgcc.a*. At present, these functions are only provided by the `mips64vr*-elf` configurations.

Other VR4120 errata require a NOP to be inserted between certain pairs of instructions. These errata are handled by the assembler, not by GCC itself.

−mfix−vr4130

Work around the VR4130 `mflo`/`mfhi` errata. The workarounds are implemented by the assembler rather than by GCC, although GCC avoids using `mflo` and `mfhi` if the VR4130 `macc`, `macchi`, `dmacc` and `dmacchi` instructions are available instead.

−mfix−sb1
−mno−fix−sb1
> Work around certain SB−1 CPU core errata. (This flag currently works around the SB−1 revision 2 "F1" and "F2" floating−point errata.)

−mr10k−cache−barrier=_setting_
> Specify whether GCC should insert cache barriers to avoid the side effects of speculation on R10K processors.
>
> In common with many processors, the R10K tries to predict the outcome of a conditional branch and speculatively executes instructions from the "taken" branch. It later aborts these instructions if the predicted outcome is wrong. However, on the R10K, even aborted instructions can have side effects.
>
> This problem only affects kernel stores and, depending on the system, kernel loads. As an example, a speculatively−executed store may load the target memory into cache and mark the cache line as dirty, even if the store itself is later aborted. If a DMA operation writes to the same area of memory before the "dirty" line is flushed, the cached data overwrites the DMA−ed data. See the R10K processor manual for a full description, including other potential problems.
>
> One workaround is to insert cache barrier instructions before every memory access that might be speculatively executed and that might have side effects even if aborted. **−mr10k−cache−barrier=**_setting_ controls GCC's implementation of this workaround. It assumes that aborted accesses to any byte in the following regions does not have side effects:
>
> 1.   the memory occupied by the current function's stack frame;
>
> 2.   the memory occupied by an incoming stack argument;
>
> 3.   the memory occupied by an object with a link−time−constant address.
>
> It is the kernel's responsibility to ensure that speculative accesses to these regions are indeed safe.
>
> If the input program contains a function declaration such as:
>
>   void foo (void);
>
> then the implementation of `foo` must allow `j foo` and `jal foo` to be executed speculatively. GCC honors this restriction for functions it compiles itself. It expects non−GCC functions (such as hand−written assembly code) to do the same.
>
> The option has three forms:
>
> **−mr10k−cache−barrier=load−store**
> > Insert a cache barrier before a load or store that might be speculatively executed and that might have side effects even if aborted.
>
> **−mr10k−cache−barrier=store**
> > Insert a cache barrier before a store that might be speculatively executed and that might have side effects even if aborted.
>
> **−mr10k−cache−barrier=none**
> > Disable the insertion of cache barriers. This is the default setting.

−mflush−func=_func_
−mno−flush−func
> Specifies the function to call to flush the I and D caches, or to not call any such function. If called, the function must take the same arguments as the common `_flush_func`, that is, the address of the memory range for which the cache is being flushed, the size of the memory range, and the number 3 (to flush both caches). The default depends on the target GCC was configured for, but commonly is either `_flush_func` or `__cpu_flush`.

−mbranch−cost=_num_
> Set the cost of branches to roughly _num_ "simple" instructions. This cost is only a heuristic and is not guaranteed to produce consistent results across releases. A zero cost redundantly selects the default,

which is based on the **−mtune** setting.

−mbranch−likely
−mno−branch−likely
>   Enable or disable use of Branch Likely instructions, regardless of the default for the selected architecture. By default, Branch Likely instructions may be generated if they are supported by the selected architecture. An exception is for the MIPS32 and MIPS64 architectures and processors that implement those architectures; for those, Branch Likely instructions are not be generated by default because the MIPS32 and MIPS64 architectures specifically deprecate their use.

−mcompact−branches=never
−mcompact−branches=optimal
−mcompact−branches=always
>   These options control which form of branches will be generated. The default is **−mcompact−branches=optimal**.
>
>   The **−mcompact−branches=never** option ensures that compact branch instructions will never be generated.
>
>   The **−mcompact−branches=always** option ensures that a compact branch instruction will be generated if available for MIPS Release 6 onwards. If a compact branch instruction is not available (or pre−R6), a delay slot form of the branch will be used instead.
>
>   If it is used for MIPS16/microMIPS targets, it will be just ignored now. The behavior for MIPS16/microMIPS may change in future, since they do have some compact branch instructions.
>
>   The **−mcompact−branches=optimal** option will cause a delay slot branch to be used if one is available in the current ISA and the delay slot is successfully filled. If the delay slot is not filled, a compact branch will be chosen if one is available.

−mfp−exceptions
−mno−fp−exceptions
>   Specifies whether FP exceptions are enabled. This affects how FP instructions are scheduled for some processors. The default is that FP exceptions are enabled.
>
>   For instance, on the SB−1, if FP exceptions are disabled, and we are emitting 64−bit code, then we can use both FP pipes. Otherwise, we can only use one FP pipe.

−mvr4130−align
−mno−vr4130−align
>   The VR4130 pipeline is two−way superscalar, but can only issue two instructions together if the first one is 8−byte aligned. When this option is enabled, GCC aligns pairs of instructions that it thinks should execute in parallel.
>
>   This option only has an effect when optimizing for the VR4130. It normally makes code faster, but at the expense of making it bigger. It is enabled by default at optimization level **−O3**.

−msynci
−mno−synci
>   Enable (disable) generation of `synci` instructions on architectures that support it. The `synci` instructions (if enabled) are generated when `__builtin___clear_cache` is compiled.
>
>   This option defaults to **−mno−synci**, but the default can be overridden by configuring GCC with **−−with−synci**.
>
>   When compiling code for single processor systems, it is generally safe to use `synci`. However, on many multi−core (SMP) systems, it does not invalidate the instruction caches on all cores and may lead to undefined behavior.

−mrelax−pic−calls

−mno−relax−pic−calls

Try to turn PIC calls that are normally dispatched via register $25 into direct calls. This is only possible if the linker can resolve the destination at link time and if the destination is within range for a direct call.

−mrelax−pic−calls is the default if GCC was configured to use an assembler and a linker that support the .reloc assembly directive and **−mexplicit−relocs** is in effect. With **−mno−explicit−relocs**, this optimization can be performed by the assembler and the linker alone without help from the compiler.

−mmcount−ra−address
−mno−mcount−ra−address

Emit (do not emit) code that allows _mcount to modify the calling function's return address. When enabled, this option extends the usual _mcount interface with a new *ra−address* parameter, which has type intptr_t * and is passed in register $12. _mcount can then modify the return address by doing both of the following:

*   Returning the new address in register $31.

*   Storing the new address in *ra−address*, if *ra−address* is nonnull.

The default is **−mno−mcount−ra−address**.

−mframe−header−opt
−mno−frame−header−opt

Enable (disable) frame header optimization in the o32 ABI. When using the o32 ABI, calling functions will allocate 16 bytes on the stack for the called function to write out register arguments. When enabled, this optimization will suppress the allocation of the frame header if it can be determined that it is unused.

This optimization is off by default at all optimization levels.

−mlxc1−sxc1
−mno−lxc1−sxc1

When applicable, enable (disable) the generation of lwxc1, swxc1, ldxc1, sdxc1 instructions. Enabled by default.

−mmadd4
−mno−madd4

When applicable, enable (disable) the generation of 4−operand madd.s, madd.d and related instructions. Enabled by default.

MMIX Options

These options are defined for the MMIX:

−mlibfuncs
−mno−libfuncs

Specify that intrinsic library functions are being compiled, passing all values in registers, no matter the size.

−mepsilon
−mno−epsilon

Generate floating−point comparison instructions that compare with respect to the rE epsilon register.

−mabi=mmixware
−mabi=gnu

Generate code that passes function parameters and return values that (in the called function) are seen as registers $0 and up, as opposed to the GNU ABI which uses global registers $231 and up.

−mzero−extend
−mno−zero−extend

When reading data from memory in sizes shorter than 64 bits, use (do not use) zero−extending load instructions by default, rather than sign−extending ones.

−mknuthdiv
−mno−knuthdiv
Make the result of a division yielding a remainder have the same sign as the divisor. With the default, **−mno−knuthdiv**, the sign of the remainder follows the sign of the dividend. Both methods are arithmetically valid, the latter being almost exclusively used.

−mtoplevel−symbols
−mno−toplevel−symbols
Prepend (do not prepend) a **:** to all global symbols, so the assembly code can be used with the PREFIX assembly directive.

−melf
Generate an executable in the ELF format, rather than the default **mmo** format used by the **mmix** simulator.

−mbranch−predict
−mno−branch−predict
Use (do not use) the probable−branch instructions, when static branch prediction indicates a probable branch.

−mbase−addresses
−mno−base−addresses
Generate (do not generate) code that uses *base addresses*. Using a base address automatically generates a request (handled by the assembler and the linker) for a constant to be set up in a global register. The register is used for one or more base address requests within the range 0 to 255 from the value held in the register. The generally leads to short and fast code, but the number of different data items that can be addressed is limited. This means that a program that uses lots of static data may require **−mno−base−addresses**.

−msingle−exit
−mno−single−exit
Force (do not force) generated code to have a single exit point in each function.

MN10300 Options
These **−m** options are defined for Matsushita MN10300 architectures:

−mmult−bug
Generate code to avoid bugs in the multiply instructions for the MN10300 processors. This is the default.

−mno−mult−bug
Do not generate code to avoid bugs in the multiply instructions for the MN10300 processors.

−mam33
Generate code using features specific to the AM33 processor.

−mno−am33
Do not generate code using features specific to the AM33 processor. This is the default.

−mam33−2
Generate code using features specific to the AM33/2.0 processor.

−mam34
Generate code using features specific to the AM34 processor.

−mtune=*cpu−type*
Use the timing characteristics of the indicated CPU type when scheduling instructions. This does not change the targeted processor type. The CPU type must be one of **mn10300**, **am33**, **am33−2** or **am34**.

−mreturn−pointer−on−d0
When generating a function that returns a pointer, return the pointer in both a0 and d0. Otherwise, the pointer is returned only in a0, and attempts to call such functions without a prototype result in

errors. Note that this option is on by default; use **−mno−return−pointer−on−d0** to disable it.

−mno−crt0

Do not link in the C run−time initialization object file.

−mrelax

Indicate to the linker that it should perform a relaxation optimization pass to shorten branches, calls and absolute memory addresses. This option only has an effect when used on the command line for the final link step.

This option makes symbolic debugging impossible.

−mliw

Allow the compiler to generate *Long Instruction Word* instructions if the target is the **AM33** or later. This is the default. This option defines the preprocessor macro `__LIW__`.

−mno−liw

Do not allow the compiler to generate *Long Instruction Word* instructions. This option defines the preprocessor macro `__NO_LIW__`.

−msetlb

Allow the compiler to generate the *SETLB* and *Lcc* instructions if the target is the **AM33** or later. This is the default. This option defines the preprocessor macro `__SETLB__`.

−mno−setlb

Do not allow the compiler to generate *SETLB* or *Lcc* instructions. This option defines the preprocessor macro `__NO_SETLB__`.

Moxie Options

−meb

Generate big−endian code. This is the default for **moxie−*−*** configurations.

−mel

Generate little−endian code.

−mmul.x

Generate mul.x and umul.x instructions. This is the default for **moxiebox−*−*** configurations.

−mno−crt0

Do not link in the C run−time initialization object file.

MSP430 Options

These options are defined for the MSP430:

−masm−hex

Force assembly output to always use hex constants. Normally such constants are signed decimals, but this option is available for testsuite and/or aesthetic purposes.

−mmcu=

Select the MCU to target. This is used to create a C preprocessor symbol based upon the MCU name, converted to upper case and pre− and post−fixed with `__`. This in turn is used by the *msp430.h* header file to select an MCU−specific supplementary header file.

The option also sets the ISA to use. If the MCU name is one that is known to only support the 430 ISA then that is selected, otherwise the 430X ISA is selected. A generic MCU name of **msp430** can also be used to select the 430 ISA. Similarly the generic **msp430x** MCU name selects the 430X ISA.

In addition an MCU−specific linker script is added to the linker command line. The script's name is the name of the MCU with *.ld* appended. Thus specifying **−mmcu=xxx** on the **gcc** command line defines the C preprocessor symbol `__XXX__` and cause the linker to search for a script called *xxx.ld*.

The ISA and hardware multiply supported for the different MCUs is hard−coded into GCC. However, an external **devices.csv** file can be used to extend device support beyond those that have been hard−coded.

GCC searches for the **devices.csv** file using the following methods in the given precedence order, where the first method takes precedence over the second which takes precedence over the third.

Include path specified with −I and −L
> **devices.csv** is searched for in each of the directories specified by include paths and linker library search paths.

Path specified by the environment variable **MSP430_GCC_INCLUDE_DIR**
> Define the value of the global environment variable **MSP430_GCC_INCLUDE_DIR** to the full path to the directory containing devices.csv, and GCC will search this directory for devices.csv. If devices.csv is found, this directory is also registered as an include path and linker library path. Header files and linker scripts in this directory can therefore be used without manually specifying −I and −L on the command line.

The **msp430−elf{,bare}/include/devices** directory
> Finally, GCC examines **msp430−elf{,bare}/include/devices** from the toolchain root directory. This directory does not exist in a default installation, but if the user has created it and copied **devices.csv** there, then the MCU data is read. As above, this directory is also registered as an include path and linker library path.

If none of the above search methods find **devices.csv**, then the hard−coded MCU data is used.

−mwarn−mcu
−mno−warn−mcu
> This option enables or disables warnings about conflicts between the MCU name specified by the **−mmcu** option and the ISA set by the **−mcpu** option and/or the hardware multiply support set by the **−mhwmult** option. It also toggles warnings about unrecognized MCU names. This option is on by default.

−mcpu=
> Specifies the ISA to use. Accepted values are **msp430**, **msp430x** and **msp430xv2**. This option is deprecated. The **−mmcu=** option should be used to select the ISA.

−msim
> Link to the simulator runtime libraries and linker script. Overrides any scripts that would be selected by the **−mmcu=** option.

−mlarge
> Use large−model addressing (20−bit pointers, 20−bit size_t).

−msmall
> Use small−model addressing (16−bit pointers, 16−bit size_t).

−mrelax
> This option is passed to the assembler and linker, and allows the linker to perform certain optimizations that cannot be done until the final link.

mhwmult=
> Describes the type of hardware multiply supported by the target. Accepted values are **none** for no hardware multiply, **16bit** for the original 16−bit−only multiply supported by early MCUs. **32bit** for the 16/32−bit multiply supported by later MCUs and **f5series** for the 16/32−bit multiply supported by F5−series MCUs. A value of **auto** can also be given. This tells GCC to deduce the hardware multiply support based upon the MCU name provided by the **−mmcu** option. If no **−mmcu** option is specified or if the MCU name is not recognized then no hardware multiply support is assumed. auto is the default setting.

> Hardware multiplies are normally performed by calling a library routine. This saves space in the generated code. When compiling at **−O3** or higher however the hardware multiplier is invoked inline. This makes for bigger, but faster code.

> The hardware multiply routines disable interrupts whilst running and restore the previous interrupt state when they finish. This makes them safe to use inside interrupt handlers as well as in normal

code.

–minrt

Enable the use of a minimum runtime environment − no static initializers or constructors. This is intended for memory−constrained devices. The compiler includes special symbols in some objects that tell the linker and runtime which code fragments are required.

–mtiny−printf

Enable reduced code size `printf` and `puts` library functions. The **tiny** implementations of these functions are not reentrant, so must be used with caution in multi−threaded applications.

Support for streams has been removed and the string to be printed will always be sent to stdout via the `write` syscall. The string is not buffered before it is sent to write.

This option requires Newlib Nano IO, so GCC must be configured with **−−enable−newlib−nano−formatted−io**.

–mmax−inline−shift=

This option takes an integer between 0 and 64 inclusive, and sets the maximum number of inline shift instructions which should be emitted to perform a shift operation by a constant amount. When this value needs to be exceeded, an mspabi helper function is used instead. The default value is 4.

This only affects cases where a shift by multiple positions cannot be completed with a single instruction (e.g. all shifts >1 on the 430 ISA).

Shifts of a 32−bit value are at least twice as costly, so the value passed for this option is divided by 2 and the resulting value used instead.

–mcode−region=
–mdata−region=

These options tell the compiler where to place functions and data that do not have one of the `lower`, `upper`, `either` or `section` attributes. Possible values are `lower`, `upper`, `either` or `any`. The first three behave like the corresponding attribute. The fourth possible value − `any` − is the default. It leaves placement entirely up to the linker script and how it assigns the standard sections (`.text`, `.data`, etc) to the memory regions.

–msilicon−errata=

This option passes on a request to assembler to enable the fixes for the named silicon errata.

–msilicon−errata−warn=

This option passes on a request to the assembler to enable warning messages when a silicon errata might need to be applied.

–mwarn−devices−csv
–mno−warn−devices−csv

Warn if **devices.csv** is not found or there are problem parsing it (default: on).

NDS32 Options

These options are defined for NDS32 implementations:

–mbig−endian

Generate code in big−endian mode.

–mlittle−endian

Generate code in little−endian mode.

–mreduced−regs

Use reduced−set registers for register allocation.

–mfull−regs

Use full−set registers for register allocation.

−mcmov
　　　　Generate conditional move instructions.

−mno−cmov
　　　　Do not generate conditional move instructions.

−mext−perf
　　　　Generate performance extension instructions.

−mno−ext−perf
　　　　Do not generate performance extension instructions.

−mext−perf2
　　　　Generate performance extension 2 instructions.

−mno−ext−perf2
　　　　Do not generate performance extension 2 instructions.

−mext−string
　　　　Generate string extension instructions.

−mno−ext−string
　　　　Do not generate string extension instructions.

−mv3push
　　　　Generate v3 push25/pop25 instructions.

−mno−v3push
　　　　Do not generate v3 push25/pop25 instructions.

−m16−bit
　　　　Generate 16−bit instructions.

−mno−16−bit
　　　　Do not generate 16−bit instructions.

−misr−vector−size=*num*
　　　　Specify the size of each interrupt vector, which must be 4 or 16.

−mcache−block−size=*num*
　　　　Specify the size of each cache block, which must be a power of 2 between 4 and 512.

−march=*arch*
　　　　Specify the name of the target architecture.

−mcmodel=*code−model*
　　　　Set the code model to one of

　　　　small
　　　　　　　All the data and read−only data segments must be within 512KB addressing space. The text
　　　　　　　segment must be within 16MB addressing space.

　　　　medium
　　　　　　　The data segment must be within 512KB while the read−only data segment can be within 4GB
　　　　　　　addressing space. The text segment should be still within 16MB addressing space.

　　　　large
　　　　　　　All the text and data segments can be within 4GB addressing space.

−mctor−dtor
　　　　Enable constructor/destructor feature.

−mrelax
　　　　Guide linker to relax instructions.

Nvidia PTX Options

These options are defined for Nvidia PTX:

−m64

Ignored, but preserved for backward compatibility. Only 64−bit ABI is supported.

−march=*architecture−string*

Generate code for the specified PTX ISA target architecture. Valid architecture strings are **sm_30**, **sm_35**, **sm_37**, **sm_52**, **sm_53**, **sm_70**, **sm_75**, **sm_80**, and **sm_89**. The default depends on how the compiler has been configured, see **−−with−arch**.

This option sets the value of the preprocessor macro __PTX_SM__; for instance, for **sm_35**, it has the value **350**.

−misa=*architecture−string*

Alias of **−march=**.

−march−map=*architecture−string*

Select the closest available **−march=** value that is not more capable. For instance, for **−march−map=sm_50** select **−march=sm_37**, and for **−march−map=sm_53** select **−march=sm_53**.

−mptx=*version−string*

Generate code for the specified PTX ISA version. Valid version strings are **3.1**, **4.1**, **4.2**, **6.0**, **6.3**, **7.0**, **7.3**, and **7.8**. The default PTX ISA version is the one that added support for the selected PTX ISA target architecture, see **−march=**, but at least **6.3**, or **7.3** for **−march=sm_52** and higher.

This option sets the values of the preprocessor macros __PTX_ISA_VERSION_MAJOR__ and __PTX_ISA_VERSION_MINOR__; for instance, for **3.1** the macros have the values **3** and **1**, respectively.

−mmainkernel

Link in code for a __main kernel. This is for stand−alone instead of offloading execution.

−moptimize

Apply partitioned execution optimizations. This is the default when any level of optimization is selected.

−msoft−stack
−mno−soft−stack

For **−mno−soft−stack** (the default, unless **−mgomp** has been specified), use PTX "native" stacks, that is, generate code that uses .local memory or PTX alloca directly for stack storage. Unless **−mptx=7.3** or higher and **−march=sm_52** or higher are active, variable−length arrays and dynamically allocating memory on the stack with alloca are not supported.

For **−msoft−stack** (implied by **−mgomp**), generate code that does not use .local memory or PTX alloca directly for stack storage. Instead, a per−warp stack pointer is maintained explicitly. This enables variable−length stack allocation (with variable−length arrays or alloca), and when global memory is used for underlying storage, makes it possible to access automatic variables from other threads, or with atomic instructions. This code generation variant is used for OpenMP offloading, but the option is exposed on its own for the purpose of testing the compiler; to generate code suitable for linking into programs using OpenMP offloading, use option **−mgomp**.

−muniform−simt

Switch to code generation variant that allows to execute all threads in each warp, while maintaining memory state and side effects as if only one thread in each warp was active outside of OpenMP SIMD regions. All atomic operations and calls to runtime (malloc, free, vprintf) are conditionally executed (iff current lane index equals the master lane index), and the register being assigned is copied via a shuffle instruction from the master lane. Outside of SIMD regions lane 0 is the master; inside, each thread sees itself as the master. Shared memory array int __nvptx_uni[] stores all−zeros or all−ones bitmasks for each warp, indicating current mode (0 outside of SIMD regions). Each thread can bitwise−and the bitmask at position tid.y with current lane index to compute the master lane

index.

−mgomp

Generate code for use in OpenMP offloading: enables **−msoft−stack** and **−muniform−simt** options, and selects corresponding multilib variant.

OpenRISC Options

These options are defined for OpenRISC:

−mboard=*name*

Configure a board specific runtime. This will be passed to the linker for newlib board library linking. The default is or1ksim.

−mnewlib

This option is ignored; it is for compatibility purposes only. This used to select linker and preprocessor options for use with newlib.

−msoft−div
−mhard−div

Select software or hardware divide (l.div, l.divu) instructions. This default is hardware divide.

−msoft−mul
−mhard−mul

Select software or hardware multiply (l.mul, l.muli) instructions. This default is hardware multiply.

−msoft−float
−mhard−float

Select software or hardware for floating point operations. The default is software.

−mdouble−float

When **−mhard−float** is selected, enables generation of double−precision floating point instructions. By default functions from *libgcc* are used to perform double−precision floating point operations.

−munordered−float

When **−mhard−float** is selected, enables generation of unordered floating point compare and set flag (lf.sfun*) instructions. By default functions from *libgcc* are used to perform unordered floating point compare and set flag operations.

−mcmov

Enable generation of conditional move (l.cmov) instructions. By default the equivalent will be generated using set and branch.

−mror

Enable generation of rotate right (l.ror) instructions. By default functions from *libgcc* are used to perform rotate right operations.

−mrori

Enable generation of rotate right with immediate (l.rori) instructions. By default functions from *libgcc* are used to perform rotate right with immediate operations.

−msext

Enable generation of sign extension (l.ext*) instructions. By default memory loads are used to perform sign extension.

−msfimm

Enable generation of compare and set flag with immediate (l.sf*i) instructions. By default extra instructions will be generated to store the immediate to a register first.

−mshftimm

Enable generation of shift with immediate (l.srai, l.srli, l.slli) instructions. By default extra instructions will be generated to store the immediate to a register first.

−mcmodel=small

Generate OpenRISC code for the small model: The GOT is limited to 64k. This is the default model.

−mcmodel=large

Generate OpenRISC code for the large model: The GOT may grow up to 4G in size.

PDP−11 Options

These options are defined for the PDP−11:

−mfpu

Use hardware FPP floating point. This is the default. (FIS floating point on the PDP−11/40 is not supported.) Implies −m45.

−msoft−float

Do not use hardware floating point.

−mac0

Return floating−point results in ac0 (fr0 in Unix assembler syntax).

−mno−ac0

Return floating−point results in memory. This is the default.

−m40

Generate code for a PDP−11/40. Implies −msoft−float −mno−split.

−m45

Generate code for a PDP−11/45. This is the default.

−m10

Generate code for a PDP−11/10. Implies −msoft−float −mno−split.

−mint16
−mno−int32

Use 16−bit `int`. This is the default.

−mint32
−mno−int16

Use 32−bit `int`.

−msplit

Target has split instruction and data space. Implies −m45.

−munix−asm

Use Unix assembler syntax.

−mdec−asm

Use DEC assembler syntax.

−mgnu−asm

Use GNU assembler syntax. This is the default.

−mlra

Use the new LRA register allocator. By default, the old "reload" allocator is used.

PowerPC Options

These are listed under

PRU Options

These command−line options are defined for PRU target:

−minrt

Link with a minimum runtime environment. This can significantly reduce the size of the final ELF binary, but some standard C runtime features are removed.

This option disables support for static initializers and constructors. Beware that the compiler could still generate code with static initializers and constructors. It is up to the programmer to ensure that

the source program will not use those features.

The minimal startup code would not pass `argc` and `argv` arguments to `main`, so the latter must be declared as `int main (void)`. This is already the norm for most firmware projects.

−mmcu=*mcu*

Specify the PRU hardware variant to use. A correspondingly named spec file would be loaded, passing the memory region sizes to the linker and defining hardware−specific C macros.

Newlib provides only the `sim` spec, intended for running regression tests using a simulator. Specs for real hardware can be obtained by installing the GnuPruMcu (`https://github.com/dinuxbg/gnuprumcu/`) package.

−mno−relax

Make GCC pass the **−−no−relax** command−line option to the linker instead of the **−−relax** option.

−mloop

Allow (or do not allow) GCC to use the LOOP instruction.

−mabi=*variant*

Specify the ABI variant to output code for. **−mabi=ti** selects the unmodified TI ABI while **−mabi=gnu** selects a GNU variant that copes more naturally with certain GCC assumptions. These are the differences:

Function Pointer Size

TI ABI specifies that function (code) pointers are 16−bit, whereas GNU supports only 32−bit data and code pointers.

Optional Return Value Pointer

Function return values larger than 64 bits are passed by using a hidden pointer as the first argument of the function. TI ABI, though, mandates that the pointer can be NULL in case the caller is not using the returned value. GNU always passes and expects a valid return value pointer.

The current **−mabi=ti** implementation simply raises a compile error when any of the above code constructs is detected. As a consequence the standard C library cannot be built and it is omitted when linking with **−mabi=ti**.

Relaxation is a GNU feature and for safety reasons is disabled when using **−mabi=ti**. The TI toolchain does not emit relocations for QBBx instructions, so the GNU linker cannot adjust them when shortening adjacent LDI32 pseudo instructions.

RISC−V Options

These command−line options are defined for RISC−V targets:

−mbranch−cost=*n*

Set the cost of branches to roughly *n* instructions.

−mplt
−mno−plt

When generating PIC code, do or don't allow the use of PLTs. Ignored for non−PIC. The default is **−mplt**.

−mabi=*ABI−string*

Specify integer and floating−point calling convention. *ABI−string* contains two parts: the size of integer types and the registers used for floating−point types. For example **−march=rv64ifd −mabi=lp64d** means that **long** and pointers are 64−bit (implicitly defining **int** to be 32−bit), and that floating−point values up to 64 bits wide are passed in F registers. Contrast this with **−march=rv64ifd −mabi=lp64f**, which still allows the compiler to generate code that uses the F and D extensions but only allows floating−point values up to 32 bits long to be passed in registers; or **−march=rv64ifd −mabi=lp64**, in which no floating−point arguments will be passed in registers.

The default for this argument is system dependent, users who want a specific calling convention

should specify one explicitly. The valid calling conventions are: **ilp32**, **ilp32f**, **ilp32d**, **lp64**, **lp64f**, and **lp64d**. Some calling conventions are impossible to implement on some ISAs: for example, **−march=rv32if −mabi=ilp32d** is invalid because the ABI requires 64−bit values be passed in F registers, but F registers are only 32 bits wide. There are also the **ilp32e** ABI that can only be used with the **rv32e** architecture and the **lp64e** ABI that can only be used with the **rv64e**. Those ABIs are not well specified at present, and are subject to change.

−mfdiv
−mno−fdiv
> Do or don't use hardware floating−point divide and square root instructions. This requires the F or D extensions for floating−point registers. The default is to use them if the specified architecture has these instructions.

−mfence−tso
−mno−fence−tso
> Do or don't use the **fence.tso** instruction, which is unimplemented on some processors (including those from T−Head). If the **fence.tso** instruction is not available then a stronger fence is used instead.

−mdiv
−mno−div
> Do or don't use hardware instructions for integer division. This requires the M extension. The default is to use them if the specified architecture has these instructions.

−misa−spec=*ISA−spec−string*
> Specify the version of the RISC−V Unprivileged (formerly User−Level) ISA specification to produce code conforming to. The possibilities for *ISA−spec−string* are:

> `2.2`
>> Produce code conforming to version 2.2.

> `20190608`
>> Produce code conforming to version 20190608.

> `20191213`
>> Produce code conforming to version 20191213.

> The default is **−misa−spec=20191213** unless GCC has been configured with **−−with−isa−spec=** specifying a different default version.

−march=*ISA−string*
> Generate code for given RISC−V ISA (e.g. **rv64im**). ISA strings must be lower−case. Examples include **rv64i**, **rv32g**, **rv32e**, and **rv32imaf**. Additionally, a special value **help** (**−march=help**) is accepted to list all supported extensions.

> The syntax of the ISA string is defined as follows:

> `The string must start with` **rv32** `or` **rv64,** `followed by`
>> **i**, **e**, or **g**, referred to as the base ISA.

> `The subsequent part of the string is a list of extension names.`
> `Extension`
>> names can be categorized as multi−letter (e.g. **zba**) and single−letter (e.g. **v**). Single−letter extensions can appear consecutively, but multi−letter extensions must be separated by underscores.

> `An underscore can appear anywhere after the base ISA. It has no`
> `specific`
>> effect but is used to improve readability and can act as a separator.

> `Extension names may include an optional version number, following`
> `the`
>> syntax **<major>p<minor>** or **<major>**, (e.g. **m2p1** or **m2**).

Supported extension are listed below:

Extension Name : *Supported Version* : *Description*

i   @tab 2.0, 2.1 @tab Base integer extension.

e   @tab 2.0 @tab Reduced base integer extension.

g   @tab − @tab General−purpose computing base extension, **g** will expand to **i**, **m**, **a**, **f**, **d**, **zicsr** and **zifencei**.

m   @tab 2.0 @tab Integer multiplication and division extension.

a   @tab 2.0, 2.1 @tab Atomic extension.

f   @tab 2.0, 2.2 @tab Single−precision floating−point extension.

d   @tab 2.0, 2.2 @tab Double−precision floating−point extension.

c   @tab 2.0 @tab Compressed extension.

h   @tab 1.0 @tab Hypervisor extension.

v   @tab 1.0 @tab Vector extension.

zicsr
    @tab 2.0 @tab Control and status register access extension.

zifencei
    @tab 2.0 @tab Instruction−fetch fence extension.

zicond
    @tab 1.0 @tab Integer conditional operations extension.

za64rs
    @tab 1.0 @tab Reservation set size of 64 bytes.

za128rs
    @tab 1.0 @tab Reservation set size of 128 bytes.

zawrs
    @tab 1.0 @tab Wait−on−reservation−set extension.

zba @tab 1.0 @tab Address calculation extension.

zbb @tab 1.0 @tab Basic bit manipulation extension.

zbc @tab 1.0 @tab Carry−less multiplication extension.

zbs @tab 1.0 @tab Single−bit operation extension.

zfinx
    @tab 1.0 @tab Single−precision floating−point in integer registers extension.

zdinx
    @tab 1.0 @tab Double−precision floating−point in integer registers extension.

zhinx
    @tab 1.0 @tab Half−precision floating−point in integer registers extension.

zhinxmin
    @tab 1.0 @tab Minimal half−precision floating−point in integer registers extension.

zbkb
    @tab 1.0 @tab Cryptography bit−manipulation extension.

zbkc
    @tab 1.0 @tab Cryptography carry−less multiply extension.

zbkx
    @tab 1.0 @tab Cryptography crossbar permutation extension.

zkne
> `@tab` 1.0 `@tab` AES Encryption extension.

zknd
> `@tab` 1.0 `@tab` AES Decryption extension.

zknh
> `@tab` 1.0 `@tab` Hash function extension.

zkr `@tab` 1.0 `@tab` Entropy source extension.

zksed
> `@tab` 1.0 `@tab` SM4 block cipher extension.

zksh
> `@tab` 1.0 `@tab` SM3 hash function extension.

zkt `@tab` 1.0 `@tab` Data independent execution latency extension.

zk   `@tab` 1.0 `@tab` Standard scalar cryptography extension.

zkn `@tab` 1.0 `@tab` NIST algorithm suite extension.

zks `@tab` 1.0 `@tab` ShangMi algorithm suite extension.

zihintntl
> `@tab` 1.0 `@tab` Non−temporal locality hints extension.

zihintpause
> `@tab` 1.0 `@tab` Pause hint extension.

zicboz
> `@tab` 1.0 `@tab` Cache−block zero extension.

zicbom
> `@tab` 1.0 `@tab` Cache−block management extension.

zicbop
> `@tab` 1.0 `@tab` Cache−block prefetch extension.

zic64b
> `@tab` 1.0 `@tab` Cache block size isf 64 bytes.

ziccamoa
> `@tab` 1.0 `@tab` Main memory supports all atomics in A.

ziccif
> `@tab` 1.0 `@tab` Main memory supports instruction fetch with atomicity requirement.

zicclsm
> `@tab` 1.0 `@tab` Main memory supports misaligned loads/stores.

ziccrse
> `@tab` 1.0 `@tab` Main memory supports forward progress on LR/SC sequences.

zicntr
> `@tab` 2.0 `@tab` Standard extension for base counters and timers.

zihpm
> `@tab` 2.0 `@tab` Standard extension for hardware performance counters.

ztso
> `@tab` 1.0 `@tab` Total store ordering extension.

zve32x
> `@tab` 1.0 `@tab` Vector extensions for embedded processors.

zve32f
@tab 1.0 @tab Vector extensions for embedded processors.

zve64x
@tab 1.0 @tab Vector extensions for embedded processors.

zve64f
@tab 1.0 @tab Vector extensions for embedded processors.

zve64d
@tab 1.0 @tab Vector extensions for embedded processors.

zvl32b
@tab 1.0 @tab Minimum vector length standard extensions

zvl64b
@tab 1.0 @tab Minimum vector length standard extensions

zvl128b
@tab 1.0 @tab Minimum vector length standard extensions

zvl256b
@tab 1.0 @tab Minimum vector length standard extensions

zvl512b
@tab 1.0 @tab Minimum vector length standard extensions

zvl1024b
@tab 1.0 @tab Minimum vector length standard extensions

zvl2048b
@tab 1.0 @tab Minimum vector length standard extensions

zvl4096b
@tab 1.0 @tab Minimum vector length standard extensions

zvbb
@tab 1.0 @tab Vector basic bit−manipulation extension.

zvbc
@tab 1.0 @tab Vector carryless multiplication extension.

zvkb
@tab 1.0 @tab Vector cryptography bit−manipulation extension.

zvkg
@tab 1.0 @tab Vector GCM/GMAC extension.

zvkned
@tab 1.0 @tab Vector AES block cipher extension.

zvknha
@tab 1.0 @tab Vector SHA−2 secure hash extension.

zvknhb
@tab 1.0 @tab Vector SHA−2 secure hash extension.

zvksed
@tab 1.0 @tab Vector SM4 Block Cipher extension.

zvksh
@tab 1.0 @tab Vector SM3 Secure Hash extension.

zvkn
@tab 1.0 @tab Vector NIST Algorithm Suite extension, **zvkn** will expand to **zvkned**, **zvknhb**, **zvkb** and **zvkt**.

zvknc
> `@tab` 1.0 `@tab` Vector NIST Algorithm Suite with carryless multiply extension, **zvknc** will expand to **zvkn** and **zvbc**.

zvkng
> `@tab` 1.0 `@tab` Vector NIST Algorithm Suite with GCM extension, **zvkng** will expand to **zvkn** and **zvkg**.

zvks
> `@tab` 1.0 `@tab` Vector ShangMi algorithm suite extension, **zvks** will expand to **zvksed**, **zvksh**, **zvkb** and **zvkt**.

zvksc
> `@tab` 1.0 `@tab` Vector ShangMi algorithm suite with carryless multiplication extension, **zvksc** will expand to **zvks** and **zvbc**.

zvksg
> `@tab` 1.0 `@tab` Vector ShangMi algorithm suite with GCM extension, **zvksg** will expand to **zvks** and **zvkg**.

zvkt
> `@tab` 1.0 `@tab` Vector data independent execution latency extension.

zfh `@tab` 1.0 `@tab` Half–precision floating–point extension.

zfhmin
> `@tab` 1.0 `@tab` Minimal half–precision floating–point extension.

zvfh
> `@tab` 1.0 `@tab` Vector half–precision floating–point extension.

zvfhmin
> `@tab` 1.0 `@tab` Vector minimal half–precision floating–point extension.

zvfbfmin
> `@tab` 1.0 `@tab` Vector BF16 converts extension.

zfa `@tab` 1.0 `@tab` Additional floating–point extension.

zmmul
> `@tab` 1.0 `@tab` Integer multiplication extension.

zca `@tab` 1.0 `@tab` Integer compressed instruction extension.

zcf `@tab` 1.0 `@tab` Compressed single–precision floating point loads and stores extension.

zcd `@tab` 1.0 `@tab` Compressed double–precision floating point loads and stores extension.

zcb `@tab` 1.0 `@tab` Simple compressed instruction extension.

zce `@tab` 1.0 `@tab` Compressed instruction extensions for embedded processors.

zcmp
> `@tab` 1.0 `@tab` Compressed push pop extension.

zcmt
> `@tab` 1.0 `@tab` Table jump instruction extension.

smaia
> `@tab` 1.0 `@tab` Advanced interrupt architecture extension.

smepmp
> `@tab` 1.0 `@tab` PMP Enhancements for memory access and execution prevention on Machine mode.

smstateen
> `@tab` 1.0 `@tab` State enable extension.

ssaia
  `@tab` 1.0 `@tab` Advanced interrupt architecture extension for supervisor−mode.

sscofpmf
  `@tab` 1.0 `@tab` Count overflow & filtering extension.

ssstateen
  `@tab` 1.0 `@tab` State−enable extension for supervisor−mode.

sstc `@tab` 1.0 `@tab` Supervisor−mode timer interrupts extension.

svinval
  `@tab` 1.0 `@tab` Fine−grained address−translation cache invalidation extension.

svnapot
  `@tab` 1.0 `@tab` NAPOT translation contiguity extension.

svpbmt
  `@tab` 1.0 `@tab` Page−based memory types extension.

xcvmac
  `@tab` 1.0 `@tab` Core−V multiply−accumulate extension.

xcvalu
  `@tab` 1.0 `@tab` Core−V miscellaneous ALU extension.

xcvelw
  `@tab` 1.0 `@tab` Core−V event load word extension.

xtheadba
  `@tab` 1.0 `@tab` T−head address calculation extension.

xtheadbb
  `@tab` 1.0 `@tab` T−head basic bit−manipulation extension.

xtheadbs
  `@tab` 1.0 `@tab` T−head single−bit instructions extension.

xtheadcmo
  `@tab` 1.0 `@tab` T−head cache management operations extension.

xtheadcondmov
  `@tab` 1.0 `@tab` T−head conditional move extension.

xtheadfmemidx
  `@tab` 1.0 `@tab` T−head indexed memory operations for floating−point registers extension.

xtheadfmv
  `@tab` 1.0 `@tab` T−head double floating−point high−bit data transmission extension.

xtheadint
  `@tab` 1.0 `@tab` T−head acceleration interruption extension.

xtheadmac
  `@tab` 1.0 `@tab` T−head multiply−accumulate extension.

xtheadmemidx
  `@tab` 1.0 `@tab` T−head indexed memory operation extension.

xtheadmempair
  `@tab` 1.0 `@tab` T−head two−GPR memory operation extension.

xtheadsync
  `@tab` 1.0 `@tab` T−head multi−core synchronization extension.

xventanacondops
  `@tab` 1.0 `@tab` Ventana integer conditional operations extension.

When **−march=** is not specified, use the setting from **−mcpu**.

If both **−march** and **−mcpu=** are not specified, the default for this argument is system dependent, users who want a specific architecture extensions should specify one explicitly.

When the RISC−V specifications define an extension as depending on other extensions, GCC will implicitly add the dependent extensions to the enabled extension set if they weren't added explicitly.

−mcpu=*processor−string*
Use architecture of and optimize the output for the given processor, specified by particular CPU name. Permissible values for this option are: **sifive−e20**, **sifive−e21**, **sifive−e24**, **sifive−e31**, **sifive−e34**, **sifive−e76**, **sifive−s21**, **sifive−s51**, **sifive−s54**, **sifive−s76**, **sifive−u54**, **sifive−u74**, **sifive−x280**, **sifive−xp450**, **sifive−x670**, **thead−c906**, **tt−ascalon−d8**, **xiangshan−nanhu**.

Note that **−mcpu** does not override **−march** or **−mtune**.

−mtune=*processor−string*
Optimize the output for the given processor, specified by microarchitecture or particular CPU name. Permissible values for this option are: **generic−ooo**, **rocket**, **sifive−3−series**, **sifive−5−series**, **sifive−7−series**, **size**, **sifive−p400−series**, **sifive−p600−series**, and all valid options for **−mcpu=**.

When **−mtune=** is not specified, use the setting from **−mcpu**, the default is **rocket** if both are not specified.

The **size** choice is not intended for use by end−users. This is used when **−Os** is specified. It overrides the instruction cost info provided by **−mtune=**, but does not override the pipeline info. This helps reduce code size while still giving good performance.

−mpreferred−stack−boundary=*num*
Attempt to keep the stack boundary aligned to a 2 raised to *num* byte boundary. If **−mpreferred−stack−boundary** is not specified, the default is 4 (16 bytes or 128−bits).

Warning: If you use this switch, then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

−msmall−data−limit=*n*
Put global and static data smaller than *n* bytes into a special section (on some targets).

−msave−restore
−mno−save−restore
Do or don't use smaller but slower prologue and epilogue code that uses library function calls. The default is to use fast inline prologues and epilogues.

−mmovcc
−mno−movcc
Do or don't produce branchless conditional−move code sequences even with targets that do not have specific instructions for conditional operations. If enabled, sequences of ALU operations are produced using base integer ISA instructions where profitable.

−minline−atomics
−mno−inline−atomics
Do or don't use smaller but slower subword atomic emulation code that uses libatomic function calls. The default is to use fast inline subword atomics that do not require libatomic.

−minline−strlen
−mno−inline−strlen
Do or do not attempt to inline strlen calls if possible. Inlining will only be done if the string is properly aligned and instructions for accelerated processing are available. The default is to not inline strlen calls.

−minline−strcmp

−mno−inline−strcmp

Do or do not attempt to inline strcmp calls if possible. Inlining will only be done if the strings are properly aligned and instructions for accelerated processing are available. The default is to not inline strcmp calls.

The **−−param riscv−strcmp−inline−limit=***n* parameter controls the maximum number of bytes compared by the inlined code. The default value is 64.

−minline−strncmp
−mno−inline−strncmp

Do or do not attempt to inline strncmp calls if possible. Inlining will only be done if the strings are properly aligned and instructions for accelerated processing are available. The default is to not inline strncmp calls.

The **−−param riscv−strcmp−inline−limit=***n* parameter controls the maximum number of bytes compared by the inlined code. The default value is 64.

−mshorten−memrefs
−mno−shorten−memrefs

Do or do not attempt to make more use of compressed load/store instructions by replacing a load/store of 'base register + large offset' with a new load/store of 'new base + small offset'. If the new base gets stored in a compressed register, then the new load/store can be compressed. Currently targets 32−bit integer load/stores only.

−mstrict−align
−mno−strict−align

Do not or do generate unaligned memory accesses. The default is set depending on whether the processor we are optimizing for supports fast unaligned access or not.

−mscalar−strict−align
−mno−scalar−strict−align

Do not or do generate unaligned memory accesses. The default is set depending on whether the processor we are optimizing for supports fast unaligned access or not. This is an alias for **−mstrict−align**.

−mvector−strict−align
−mno−vector−strict−align

Do not or do generate unaligned vector memory accesses. The default is set to off unless the processor we are optimizing for explicitly supports element−misaligned vector memory access.

−mcmodel=medlow

Generate code for the medium−low code model. The program and its statically defined symbols must lie within a single 2 GiB address range and must lie between absolute addresses −2 GiB and +2 GiB. Programs can be statically or dynamically linked. This is the default code model unless GCC has been configured with **−−with−cmodel=** specifying a different default code model.

−mcmodel=medany

Generate code for the medium−any code model. The program and its statically defined symbols must be within any single 2 GiB address range. Programs can be statically or dynamically linked.

The code generated by the medium−any code model is position−independent, but is not guaranteed to function correctly when linked into position−independent executables or libraries.

−mcmodel=large

Generate code for a large code model, which has no restrictions on size or placement of symbols.

−mexplicit−relocs
−mno−exlicit−relocs

Use or do not use assembler relocation operators when dealing with symbolic addresses. The alternative is to use assembler macros instead, which may limit optimization.

−mrelax
−mno−relax
>   Take advantage of linker relaxations to reduce the number of instructions required to materialize symbol addresses. The default is to take advantage of linker relaxations.

−mriscv−attribute
−mno−riscv−attribute
>   Emit (do not emit) RISC−V attribute to record extra information into ELF objects. This feature requires at least binutils 2.32.

−mcsr−check
−mno−csr−check
>   Enables or disables the CSR checking.

−malign−data=*type*
>   Control how GCC aligns variables and constants of array, structure, or union types. Supported values for *type* are **xlen** which uses x register width as the alignment value, and **natural** which uses natural alignment. **xlen** is the default.

−mbig−endian
>   Generate big−endian code. This is the default when GCC is configured for a **riscv64be−*−*** or **riscv32be−*−*** target.

−mlittle−endian
>   Generate little−endian code. This is the default when GCC is configured for a **riscv64−*−*** or **riscv32−*−*** but not a **riscv64be−*−*** or **riscv32be−*−*** target.

−mstack−protector−guard=*guard*
−mstack−protector−guard−reg=*reg*
−mstack−protector−guard−offset=*offset*
>   Generate stack protection code using canary at *guard*. Supported locations are **global** for a global canary or **tls** for per−thread canary in the TLS block.
>
>   With the latter choice the options **−mstack−protector−guard−reg=***reg* and **−mstack−protector−guard−offset=***offset* furthermore specify which register to use as base register for reading the canary, and from what offset from that base register. There is no default register or offset as this is entirely for use within the Linux kernel.

−mtls−dialect=desc
>   Use TLS descriptors as the thread−local storage mechanism for dynamic accesses of TLS variables.

−mtls−dialect=trad
>   Use traditional TLS as the thread−local storage mechanism for dynamic accesses of TLS variables. This is the default.

RL78 Options
−msim
>   Links in additional target libraries to support operation within a simulator.

−mmul=none
−mmul=g10
−mmul=g13
−mmul=g14
−mmul=rl78
>   Specifies the type of hardware multiplication and division support to be used. The simplest is `none`, which uses software for both multiplication and division. This is the default. The `g13` value is for the hardware multiply/divide peripheral found on the RL78/G13 (S2 core) targets. The `g14` value selects the use of the multiplication and division instructions supported by the RL78/G14 (S3 core) parts. The value `rl78` is an alias for `g14` and the value `mg10` is an alias for `none`.
>
>   In addition a C preprocessor macro is defined, based upon the setting of this option. Possible values

are: `__RL78_MUL_NONE__`, `__RL78_MUL_G13__` or `__RL78_MUL_G14__`.

−mcpu=g10
−mcpu=g13
−mcpu=g14
−mcpu=rl78

Specifies the RL78 core to target. The default is the G14 core, also known as an S3 core or just RL78. The G13 or S2 core does not have multiply or divide instructions, instead it uses a hardware peripheral for these operations. The G10 or S1 core does not have register banks, so it uses a different calling convention.

If this option is set it also selects the type of hardware multiply support to use, unless this is overridden by an explicit **−mmul=none** option on the command line. Thus specifying **−mcpu=g13** enables the use of the G13 hardware multiply peripheral and specifying **−mcpu=g10** disables the use of hardware multiplications altogether.

Note, although the RL78/G14 core is the default target, specifying **−mcpu=g14** or **−mcpu=rl78** on the command line does change the behavior of the toolchain since it also enables G14 hardware multiply support. If these options are not specified on the command line then software multiplication routines will be used even though the code targets the RL78 core. This is for backwards compatibility with older toolchains which did not have hardware multiply and divide support.

In addition a C preprocessor macro is defined, based upon the setting of this option. Possible values are: `__RL78_G10__`, `__RL78_G13__` or `__RL78_G14__`.

−mg10
−mg13
−mg14
−mrl78

These are aliases for the corresponding **−mcpu=** option. They are provided for backwards compatibility.

−mallregs

Allow the compiler to use all of the available registers. By default registers `r24..r31` are reserved for use in interrupt handlers. With this option enabled these registers can be used in ordinary functions as well.

−m64bit−doubles
−m32bit−doubles

Make the `double` data type be 64 bits (**−m64bit−doubles**) or 32 bits (**−m32bit−doubles**) in size. The default is **−m32bit−doubles**.

−msave−mduc−in−interrupts
−mno−save−mduc−in−interrupts

Specifies that interrupt handler functions should preserve the MDUC registers. This is only necessary if normal code might use the MDUC registers, for example because it performs multiplication and division operations. The default is to ignore the MDUC registers as this makes the interrupt handlers faster. The target option −mg13 needs to be passed for this to work as this feature is only available on the G13 target (S2 core). The MDUC registers will only be saved if the interrupt handler performs a multiplication or division operation or it calls another function.

IBM RS/6000 and PowerPC Options

These **−m** options are defined for the IBM RS/6000 and PowerPC:

−mpowerpc−gpopt
−mno−powerpc−gpopt
−mpowerpc−gfxopt
−mno−powerpc−gfxopt

−mpowerpc64
−mno−powerpc64
−mmfcrf
−mno−mfcrf
−mpopcntb
−mno−popcntb
−mpopcntd
−mno−popcntd
−mfprnd
−mno−fprnd
−mcmpb
−mno−cmpb
−mhard−dfp
−mno−hard−dfp

You use these options to specify which instructions are available on the processor you are using. The default value of these options is determined when configuring GCC. Specifying the **−mcpu=***cpu_type* overrides the specification of these options. We recommend you use the **−mcpu=***cpu_type* option rather than the options listed above.

Specifying **−mpowerpc−gpopt** allows GCC to use the optional PowerPC architecture instructions in the General Purpose group, including floating−point square root. Specifying **−mpowerpc−gfxopt** allows GCC to use the optional PowerPC architecture instructions in the Graphics group, including floating−point select.

The **−mmfcrf** option allows GCC to generate the move from condition register field instruction implemented on the POWER4 processor and other processors that support the PowerPC V2.01 architecture. The **−mpopcntb** option allows GCC to generate the popcount and double−precision FP reciprocal estimate instruction implemented on the POWER5 processor and other processors that support the PowerPC V2.02 architecture. The **−mpopcntd** option allows GCC to generate the popcount instruction implemented on the POWER7 processor and other processors that support the PowerPC V2.06 architecture. The **−mfprnd** option allows GCC to generate the FP round to integer instructions implemented on the POWER5+ processor and other processors that support the PowerPC V2.03 architecture. The **−mcmpb** option allows GCC to generate the compare bytes instruction implemented on the POWER6 processor and other processors that support the PowerPC V2.05 architecture. The **−mhard−dfp** option allows GCC to generate the decimal floating−point instructions implemented on some POWER processors.

The **−mpowerpc64** option allows GCC to generate the additional 64−bit instructions that are found in the full PowerPC64 architecture and to treat GPRs as 64−bit, doubleword quantities. GCC defaults to **−mno−powerpc64**.

−mcpu=*cpu_type*

Set architecture type, register usage, and instruction scheduling parameters for machine type *cpu_type*. Supported values for *cpu_type* are **401**, **403**, **405**, **405fp**, **440**, **440fp**, **464**, **464fp**, **476**, **476fp**, **505**, **601**, **602**, **603**, **603e**, **604**, **604e**, **620**, **630**, **740**, **7400**, **7450**, **750**, **801**, **821**, **823**, **860**, **970**, **8540**, **a2**, **e300c2**, **e300c3**, **e500mc**, **e500mc64**, **e5500**, **e6500**, **ec603e**, **G3**, **G4**, **G5**, **titan**, **power3**, **power4**, **power5**, **power5+**, **power6**, **power6x**, **power7**, **power8**, **power9**, **power10**, **power11**, **powerpc**, **powerpc64**, **powerpc64le**, **rs64**, and **native**.

−mcpu=powerpc, **−mcpu=powerpc64**, and **−mcpu=powerpc64le** specify pure 32−bit PowerPC (either endian), 64−bit big endian PowerPC and 64−bit little endian PowerPC architecture machine types, with an appropriate, generic processor model assumed for scheduling purposes.

Specifying **native** as cpu type detects and selects the architecture option that corresponds to the host processor of the system performing the compilation. **−mcpu=native** has no effect if GCC does not recognize the processor.

The other options specify a specific processor. Code generated under those options runs best on that

processor, and may not run at all on others.

The **−mcpu** options automatically enable or disable the following options:

−maltivec −mfprnd −mhard−float −mmfcrf −mmultiple −mpopcntb −mpopcntd −mpowerpc64 −mpowerpc−gpopt −mpowerpc−gfxopt −mmulhw −mdlmzb −mmfpgpr −mvsx −mcrypto −mhtm −mpower8−fusion −mquad−memory −mquad−memory−atomic −mfloat128 −mfloat128−hardware −mprefixed −mpcrel −mmma −mrop−protect

The particular options set for any particular CPU varies between compiler versions, depending on what setting seems to produce optimal code for that CPU; it doesn't necessarily reflect the actual hardware's capabilities. If you wish to set an individual option to a particular value, you may specify it after the **−mcpu** option, like **−mcpu=970 −mno−altivec**.

On AIX, the **−maltivec** and **−mpowerpc64** options are not enabled or disabled by the **−mcpu** option at present because AIX does not have full support for these options. You may still enable or disable them individually if you're sure it'll work in your environment.

−mtune=*cpu_type*
> Set the instruction scheduling parameters for machine type *cpu_type*, but do not set the architecture type or register usage, as **−mcpu=***cpu_type* does. The same values for *cpu_type* are used for **−mtune** as for **−mcpu**. If both are specified, the code generated uses the architecture and registers set by **−mcpu**, but the scheduling parameters set by **−mtune**.

−mcmodel=small
> Generate PowerPC64 code for the small model: The TOC is limited to 64k.

−mcmodel=medium
> Generate PowerPC64 code for the medium model: The TOC and other static data may be up to a total of 4G in size. This is the default for 64−bit Linux.

−mcmodel=large
> Generate PowerPC64 code for the large model: The TOC may be up to 4G in size. Other data and code is only limited by the 64−bit address space.

−maltivec
−mno−altivec
> Generate code that uses (does not use) AltiVec instructions, and also enable the use of built−in functions that allow more direct access to the AltiVec instruction set. You may also need to set **−mabi=altivec** to adjust the current ABI with AltiVec ABI enhancements.
>
> When **−maltivec** is used, the element order for AltiVec intrinsics such as `vec_splat`, `vec_extract`, and `vec_insert` match array element order corresponding to the endianness of the target. That is, element zero identifies the leftmost element in a vector register when targeting a big−endian platform, and identifies the rightmost element in a vector register when targeting a little−endian platform.

−mvrsave
−mno−vrsave
> Generate VRSAVE instructions when generating AltiVec code.

−msecure−plt
> Generate code that allows **ld** and **ld.so** to build executables and shared libraries with non−executable `.plt` and `.got` sections. This is a PowerPC 32−bit SYSV ABI option.

−mbss−plt
> Generate code that uses a BSS `.plt` section that **ld.so** fills in, and requires `.plt` and `.got` sections that are both writable and executable. This is a PowerPC 32−bit SYSV ABI option.

−msplit−patch−nops
> When adding NOPs for a patchable area via the **−fpatchable−function−entry** option emit the "before" NOPs in front of the global entry point and the "after" NOPs after the local entry point. This

makes the sequence of NOPs not consecutive when a global entry point is generated. Without this option the NOPs are emitted directly before and after the local entry point, making them consecutive but moving global and local entry point further apart. If only a single entry point is generated this option has no effect.

−misel
−mno−isel
This switch enables or disables the generation of ISEL instructions.

−mvsx
−mno−vsx
Generate code that uses (does not use) vector/scalar (VSX) instructions, and also enable the use of built−in functions that allow more direct access to the VSX instruction set.

−mcrypto
−mno−crypto
Enable the use (disable) of the built−in functions that allow direct access to the cryptographic instructions that were added in version 2.07 of the PowerPC ISA.

−mhtm
−mno−htm
Enable (disable) the use of the built−in functions that allow direct access to the Hardware Transactional Memory (HTM) instructions that were added in version 2.07 of the PowerPC ISA.

−mpower8−fusion
−mno−power8−fusion
Generate code that keeps (does not keeps) some integer operations adjacent so that the instructions can be fused together on power8 and later processors.

−mquad−memory
−mno−quad−memory
Generate code that uses (does not use) the non−atomic quad word memory instructions. The **−mquad−memory** option requires use of 64−bit mode.

−mquad−memory−atomic
−mno−quad−memory−atomic
Generate code that uses (does not use) the atomic quad word memory instructions. The **−mquad−memory−atomic** option requires use of 64−bit mode.

−mfloat128
−mno−float128
Enable/disable the *__float128* keyword for IEEE 128−bit floating point and use either software emulation for IEEE 128−bit floating point or hardware instructions.

The VSX instruction set (**−mvsx**) must be enabled to use the IEEE 128−bit floating point support. The IEEE 128−bit floating point is only supported on Linux.

The default for **−mfloat128** is enabled on PowerPC Linux systems using the VSX instruction set, and disabled on other systems.

If you use the ISA 3.0 instruction set (**−mcpu=power9**) on a 64−bit system, the IEEE 128−bit floating point support will also enable the generation of ISA 3.0 IEEE 128−bit floating point instructions. Otherwise, if you do not specify to generate ISA 3.0 instructions or you are targeting a 32−bit big endian system, IEEE 128−bit floating point will be done with software emulation.

−mfloat128−hardware
−mno−float128−hardware
Enable/disable using ISA 3.0 hardware instructions to support the *__float128* data type.

The default for **−mfloat128−hardware** is enabled on PowerPC Linux systems using the ISA 3.0 instruction set, and disabled on other systems.

−m32
−m64

Generate code for 32−bit or 64−bit environments of Darwin and SVR4 targets (including GNU/Linux). The 32−bit environment sets int, long and pointer to 32 bits and generates code that runs on any PowerPC variant. The 64−bit environment sets int to 32 bits and long and pointer to 64 bits, and generates code for PowerPC64, as for **−mpowerpc64**.

−mfull−toc
−mno−fp−in−toc
−mno−sum−in−toc
−mminimal−toc

Modify generation of the TOC (Table Of Contents), which is created for every executable file. The **−mfull−toc** option is selected by default. In that case, GCC allocates at least one TOC entry for each unique non−automatic variable reference in your program. GCC also places floating−point constants in the TOC. However, only 16,384 entries are available in the TOC.

If you receive a linker error message that saying you have overflowed the available TOC space, you can reduce the amount of TOC space used with the **−mno−fp−in−toc** and **−mno−sum−in−toc** options. **−mno−fp−in−toc** prevents GCC from putting floating−point constants in the TOC and **−mno−sum−in−toc** forces GCC to generate code to calculate the sum of an address and a constant at run time instead of putting that sum into the TOC. You may specify one or both of these options. Each causes GCC to produce very slightly slower and larger code at the expense of conserving TOC space.

If you still run out of space in the TOC even when you specify both of these options, specify **−mminimal−toc** instead. This option causes GCC to make only one TOC entry for every file. When you specify this option, GCC produces code that is slower and larger but which uses extremely little TOC space. You may wish to use this option only on files that contain less frequently−executed code.

−maix64
−maix32

Enable 64−bit AIX ABI and calling convention: 64−bit pointers, 64−bit long type, and the infrastructure needed to support them. Specifying **−maix64** implies **−mpowerpc64**, while **−maix32** disables the 64−bit ABI and implies **−mno−powerpc64**. GCC defaults to **−maix32**.

−mxl−compat
−mno−xl−compat

Produce code that conforms more closely to IBM XL compiler semantics when using AIX−compatible ABI. Pass floating−point arguments to prototyped functions beyond the register save area (RSA) on the stack in addition to argument FPRs. Do not assume that most significant double in 128−bit long double value is properly rounded when comparing values and converting to double. Use XL symbol names for long double support routines.

The AIX calling convention was extended but not initially documented to handle an obscure K&R C case of calling a function that takes the address of its arguments with fewer arguments than declared. IBM XL compilers access floating−point arguments that do not fit in the RSA from the stack when a subroutine is compiled without optimization. Because always storing floating−point arguments on the stack is inefficient and rarely needed, this option is not enabled by default and only is necessary when calling subroutines compiled by IBM XL compilers without optimization.

−mpe

Support *IBM RS/6000 SP Parallel Environment* (PE). Link an application written to use message passing with special startup code to enable the application to run. The system must have PE installed in the standard location (*/usr/lpp/ppe.poe/*), or the *specs* file must be overridden with the **−specs=** option to specify the appropriate directory location. The Parallel Environment does not support threads, so the **−mpe** option and the **−pthread** option are incompatible.

−malign−natural
−malign−power
> On AIX, 32−bit Darwin, and 64−bit PowerPC GNU/Linux, the option **−malign−natural** overrides the ABI−defined alignment of larger types, such as floating−point doubles, on their natural size−based boundary. The option **−malign−power** instructs GCC to follow the ABI−specified alignment rules. GCC defaults to the standard alignment defined in the ABI.
>
> On 64−bit Darwin, natural alignment is the default, and **−malign−power** is not supported.

−msoft−float
−mhard−float
> Generate code that does not use (uses) the floating−point register set. Software floating−point emulation is provided if you use the **−msoft−float** option, and pass the option to GCC when linking.

−mmultiple
−mno−multiple
> Generate code that uses (does not use) the load multiple word instructions and the store multiple word instructions. These instructions are generated by default on POWER systems, and not generated on PowerPC systems. Do not use **−mmultiple** on little−endian PowerPC systems, since those instructions do not work when the processor is in little−endian mode. The exceptions are PPC740 and PPC750 which permit these instructions in little−endian mode.

−mupdate
−mno−update
> Generate code that uses (does not use) the load or store instructions that update the base register to the address of the calculated memory location. These instructions are generated by default. If you use **−mno−update**, there is a small window between the time that the stack pointer is updated and the address of the previous frame is stored, which means code that walks the stack frame across interrupts or signals may get corrupted data.

−mavoid−indexed−addresses
−mno−avoid−indexed−addresses
> Generate code that tries to avoid (not avoid) the use of indexed load or store instructions. These instructions can incur a performance penalty on Power6 processors in certain situations, such as when stepping through large arrays that cross a 16M boundary. This option is enabled by default when targeting Power6 and disabled otherwise.

−mfused−madd
−mno−fused−madd
> Generate code that uses (does not use) the floating−point multiply and accumulate instructions. These instructions are generated by default if hardware floating point is used. The machine−dependent **−mfused−madd** option is now mapped to the machine−independent **−ffp−contract=fast** option, and **−mno−fused−madd** is mapped to **−ffp−contract=off**.

−mmulhw
−mno−mulhw
> Generate code that uses (does not use) the half−word multiply and multiply−accumulate instructions on the IBM 405, 440, 464 and 476 processors. These instructions are generated by default when targeting those processors.

−mdlmzb
−mno−dlmzb
> Generate code that uses (does not use) the string−search **dlmzb** instruction on the IBM 405, 440, 464 and 476 processors. This instruction is generated by default when targeting those processors.

−mno−bit−align
−mbit−align
> On System V.4 and embedded PowerPC systems do not (do) force structures and unions that contain bit−fields to be aligned to the base type of the bit−field.

For example, by default a structure containing nothing but 8 `unsigned` bit−fields of length 1 is aligned to a 4−byte boundary and has a size of 4 bytes. By using **−mno−bit−align**, the structure is aligned to a 1−byte boundary and is 1 byte in size.

−mno−strict−align
−mstrict−align
> On System V.4 and embedded PowerPC systems do not (do) assume that unaligned memory references are handled by the system.

−mrelocatable
−mno−relocatable
> Generate code that allows (does not allow) a static executable to be relocated to a different address at run time. A simple embedded PowerPC system loader should relocate the entire contents of `.got2` and 4−byte locations listed in the `.fixup` section, a table of 32−bit addresses generated by this option. For this to work, all objects linked together must be compiled with **−mrelocatable** or **−mrelocatable−lib**. **−mrelocatable** code aligns the stack to an 8−byte boundary.

−mrelocatable−lib
−mno−relocatable−lib
> Like **−mrelocatable**, **−mrelocatable−lib** generates a `.fixup` section to allow static executables to be relocated at run time, but **−mrelocatable−lib** does not use the smaller stack alignment of **−mrelocatable**. Objects compiled with **−mrelocatable−lib** may be linked with objects compiled with any combination of the **−mrelocatable** options.

−mno−toc
−mtoc
> On System V.4 and embedded PowerPC systems do not (do) assume that register 2 contains a pointer to a global area pointing to the addresses used in the program.

−mlittle
−mlittle−endian
> On System V.4 and embedded PowerPC systems compile code for the processor in little−endian mode. The **−mlittle−endian** option is the same as **−mlittle**.

−mbig
−mbig−endian
> On System V.4 and embedded PowerPC systems compile code for the processor in big−endian mode. The **−mbig−endian** option is the same as **−mbig**.

−mdynamic−no−pic
> On Darwin / macOS systems, compile code so that it is not relocatable, but that its external references are relocatable. The resulting code is suitable for applications, but not shared libraries.

−msingle−pic−base
> Treat the register used for PIC addressing as read−only, rather than loading it in the prologue for each function. The runtime system is responsible for initializing this register with an appropriate value before execution begins.

−mprioritize−restricted−insns=*priority*
> This option controls the priority that is assigned to dispatch−slot restricted instructions during the second scheduling pass. The argument *priority* takes the value **0**, **1**, or **2** to assign no, highest, or second−highest (respectively) priority to dispatch−slot restricted instructions.

−msched−costly−dep=*dependence_type*
> This option controls which dependences are considered costly by the target during instruction scheduling. The argument *dependence_type* takes one of the following values:

> **no** No dependence is costly.

> **all** All dependences are costly.

> **true_store_to_load**
>> A true dependence from store to load is costly.
>
> **store_to_load**
>> Any dependence from store to load is costly.
>
> *number*
>> Any dependence for which the latency is greater than or equal to *number* is costly.

−minsert−sched−nops=*scheme*
> This option controls which NOP insertion scheme is used during the second scheduling pass. The argument *scheme* takes one of the following values:
>
> **no**   Don't insert NOPs.
>
> **pad**
>> Pad with NOPs any dispatch group that has vacant issue slots, according to the scheduler's grouping.
>
> **regroup_exact**
>> Insert NOPs to force costly dependent insns into separate groups. Insert exactly as many NOPs as needed to force an insn to a new group, according to the estimated processor grouping.
>
> *number*
>> Insert NOPs to force costly dependent insns into separate groups. Insert *number* NOPs to force an insn to a new group.

−mcall−sysv
> On System V.4 and embedded PowerPC systems compile code using calling conventions that adhere to the March 1995 draft of the System V Application Binary Interface, PowerPC processor supplement. This is the default unless you configured GCC using **powerpc−*−eabiaix**.

−mcall−sysv−eabi
−mcall−eabi
> Specify both **−mcall−sysv** and **−meabi** options.

−mcall−sysv−noeabi
> Specify both **−mcall−sysv** and **−mno−eabi** options.

−mcall−aixdesc
> On System V.4 and embedded PowerPC systems compile code for the AIX operating system.

−mcall−linux
> On System V.4 and embedded PowerPC systems compile code for the Linux−based GNU system.

−mcall−freebsd
> On System V.4 and embedded PowerPC systems compile code for the FreeBSD operating system.

−mcall−netbsd
> On System V.4 and embedded PowerPC systems compile code for the NetBSD operating system.

−mcall−openbsd
> On System V.4 and embedded PowerPC systems compile code for the OpenBSD operating system.

−mtraceback=*traceback_type*
> Select the type of traceback table. Valid values for *traceback_type* are **full**, **part**, and **no**.

−maix−struct−return
> Return all structures in memory (as specified by the AIX ABI).

−msvr4−struct−return
> Return structures smaller than 8 bytes in registers (as specified by the SVR4 ABI).

−mabi=*abi−type*
> Extend the current ABI with a particular extension, or remove such extension. Valid values are: **altivec**, **no−altivec**, **ibmlongdouble**, **ieeelongdouble**, **elfv1**, **elfv2**, and for AIX: **vec−extabi**,

vec−default.

−mabi=ibmlongdouble
> Change the current ABI to use IBM extended−precision long double. This is not likely to work if your system defaults to using IEEE extended−precision long double. If you change the long double type from IEEE extended−precision, the compiler issues a warning unless you use the **−Wno−psabi** option. Requires **−mlong−double−128** to be enabled.

−mabi=ieeelongdouble
> Change the current ABI to use IEEE extended−precision long double. This is not likely to work if your system defaults to using IBM extended−precision long double. If you change the long double type from IBM extended−precision, the compiler issues a warning unless you use the **−Wno−psabi** option. Requires **−mlong−double−128** to be enabled.

−mabi=elfv1
> Change the current ABI to use the ELFv1 ABI. This is the default ABI for big−endian PowerPC 64−bit Linux. Overriding the default ABI requires special system support and is likely to fail in spectacular ways.

−mabi=elfv2
> Change the current ABI to use the ELFv2 ABI. This is the default ABI for little−endian PowerPC 64−bit Linux. Overriding the default ABI requires special system support and is likely to fail in spectacular ways.

−mgnu−attribute
−mno−gnu−attribute
> Emit .gnu_attribute assembly directives to set tag/value pairs in a .gnu.attributes section that specify ABI variations in function parameters or return values.

−mprototype
−mno−prototype
> On System V.4 and embedded PowerPC systems assume that all calls to variable argument functions are properly prototyped. Otherwise, the compiler must insert an instruction before every non−prototyped call to set or clear bit 6 of the condition code register (CR) to indicate whether floating−point values are passed in the floating−point registers in case the function takes variable arguments. With **−mprototype**, only calls to prototyped variable argument functions set or clear the bit.

−msim
> On embedded PowerPC systems, assume that the startup module is called *sim−crt0.o* and that the standard C libraries are *libsim.a* and *libc.a*. This is the default for **powerpc−*−eabisim** configurations.

−mmvme
> On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libmvme.a* and *libc.a*.

−mads
> On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libads.a* and *libc.a*.

−myellowknife
> On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libyk.a* and *libc.a*.

−mvxworks
> On System V.4 and embedded PowerPC systems, specify that you are compiling for a VxWorks system.

−memb
> On embedded PowerPC systems, set the PPC_EMB bit in the ELF flags header to indicate that **eabi** extended relocations are used.

−meabi
−mno−eabi

On System V.4 and embedded PowerPC systems do (do not) adhere to the Embedded Applications Binary Interface (EABI), which is a set of modifications to the System V.4 specifications. Selecting **−meabi** means that the stack is aligned to an 8−byte boundary, a function `__eabi` is called from `main` to set up the EABI environment, and the **−msdata** option can use both `r2` and `r13` to point to two separate small data areas. Selecting **−mno−eabi** means that the stack is aligned to a 16−byte boundary, no EABI initialization function is called from `main`, and the **−msdata** option only uses `r13` to point to a single small data area. The **−meabi** option is on by default if you configured GCC using one of the **powerpc*−*−eabi*** options.

−msdata=eabi

On System V.4 and embedded PowerPC systems, put small initialized `const` global and static data in the `.sdata2` section, which is pointed to by register `r2`. Put small initialized non−`const` global and static data in the `.sdata` section, which is pointed to by register `r13`. Put small uninitialized global and static data in the `.sbss` section, which is adjacent to the `.sdata` section. The **−msdata=eabi** option is incompatible with the **−mrelocatable** option. The **−msdata=eabi** option also sets the **−memb** option.

−msdata=sysv

On System V.4 and embedded PowerPC systems, put small global and static data in the `.sdata` section, which is pointed to by register `r13`. Put small uninitialized global and static data in the `.sbss` section, which is adjacent to the `.sdata` section. The **−msdata=sysv** option is incompatible with the **−mrelocatable** option.

−msdata=default
−msdata

On System V.4 and embedded PowerPC systems, if **−meabi** is used, compile code the same as **−msdata=eabi**, otherwise compile code the same as **−msdata=sysv**.

−msdata=data

On System V.4 and embedded PowerPC systems, put small global data in the `.sdata` section. Put small uninitialized global data in the `.sbss` section. Do not use register `r13` to address small data however. This is the default behavior unless other **−msdata** options are used.

−msdata=none
−mno−sdata

On embedded PowerPC systems, put all initialized global and static data in the `.data` section, and all uninitialized data in the `.bss` section.

−mreadonly−in−sdata

Put read−only objects in the `.sdata` section as well. This is the default.

−mblock−move−inline−limit=_num_

Inline all block moves (such as calls to `memcpy` or structure copies) less than or equal to _num_ bytes. The minimum value for _num_ is 32 bytes on 32−bit targets and 64 bytes on 64−bit targets. The default value is target−specific.

−mblock−compare−inline−limit=_num_

Generate non−looping inline code for all block compares (such as calls to `memcmp` or structure compares) less than or equal to _num_ bytes. If _num_ is 0, all inline expansion (non−loop and loop) of block compare is disabled. The default value is target−specific.

−mblock−compare−inline−loop−limit=_num_

Generate an inline expansion using loop code for all block compares that are less than or equal to _num_ bytes, but greater than the limit for non−loop inline block compare expansion. If the block length is not constant, at most _num_ bytes will be compared before `memcmp` is called to compare the remainder of the block. The default value is target−specific.

−mstring−compare−inline−limit=*num*

   Compare at most *num* string bytes with inline code. If the difference or end of string is not found at the end of the inline compare a call to `strcmp` or `strncmp` will take care of the rest of the comparison. The default is 64 bytes.

−G *num*

   On embedded PowerPC systems, put global and static items less than or equal to *num* bytes into the small data or BSS sections instead of the normal data or BSS section. By default, *num* is 8. The **−G** *num* switch is also passed to the linker. All modules should be compiled with the same **−G** *num* value.

−mregnames
−mno−regnames

   On System V.4 and embedded PowerPC systems do (do not) emit register names in the assembly language output using symbolic forms.

−mlongcall
−mno−longcall

   By default assume that all calls are far away so that a longer and more expensive calling sequence is required. This is required for calls farther than 32 megabytes (33,554,432 bytes) from the current location. A short call is generated if the compiler knows the call cannot be that far away. This setting can be overridden by the `shortcall` function attribute, or by `#pragma longcall(0)`.

   Some linkers are capable of detecting out−of−range calls and generating glue code on the fly. On these systems, long calls are unnecessary and generate slower code. As of this writing, the AIX linker can do this, as can the GNU linker for PowerPC/64. It is planned to add this feature to the GNU linker for 32−bit PowerPC systems as well.

   On PowerPC64 ELFv2 and 32−bit PowerPC systems with newer GNU linkers, GCC can generate long calls using an inline PLT call sequence (see **−mpltseq**). PowerPC with **−mbss−plt** and PowerPC64 ELFv1 (big−endian) do not support inline PLT calls.

   On Darwin/PPC systems, `#pragma longcall` generates `jbsr callee, L42`, plus a *branch island* (glue code). The two target addresses represent the callee and the branch island. The Darwin/PPC linker prefers the first address and generates a `bl callee` if the PPC `bl` instruction reaches the callee directly; otherwise, the linker generates `bl L42` to call the branch island. The branch island is appended to the body of the calling function; it computes the full 32−bit address of the callee and jumps to it.

   On Mach−O (Darwin) systems, this option directs the compiler emit to the glue for every direct call, and the Darwin linker decides whether to use or discard it.

   In the future, GCC may ignore all longcall specifications when the linker is known to generate glue.

−mpltseq
−mno−pltseq

   Implement (do not implement) −fno−plt and long calls using an inline PLT call sequence that supports lazy linking and long calls to functions in dlopen'd shared libraries. Inline PLT calls are only supported on PowerPC64 ELFv2 and 32−bit PowerPC systems with newer GNU linkers, and are enabled by default if the support is detected when configuring GCC, and, in the case of 32−bit PowerPC, if GCC is configured with **−−enable−secureplt**. **−mpltseq** code and **−mbss−plt** 32−bit PowerPC relocatable objects may not be linked together.

−mtls−markers
−mno−tls−markers

   Mark (do not mark) calls to `__tls_get_addr` with a relocation specifying the function argument. The relocation allows the linker to reliably associate function call with argument setup instructions for TLS optimization, which in turn allows GCC to better schedule the sequence.

−mrecip

−mno−recip

This option enables use of the reciprocal estimate and reciprocal square root estimate instructions with additional Newton−Raphson steps to increase precision instead of doing a divide or square root and divide for floating−point arguments. You should use the **−ffast−math** option when using **−mrecip** (or at least **−funsafe−math−optimizations**, **−ffinite−math−only**, **−freciprocal−math** and **−fno−trapping−math**). Note that while the throughput of the sequence is generally higher than the throughput of the non−reciprocal instruction, the precision of the sequence can be decreased by up to 2 ulp (i.e. the inverse of 1.0 equals 0.99999994) for reciprocal square roots.

−mrecip=*opt*

This option controls which reciprocal estimate instructions may be used. *opt* is a comma−separated list of options, which may be preceded by a ! to invert the option:

all Enable all estimate instructions.

default

Enable the default instructions, equivalent to **−mrecip**.

none

Disable all estimate instructions, equivalent to **−mno−recip**.

div Enable the reciprocal approximation instructions for both single and double precision.

divf

Enable the single−precision reciprocal approximation instructions.

divd

Enable the double−precision reciprocal approximation instructions.

rsqrt

Enable the reciprocal square root approximation instructions for both single and double precision.

rsqrtf

Enable the single−precision reciprocal square root approximation instructions.

rsqrtd

Enable the double−precision reciprocal square root approximation instructions.

So, for example, **−mrecip=all,!rsqrtd** enables all of the reciprocal estimate instructions, except for the `FRSQRTE`, `XSRSQRTEDP`, and `XVRSQRTEDP` instructions which handle the double−precision reciprocal square root calculations.

−mrecip−precision
−mno−recip−precision

Assume (do not assume) that the reciprocal estimate instructions provide higher−precision estimates than is mandated by the PowerPC ABI. Selecting **−mcpu=power6**, **−mcpu=power7** or **−mcpu=power8** automatically selects **−mrecip−precision**. The double−precision square root estimate instructions are not generated by default on low−precision machines, since they do not provide an estimate that converges after three steps.

−mveclibabi=*type*

Specifies the ABI type to use for vectorizing intrinsics using an external library. The only type supported at present is **mass**, which specifies to use IBM's Mathematical Acceleration Subsystem (MASS) libraries for vectorizing intrinsics using external libraries. GCC currently emits calls to `acosd2`, `acosf4`, `acoshd2`, `acoshf4`, `asind2`, `asinf4`, `asinhd2`, `asinhf4`, `atan2d2`, `atan2f4`, `atand2`, `atanf4`, `atanhd2`, `atanhf4`, `cbrtd2`, `cbrtf4`, `cosd2`, `cosf4`, `coshd2`, `coshf4`, `erfcd2`, `erfcf4`, `erfd2`, `erff4`, `exp2d2`, `exp2f4`, `expd2`, `expf4`, `expm1d2`, `expm1f4`, `hypotd2`, `hypotf4`, `lgammad2`, `lgammaf4`, `log10d2`, `log10f4`, `log1pd2`, `log1pf4`, `log2d2`, `log2f4`, `logd2`, `logf4`, `powd2`, `powf4`, `sind2`, `sinf4`, `sinhd2`, `sinhf4`, `sqrtd2`, `sqrtf4`, `tand2`, `tanf4`, `tanhd2`, and `tanhf4` when generating code for power7. Both **−ftree−vectorize** and **−funsafe−math−optimizations** must also be enabled. The MASS libraries must be specified at link time.

−mfriz
−mno−friz

> Generate (do not generate) the `friz` instruction when the **−funsafe−math−optimizations** option is used to optimize rounding of floating−point values to 64−bit integer and back to floating point. The `friz` instruction does not return the same value if the floating−point number is too large to fit in an integer.

−mpointers−to−nested-functions
−mno−pointers−to−nested−functions

> Generate (do not generate) code to load up the static chain register (`r11`) when calling through a pointer on AIX and 64−bit Linux systems where a function pointer points to a 3−word descriptor giving the function address, TOC value to be loaded in register `r2`, and static chain value to be loaded in register `r11`. The **−mpointers−to−nested−functions** is on by default. You cannot call through pointers to nested functions or pointers to functions compiled in other languages that use the static chain if you use **−mno−pointers−to−nested−functions**.

−msave−toc-indirect
−mno−save−toc−indirect

> Generate (do not generate) code to save the TOC value in the reserved stack location in the function prologue if the function calls through a pointer on AIX and 64−bit Linux systems. If the TOC value is not saved in the prologue, it is saved just before the call through the pointer. The **−mno−save−toc−indirect** option is the default.

−mcompat−align−parm
−mno−compat−align−parm

> Generate (do not generate) code to pass structure parameters with a maximum alignment of 64 bits, for compatibility with older versions of GCC.

> Older versions of GCC (prior to 4.9.0) incorrectly did not align a structure parameter on a 128−bit boundary when that structure contained a member requiring 128−bit alignment. This is corrected in more recent versions of GCC. This option may be used to generate code that is compatible with functions compiled with older versions of GCC.

> The **−mno−compat−align−parm** option is the default.

−mstack−protector−guard=*guard*
−mstack−protector−guard−reg=*reg*
−mstack−protector−guard−offset=*offset*

> Generate stack protection code using canary at *guard*. Supported locations are **global** for global canary or **tls** for per−thread canary in the TLS block (the default with GNU libc version 2.4 or later).

> With the latter choice the options **−mstack−protector−guard−reg=***reg* and **−mstack−protector−guard−offset=***offset* furthermore specify which register to use as base register for reading the canary, and from what offset from that base register. The default for those is as specified in the relevant ABI.

−mpcrel
−mno−pcrel

> Generate (do not generate) pc−relative addressing. The **−mpcrel** option requires that the medium code model (**−mcmodel=medium**) and prefixed addressing (**−mprefixed**) options are enabled.

−mprefixed
−mno−prefixed

> Generate (do not generate) addressing modes using prefixed load and store instructions. The **−mprefixed** option requires that the option **−mcpu=power10** (or later) is enabled.

−mmma
−mno−mma

> Generate (do not generate) the MMA instructions. The **−mma** option requires that the option **−mcpu=power10** (or later) is enabled.

−mrop−protect
−mno−rop−protect
>   Generate (do not generate) ROP protection instructions when the target processor supports them. Currently this option disables the shrink−wrap optimization (**−fshrink−wrap**).

−mprivileged
−mno−privileged
>   Generate (do not generate) code that will run in privileged state.

−mblock−ops−unaligned−vsx
−mno−block−ops−unaligned−vsx
>   Generate (do not generate) unaligned vsx loads and stores for inline expansion of `memcpy` and `memmove`.

−−param rs6000−vect−unroll−limit=
>   The vectorizer will check with target information to determine whether it would be beneficial to unroll the main vectorized loop and by how much. This parameter sets the upper bound of how much the vectorizer will unroll the main loop. The default value is four.

RX Options
>   These command−line options are defined for RX targets:

−m64bit−doubles
−m32bit−doubles
>   Make the `double` data type be 64 bits (**−m64bit−doubles**) or 32 bits (**−m32bit−doubles**) in size. The default is **−m32bit−doubles**. *Note* RX floating−point hardware only works on 32−bit values, which is why the default is **−m32bit−doubles**.

−fpu
−nofpu
>   Enables (**−fpu**) or disables (**−nofpu**) the use of RX floating−point hardware. The default is enabled for the RX600 series and disabled for the RX200 series.
>
>   Floating−point instructions are only generated for 32−bit floating−point values, however, so the FPU hardware is not used for doubles if the **−m64bit−doubles** option is used.
>
>   *Note* If the **−fpu** option is enabled then **−funsafe−math−optimizations** is also enabled automatically. This is because the RX FPU instructions are themselves unsafe.

−mcpu=*name*
>   Selects the type of RX CPU to be targeted. Currently three types are supported, the generic **RX600** and **RX200** series hardware and the specific **RX610** CPU. The default is **RX600**.
>
>   The only difference between **RX600** and **RX610** is that the **RX610** does not support the `MVTIPL` instruction.
>
>   The **RX200** series does not have a hardware floating−point unit and so **−nofpu** is enabled by default when this type is selected.

−mbig−endian−data
−mlittle−endian−data
>   Store data (but not code) in the big−endian format. The default is **−mlittle−endian−data**, i.e. to store data in the little−endian format.

−msmall−data−limit=*N*
>   Specifies the maximum size in bytes of global and static variables which can be placed into the small data area. Using the small data area can lead to smaller and faster code, but the size of area is limited and it is up to the programmer to ensure that the area does not overflow. Also when the small data area is used one of the RX's registers (usually `r13`) is reserved for use pointing to this area, so it is no longer available for use by the compiler. This could result in slower and/or larger code if variables are pushed onto the stack instead of being held in this register.

Note, common variables (variables that have not been initialized) and constants are not placed into the small data area as they are assigned to other sections in the output executable.

The default value is zero, which disables this feature. Note, this feature is not enabled by default with higher optimization levels (**−O2** etc) because of the potentially detrimental effects of reserving a register. It is up to the programmer to experiment and discover whether this feature is of benefit to their program. See the description of the **−mpid** option for a description of how the actual register to hold the small data area pointer is chosen.

−msim
−mno−sim
> Use the simulator runtime. The default is to use the libgloss board−specific runtime.

−mas100−syntax
−mno−as100−syntax
> When generating assembler output use a syntax that is compatible with Renesas's AS100 assembler. This syntax can also be handled by the GAS assembler, but it has some restrictions so it is not generated by default.

−mmax−constant−size=_N_
> Specifies the maximum size, in bytes, of a constant that can be used as an operand in a RX instruction. Although the RX instruction set does allow constants of up to 4 bytes in length to be used in instructions, a longer value equates to a longer instruction. Thus in some circumstances it can be beneficial to restrict the size of constants that are used in instructions. Constants that are too big are instead placed into a constant pool and referenced via register indirection.
>
> The value _N_ can be between 0 and 4. A value of 0 (the default) or 4 means that constants of any size are allowed.

−mrelax
> Enable linker relaxation. Linker relaxation is a process whereby the linker attempts to reduce the size of a program by finding shorter versions of various instructions. Disabled by default.

−mint−register=_N_
> Specify the number of registers to reserve for fast interrupt handler functions. The value _N_ can be between 0 and 4. A value of 1 means that register `r13` is reserved for the exclusive use of fast interrupt handlers. A value of 2 reserves `r13` and `r12`. A value of 3 reserves `r13`, `r12` and `r11`, and a value of 4 reserves `r13` through `r10`. A value of 0, the default, does not reserve any registers.

−msave−acc−in−interrupts
> Specifies that interrupt handler functions should preserve the accumulator register. This is only necessary if normal code might use the accumulator register, for example because it performs 64−bit multiplications. The default is to ignore the accumulator as this makes the interrupt handlers faster.

−mpid
−mno−pid
> Enables the generation of position independent data. When enabled any access to constant data is done via an offset from a base address held in a register. This allows the location of constant data to be determined at run time without requiring the executable to be relocated, which is a benefit to embedded applications with tight memory constraints. Data that can be modified is not affected by this option.
>
> Note, using this feature reserves a register, usually `r13`, for the constant data base address. This can result in slower and/or larger code, especially in complicated functions.
>
> The actual register chosen to hold the constant data base address depends upon whether the **−msmall−data−limit** and/or the **−mint−register** command−line options are enabled. Starting with register `r13` and proceeding downwards, registers are allocated first to satisfy the requirements of **−mint−register**, then **−mpid** and finally **−msmall−data−limit**. Thus it is possible for the small data area register to be `r8` if both **−mint−register=4** and **−mpid** are specified on the command line.

By default this feature is not enabled. The default can be restored via the **−mno−pid** command−line option.

−mno−warn−multiple−fast−interrupts
−mwarn−multiple−fast−interrupts
Prevents GCC from issuing a warning message if it finds more than one fast interrupt handler when it is compiling a file. The default is to issue a warning for each extra fast interrupt handler found, as the RX only supports one such interrupt.

−mallow−string−insns
−mno−allow−string−insns
Enables or disables the use of the string manipulation instructions SMOVF, SCMPU, SMOVB, SMOVU, SUNTIL SWHILE and also the RMPA instruction. These instructions may prefetch data, which is not safe to do if accessing an I/O register. (See section 12.2.7 of the RX62N Group User's Manual for more information).

The default is to allow these instructions, but it is not possible for GCC to reliably detect all circumstances where a string instruction might be used to access an I/O register, so their use cannot be disabled automatically. Instead it is reliant upon the programmer to use the **−mno−allow−string−insns** option if their program accesses I/O space.

When the instructions are enabled GCC defines the C preprocessor symbol __RX_ALLOW_STRING_INSNS__, otherwise it defines the symbol __RX_DISALLOW_STRING_INSNS__.

−mjsr
−mno−jsr
Use only (or not only) JSR instructions to access functions. This option can be used when code size exceeds the range of BSR instructions. Note that **−mno−jsr** does not mean to not use JSR but instead means that any type of branch may be used.

Note: The generic GCC command−line option **−ffixed−***reg* has special significance to the RX port when used with the interrupt function attribute. This attribute indicates a function intended to process fast interrupts. GCC ensures that it only uses the registers r10, r11, r12 and/or r13 and only provided that the normal use of the corresponding registers have been restricted via the **−ffixed−***reg* or **−mint−register** command−line options.

S/390 and zSeries Options
These are the **−m** options defined for the S/390 and zSeries architecture.

−mhard−float
−msoft−float
Use (do not use) the hardware floating−point instructions and registers for floating−point operations. When **−msoft−float** is specified, functions in *libgcc.a* are used to perform floating−point operations. When **−mhard−float** is specified, the compiler generates IEEE floating−point instructions. This is the default.

−mhard−dfp
−mno−hard−dfp
Use (do not use) the hardware decimal−floating−point instructions for decimal−floating−point operations. When **−mno−hard−dfp** is specified, functions in *libgcc.a* are used to perform decimal−floating−point operations. When **−mhard−dfp** is specified, the compiler generates decimal−floating−point hardware instructions. This is the default for **−march=z9−ec** or higher.

−mlong−double−64
−mlong−double−128
These switches control the size of long double type. A size of 64 bits makes the long double type equivalent to the double type. This is the default.

−mbackchain
−mno−backchain

Store (do not store) the address of the caller's frame as backchain pointer into the callee's stack frame. A backchain may be needed to allow debugging using tools that do not understand DWARF call frame information. When **−mno−packed−stack** is in effect, the backchain pointer is stored at the bottom of the stack frame; when **−mpacked−stack** is in effect, the backchain is placed into the topmost word of the 96/160 byte register save area.

In general, code compiled with **−mbackchain** is call−compatible with code compiled with **−mno−backchain**; however, use of the backchain for debugging purposes usually requires that the whole binary is built with **−mbackchain**. Note that the combination of **−mbackchain**, **−mpacked−stack** and **−mhard−float** is not supported. In order to build a linux kernel use **−msoft−float**.

The default is to not maintain the backchain.

−mpacked−stack
−mno−packed−stack

Use (do not use) the packed stack layout. When **−mno−packed−stack** is specified, the compiler uses the all fields of the 96/160 byte register save area only for their default purpose; unused fields still take up stack space. When **−mpacked−stack** is specified, register save slots are densely packed at the top of the register save area; unused space is reused for other purposes, allowing for more efficient use of the available stack space. However, when **−mbackchain** is also in effect, the topmost word of the save area is always used to store the backchain, and the return address register is always saved two words below the backchain.

As long as the stack frame backchain is not used, code generated with **−mpacked−stack** is call−compatible with code generated with **−mno−packed−stack**. Note that some non−FSF releases of GCC 2.95 for S/390 or zSeries generated code that uses the stack frame backchain at run time, not just for debugging purposes. Such code is not call−compatible with code compiled with **−mpacked−stack**. Also, note that the combination of **−mbackchain**, **−mpacked−stack** and **−mhard−float** is not supported. In order to build a linux kernel use **−msoft−float**.

The default is to not use the packed stack layout.

−msmall−exec
−mno−small−exec

Generate (or do not generate) code using the `bras` instruction to do subroutine calls. This only works reliably if the total executable size does not exceed 64k. The default is to use the `basr` instruction instead, which does not have this limitation.

−m64
−m31

When **−m31** is specified, generate code compliant to the GNU/Linux for S/390 ABI. When **−m64** is specified, generate code compliant to the GNU/Linux for zSeries ABI. This allows GCC in particular to generate 64−bit instructions. For the **s390** targets, the default is **−m31**, while the **s390x** targets default to **−m64**.

−mzarch
−mesa

When **−mzarch** is specified, generate code using the instructions available on z/Architecture. When **−mesa** is specified, generate code using the instructions available on ESA/390. Note that **−mesa** is not possible with **−m64**. When generating code compliant to the GNU/Linux for S/390 ABI, the default is **−mesa**. When generating code compliant to the GNU/Linux for zSeries ABI, the default is **−mzarch**.

−mhtm
−mno−htm

The **−mhtm** option enables a set of builtins making use of instructions available with the transactional execution facility introduced with the IBM zEnterprise EC12 machine generation **S/390 System z**

Built−in Functions. **−mhtm** is enabled by default when using **−march=zEC12**.

−mvx
−mno−vx

> When **−mvx** is specified, generate code using the instructions available with the vector extension facility introduced with the IBM z13 machine generation. This option changes the ABI for some vector type values with regard to alignment and calling conventions. In case vector type values are being used in an ABI−relevant context a GAS **.gnu_attribute** command will be added to mark the resulting binary with the ABI used. **−mvx** is enabled by default when using **−march=z13**.

−mzvector
−mno−zvector

> The **−mzvector** option enables vector language extensions and builtins using instructions available with the vector extension facility introduced with the IBM z13 machine generation. This option adds support for **vector** to be used as a keyword to define vector type variables and arguments. **vector** is only available when GNU extensions are enabled. It will not be expanded when requesting strict standard compliance e.g. with **−std=c99**. In addition to the GCC low−level builtins **−mzvector** enables a set of builtins added for compatibility with AltiVec−style implementations like Power and Cell. In order to make use of these builtins the header file *vecintrin.h* needs to be included. **−mzvector** is disabled by default.

−mmvcle
−mno−mvcle

> Generate (or do not generate) code using the mvcle instruction to perform block moves. When **−mno−mvcle** is specified, use a mvc loop instead. This is the default unless optimizing for size.

−mdebug
−mno−debug

> Print (or do not print) additional debug information when compiling. The default is to not print debug information.

−march=*cpu−type*

> Generate code that runs on *cpu−type*, which is the name of a system representing a certain processor type. Possible values for *cpu−type* are **z900/arch5**, **z990/arch6**, **z9−109**, **z9−ec/arch7**, **z10/arch8**, **z196/arch9**, **zEC12**, **z13/arch11**, **z14/arch12**, **z15/arch13**, **z16/arch14**, **z17/arch15**, and **native**.

> The default is **−march=z900**.

> Specifying **native** as cpu type can be used to select the best architecture option for the host processor. **−march=native** has no effect if GCC does not recognize the processor.

−mtune=*cpu−type*

> Tune to *cpu−type* everything applicable about the generated code, except for the ABI and the set of available instructions. The list of *cpu−type* values is the same as for **−march**. The default is the value used for **−march**.

−mtpf−trace
−mno−tpf−trace

> Generate code that adds (does not add) in TPF OS specific branches to trace routines in the operating system. This option is off by default, even when compiling for the TPF OS.

−mtpf−trace−skip
−mno−tpf−trace−skip

> Generate code that changes (does not change) the default branch targets enabled by **−mtpf−trace** to point to specialized trace routines providing the ability of selectively skipping function trace entries for the TPF OS. This option is off by default, even when compiling for the TPF OS and specifying **−mtpf−trace**.

−mfused−madd

−mno−fused−madd

Generate code that uses (does not use) the floating−point multiply and accumulate instructions. These instructions are generated by default if hardware floating point is used.

−mwarn−framesize=_framesize_

Emit a warning if the current function exceeds the given frame size. Because this is a compile−time check it doesn't need to be a real problem when the program runs. It is intended to identify functions that most probably cause a stack overflow. It is useful to be used in an environment with limited stack size e.g. the linux kernel.

−mwarn−dynamicstack

Emit a warning if the function calls `alloca` or uses dynamically−sized arrays. This is generally a bad idea with a limited stack size.

−mstack−guard=_stack−guard_

−mstack−size=_stack−size_

If these options are provided the S/390 back end emits additional instructions in the function prologue that trigger a trap if the stack size is _stack−guard_ bytes above the _stack−size_ (remember that the stack on S/390 grows downward). If the _stack−guard_ option is omitted the smallest power of 2 larger than the frame size of the compiled function is chosen. These options are intended to be used to help debugging stack overflow problems. The additionally emitted code causes only little overhead and hence can also be used in production−like systems without greater performance degradation. The given values have to be exact powers of 2 and _stack−size_ has to be greater than _stack−guard_ without exceeding 64k. In order to be efficient the extra code makes the assumption that the stack starts at an address aligned to the value given by _stack−size_. The _stack−guard_ option can only be used in conjunction with _stack−size_.

−mhotpatch=_pre−halfwords_**,**_post−halfwords_

If the hotpatch option is enabled, a "hot−patching" function prologue is generated for all functions in the compilation unit. The funtion label is prepended with the given number of two−byte NOP instructions (_pre−halfwords_, maximum 1000000). After the label, 2 * _post−halfwords_ bytes are appended, using the largest NOP like instructions the architecture allows (maximum 1000000).

If both arguments are zero, hotpatching is disabled.

This option can be overridden for individual functions with the `hotpatch` attribute.

SH Options

These **−m** options are defined for the SH implementations:

−m1

Generate code for the SH1.

−m2

Generate code for the SH2.

−m2e

Generate code for the SH2e.

−m2a−nofpu

Generate code for the SH2a without FPU, or for a SH2a−FPU in such a way that the floating−point unit is not used.

−m2a−single−only

Generate code for the SH2a−FPU, in such a way that no double−precision floating−point operations are used.

−m2a−single

Generate code for the SH2a−FPU assuming the floating−point unit is in single−precision mode by default.

−m2a

> Generate code for the SH2a−FPU assuming the floating−point unit is in double−precision mode by default.

−m3

> Generate code for the SH3.

−m3e

> Generate code for the SH3e.

−m4−nofpu

> Generate code for the SH4 without a floating−point unit.

−m4−single−only

> Generate code for the SH4 with a floating−point unit that only supports single−precision arithmetic.

−m4−single

> Generate code for the SH4 assuming the floating−point unit is in single−precision mode by default.

−m4

> Generate code for the SH4.

−m4−100

> Generate code for SH4−100.

−m4−100−nofpu

> Generate code for SH4−100 in such a way that the floating−point unit is not used.

−m4−100−single

> Generate code for SH4−100 assuming the floating−point unit is in single−precision mode by default.

−m4−100−single−only

> Generate code for SH4−100 in such a way that no double−precision floating−point operations are used.

−m4−200

> Generate code for SH4−200.

−m4−200−nofpu

> Generate code for SH4−200 without in such a way that the floating−point unit is not used.

−m4−200−single

> Generate code for SH4−200 assuming the floating−point unit is in single−precision mode by default.

−m4−200−single−only

> Generate code for SH4−200 in such a way that no double−precision floating−point operations are used.

−m4−300

> Generate code for SH4−300.

−m4−300−nofpu

> Generate code for SH4−300 without in such a way that the floating−point unit is not used.

−m4−300−single

> Generate code for SH4−300 in such a way that no double−precision floating−point operations are used.

−m4−300−single−only

> Generate code for SH4−300 in such a way that no double−precision floating−point operations are used.

−m4−340

> Generate code for SH4−340 (no MMU, no FPU).

−m4−500

> Generate code for SH4−500 (no FPU). Passes **−isa=sh4−nofpu** to the assembler.

−m4a−nofpu

> Generate code for the SH4al−dsp, or for a SH4a in such a way that the floating−point unit is not used.

−m4a−single−only

> Generate code for the SH4a, in such a way that no double−precision floating−point operations are used.

−m4a−single

> Generate code for the SH4a assuming the floating−point unit is in single−precision mode by default.

−m4a

> Generate code for the SH4a.

−m4al

> Same as **−m4a−nofpu**, except that it implicitly passes **−dsp** to the assembler. GCC doesn't generate any DSP instructions at the moment.

−mb

> Compile code for the processor in big−endian mode.

−ml

> Compile code for the processor in little−endian mode.

−mdalign

> Align doubles at 64−bit boundaries. Note that this changes the calling conventions, and thus some functions from the standard C library do not work unless you recompile it first with **−mdalign**.

−mrelax

> Shorten some address references at link time, when possible; uses the linker option **−relax**.

−mbigtable

> Use 32−bit offsets in `switch` tables. The default is to use 16−bit offsets.

−mbitops

> Enable the use of bit manipulation instructions on SH2A.

−mfmovd

> Enable the use of the instruction `fmovd`. Check **−mdalign** for alignment constraints.

−mrenesas

> Comply with the calling conventions defined by Renesas.

−mno−renesas

> Comply with the calling conventions defined for GCC before the Renesas conventions were available. This option is the default for all targets of the SH toolchain.

−mnomacsave

> Mark the `MAC` register as call−clobbered, even if **−mrenesas** is given.

−mieee
−mno−ieee

> Control the IEEE compliance of floating−point comparisons, which affects the handling of cases where the result of a comparison is unordered. By default **−mieee** is implicitly enabled. If **−ffinite−math−only** is enabled **−mno−ieee** is implicitly set, which results in faster floating−point greater−equal and less−equal comparisons. The implicit settings can be overridden by specifying either **−mieee** or **−mno−ieee**.

−minline−ic_invalidate

> Inline code to invalidate instruction cache entries after setting up nested function trampolines. This option has no effect if **−musermode** is in effect and the selected code generation option (e.g. **−m4**) does not allow the use of the `icbi` instruction. If the selected code generation option does not allow the use of the `icbi` instruction, and **−musermode** is not in effect, the inlined code manipulates the

instruction cache address array directly with an associative write. This not only requires privileged mode at run time, but it also fails if the cache line had been mapped via the TLB and has become unmapped.

−misize

Dump instruction size and location in the assembly code.

−mpadstruct

This option is deprecated. It pads structures to multiple of 4 bytes, which is incompatible with the SH ABI.

−matomic−model=*model*

Sets the model of atomic operations and additional parameters as a comma separated list. For details on the atomic built−in functions see **__atomic Builtins**. The following models and parameters are supported:

none

Disable compiler generated atomic sequences and emit library calls for atomic operations. This is the default if the target is not `sh*-*-linux*`.

soft−gusa

Generate GNU/Linux compatible gUSA software atomic sequences for the atomic built−in functions. The generated atomic sequences require additional support from the interrupt/exception handling code of the system and are only suitable for SH3* and SH4* single−core systems. This option is enabled by default when the target is `sh*-*-linux*` and SH3* or SH4*. When the target is SH4A, this option also partially utilizes the hardware atomic instructions `movli.l` and `movco.l` to create more efficient code, unless **strict** is specified.

soft−tcb

Generate software atomic sequences that use a variable in the thread control block. This is a variation of the gUSA sequences which can also be used on SH1* and SH2* targets. The generated atomic sequences require additional support from the interrupt/exception handling code of the system and are only suitable for single−core systems. When using this model, the **gbr−offset=** parameter has to be specified as well.

soft−imask

Generate software atomic sequences that temporarily disable interrupts by setting `SR.IMASK = 1111`. This model works only when the program runs in privileged mode and is only suitable for single−core systems. Additional support from the interrupt/exception handling code of the system is not required. This model is enabled by default when the target is `sh*-*-linux*` and SH1* or SH2*.

hard−llcs

Generate hardware atomic sequences using the `movli.l` and `movco.l` instructions only. This is only available on SH4A and is suitable for multi−core systems. Since the hardware instructions support only 32 bit atomic variables access to 8 or 16 bit variables is emulated with 32 bit accesses. Code compiled with this option is also compatible with other software atomic model interrupt/exception handling systems if executed on an SH4A system. Additional support from the interrupt/exception handling code of the system is not required for this model.

gbr−offset=

This parameter specifies the offset in bytes of the variable in the thread control block structure that should be used by the generated atomic sequences when the **soft−tcb** model has been selected. For other models this parameter is ignored. The specified value must be an integer multiple of four and in the range 0−1020.

strict

This parameter prevents mixed usage of multiple atomic models, even if they are compatible, and makes the compiler generate atomic sequences of the specified model only.

−mtas

Generate the tas.b opcode for __atomic_test_and_set. Notice that depending on the particular hardware and software configuration this can degrade overall performance due to the operand cache line flushes that are implied by the tas.b instruction. On multi−core SH4A processors the tas.b instruction must be used with caution since it can result in data corruption for certain cache configurations.

−mprefergot

When generating position−independent code, emit function calls using the Global Offset Table instead of the Procedure Linkage Table.

−musermode
−mno−usermode

Don't allow (allow) the compiler generating privileged mode code. Specifying **−musermode** also implies **−mno−inline−ic_invalidate** if the inlined code would not work in user mode. **−musermode** is the default when the target is sh*−*−linux*. If the target is SH1* or SH2* **−musermode** has no effect, since there is no user mode.

−mmultcost=*number*

Set the cost to assume for a multiply insn.

−mdiv=*strategy*

Set the division strategy to be used for integer division operations. *strategy* can be one of:

call−div1

Calls a library function that uses the single−step division instruction div1 to perform the operation. Division by zero calculates an unspecified result and does not trap. This is the default except for SH4, SH2A and SHcompact.

call−fp

Calls a library function that performs the operation in double precision floating point. Division by zero causes a floating−point exception. This is the default for SHcompact with FPU. Specifying this for targets that do not have a double precision FPU defaults to call−div1.

call−table

Calls a library function that uses a lookup table for small divisors and the div1 instruction with case distinction for larger divisors. Division by zero calculates an unspecified result and does not trap. This is the default for SH4. Specifying this for targets that do not have dynamic shift instructions defaults to call−div1.

When a division strategy has not been specified the default strategy is selected based on the current target. For SH2A the default strategy is to use the divs and divu instructions instead of library function calls.

−maccumulate−outgoing−args

Reserve space once for outgoing arguments in the function prologue rather than around each call. Generally beneficial for performance and size. Also needed for unwinding to avoid changing the stack frame around conditional code.

−mdivsi3_libfunc=*name*

Set the name of the library function used for 32−bit signed division to *name*. This only affects the name used in the **call** division strategies, and the compiler still expects the same sets of input/output/clobbered registers as if this option were not present.

−mfixed−range=*register−range*

Generate code treating the given register range as fixed registers. A fixed register is one that the register allocator cannot use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

−mbranch−cost=*num*

Assume *num* to be the cost for a branch instruction. Higher numbers make the compiler try to generate more branch−free code if possible. If not specified the value is selected depending on the

processor type that is being compiled for.

−mzdcbranch
−mno−zdcbranch
> Assume (do not assume) that zero displacement conditional branch instructions bt and bf are fast. If **−mzdcbranch** is specified, the compiler prefers zero displacement branch code sequences. This is enabled by default when generating code for SH4 and SH4A. It can be explicitly disabled by specifying **−mno−zdcbranch**.

−mcbranch−force−delay−slot
> Force the usage of delay slots for conditional branches, which stuffs the delay slot with a nop if a suitable instruction cannot be found. By default this option is disabled. It can be enabled to work around hardware bugs as found in the original SH7055.

−mfused−madd
−mno−fused−madd
> Generate code that uses (does not use) the floating−point multiply and accumulate instructions. These instructions are generated by default if hardware floating point is used. The machine−dependent **−mfused−madd** option is now mapped to the machine−independent **−ffp−contract=fast** option, and **−mno−fused−madd** is mapped to **−ffp−contract=off**.

−mfsca
−mno−fsca
> Allow or disallow the compiler to emit the fsca instruction for sine and cosine approximations. The option **−mfsca** must be used in combination with **−funsafe−math−optimizations**. It is enabled by default when generating code for SH4A. Using **−mno−fsca** disables sine and cosine approximations even if **−funsafe−math−optimizations** is in effect.

−mfsrra
−mno−fsrra
> Allow or disallow the compiler to emit the fsrra instruction for reciprocal square root approximations. The option **−mfsrra** must be used in combination with **−funsafe−math−optimizations** and **−ffinite−math−only**. It is enabled by default when generating code for SH4A. Using **−mno−fsrra** disables reciprocal square root approximations even if **−funsafe−math−optimizations** and **−ffinite−math−only** are in effect.

−mpretend−cmove
> Prefer zero−displacement conditional branches for conditional move instruction patterns. This can result in faster code on the SH4 processor.

−mfdpic
> Generate code using the FDPIC ABI.

Solaris 2 Options
> These **−m** options are supported on Solaris 2:

−mclear−hwcap
> **−mclear−hwcap** tells the compiler to remove the hardware capabilities generated by the Solaris assembler. This is only necessary when object files use ISA extensions not supported by the current machine, but check at runtime whether or not to use them.

−mimpure−text
> **−mimpure−text**, used in addition to **−shared**, tells the compiler to not pass **−z text** to the linker when linking a shared object. Using this option, you can link position−dependent code into a shared object.

> **−mimpure−text** suppresses the "relocations remain against allocatable but non−writable sections" linker error message. However, the necessary relocations trigger copy−on−write, and the shared object is not actually shared across processes. Instead of using **−mimpure−text**, you should compile all source code with **−fpic** or **−fPIC**.

These switches are supported in addition to the above on Solaris 2:

−pthreads

This is a synonym for **−pthread**.

SPARC Options

These **−m** options are supported on the SPARC:

−mno−app−regs

−mapp−regs

Specify **−mapp−regs** to generate output using the global registers 2 through 4, which the SPARC SVR4 ABI reserves for applications. Like the global register 1, each global register 2 through 4 is then treated as an allocable register that is clobbered by function calls. This is the default.

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

−mflat

−mno−flat

With **−mflat**, the compiler does not generate save/restore instructions and uses a "flat" or single register window model. This model is compatible with the regular register window model. The local registers and the input registers (0−−5) are still treated as "call−saved" registers and are saved on the stack as needed.

With **−mno−flat** (the default), the compiler generates save/restore instructions (except for leaf functions). This is the normal operating mode.

−mfpu

−mhard−float

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

−mno−fpu

−msoft−float

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

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

−mhard−quad−float

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

−msoft−quad−float

Generate output containing library calls for quad−word (long double) floating−point instructions. The functions called are those specified in the SPARC ABI. This is the default.

As of this writing, there are no SPARC implementations that have hardware support for the quad−word floating−point instructions. They all invoke a trap handler for one of these instructions, and then the trap handler emulates the effect of the instruction. Because of the trap handler overhead, this is much slower than calling the ABI library routines. Thus the **−msoft−quad−float** option is the default.

−mno−unaligned−doubles

−munaligned−doubles

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

With **−munaligned−doubles**, GCC assumes that doubles have 8−byte alignment only if they are contained in another type, or if they have an absolute address. Otherwise, it assumes they have 4−byte alignment. Specifying this option avoids some rare compatibility problems with code generated by other compilers. It is not the default because it results in a performance loss, especially for

floating−point code.

−muser−mode

−mno−user−mode

Do not generate code that can only run in supervisor mode. This is relevant only for the `casa` instruction emitted for the LEON3 processor. This is the default.

−mfaster−structs

−mno−faster−structs

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

−mstd−struct−return

−mno−std−struct−return

With **−mstd−struct−return**, the compiler generates checking code in functions returning structures or unions to detect size mismatches between the two sides of function calls, as per the 32−bit ABI.

The default is **−mno−std−struct−return**. This option has no effect in 64−bit mode.

−mcpu=_cpu_type_

Set the instruction set, register set, and instruction scheduling parameters for machine type _cpu_type_. Supported values for _cpu_type_ are **v7**, **cypress**, **v8**, **supersparc**, **hypersparc**, **leon**, **leon3**, **leon3v7**, **leon5**, **sparclite**, **f930**, **f934**, **sparclite86x**, **sparclet**, **tsc701**, **v9**, **ultrasparc**, **ultrasparc3**, **niagara**, **niagara2**, **niagara3**, **niagara4**, **niagara7** and **m8**.

Native Solaris and GNU/Linux toolchains also support the value **native**, which selects the best architecture option for the host processor. **−mcpu=native** has no effect if GCC does not recognize the processor.

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

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

v7   cypress, leon3v7

v8   supersparc, hypersparc, leon, leon3, leon5

sparclite
       f930, f934, sparclite86x

sparclet
       tsc701

v9   ultrasparc, ultrasparc3, niagara, niagara2, niagara3, niagara4, niagara7, m8

By default (unless configured otherwise), GCC generates code for the V7 variant of the SPARC architecture. With **−mcpu=cypress**, the compiler additionally optimizes it 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 **−mcpu=v8**, GCC generates code for the V8 variant of the SPARC architecture. 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. With **−mcpu=supersparc**, the compiler additionally optimizes it for the SuperSPARC chip, as used in the SPARCStation 10, 1000 and 2000 series.

With **−mcpu=sparclite**, GCC generates code for the SPARClite variant of the SPARC architecture. This adds the integer multiply, integer divide step and scan (`ffs`) instructions which exist in SPARClite but not in SPARC−V7. With **−mcpu=f930**, the compiler additionally optimizes it for the

Fujitsu MB86930 chip, which is the original SPARClite, with no FPU. With **−mcpu=f934**, the compiler additionally optimizes it for the Fujitsu MB86934 chip, which is the more recent SPARClite with FPU.

With **−mcpu=sparclet**, GCC generates code for the SPARClet variant of the SPARC architecture. This adds the integer multiply, multiply/accumulate, integer divide step and scan (`ffs`) instructions which exist in SPARClet but not in SPARC−V7. With **−mcpu=tsc701**, the compiler additionally optimizes it for the TEMIC SPARClet chip.

With **−mcpu=v9**, GCC generates code for the V9 variant of the SPARC architecture. This adds 64−bit integer and floating−point move instructions, 3 additional floating−point condition code registers and conditional move instructions. With **−mcpu=ultrasparc**, the compiler additionally optimizes it for the Sun UltraSPARC I/II/IIi chips. With **−mcpu=ultrasparc3**, the compiler additionally optimizes it for the Sun UltraSPARC III/III+/IIIi/IIIi+/IV/IV+ chips. With **−mcpu=niagara**, the compiler additionally optimizes it for Sun UltraSPARC T1 chips. With **−mcpu=niagara2**, the compiler additionally optimizes it for Sun UltraSPARC T2 chips. With **−mcpu=niagara3**, the compiler additionally optimizes it for Sun UltraSPARC T3 chips. With **−mcpu=niagara4**, the compiler additionally optimizes it for Sun UltraSPARC T4 chips. With **−mcpu=niagara7**, the compiler additionally optimizes it for Oracle SPARC M7 chips. With **−mcpu=m8**, the compiler additionally optimizes it for Oracle M8 chips.

−mtune=_cpu_type_

Set the instruction scheduling parameters for machine type _cpu_type_, but do not set the instruction set or register set that the option **−mcpu=**_cpu_type_ does.

The same values for **−mcpu=**_cpu_type_ can be used for **−mtune=**_cpu_type_, but the only useful values are those that select a particular CPU implementation. Those are **cypress**, **supersparc**, **hypersparc**, **leon**, **leon3**, **leon3v7**, **leon5**, **f930**, **f934**, **sparclite86x**, **tsc701**, **ultrasparc**, **ultrasparc3**, **niagara**, **niagara2**, **niagara3**, **niagara4**, **niagara7** and **m8**. With native Solaris and GNU/Linux toolchains, **native** can also be used.

−mv8plus
−mno−v8plus

With **−mv8plus**, GCC generates code for the SPARC−V8+ ABI. The difference from the V8 ABI is that the global and out registers are considered 64 bits wide. This is enabled by default on Solaris in 32−bit mode for all SPARC−V9 processors.

−mvis
−mno−vis

With **−mvis**, GCC generates code that takes advantage of the UltraSPARC Visual Instruction Set extensions. The default is **−mno−vis**.

−mvis2
−mno−vis2

With **−mvis2**, GCC generates code that takes advantage of version 2.0 of the UltraSPARC Visual Instruction Set extensions. The default is **−mvis2** when targeting a cpu that supports such instructions, such as UltraSPARC−III and later. Setting **−mvis2** also sets **−mvis**.

−mvis3
−mno−vis3

With **−mvis3**, GCC generates code that takes advantage of version 3.0 of the UltraSPARC Visual Instruction Set extensions. The default is **−mvis3** when targeting a cpu that supports such instructions, such as niagara−3 and later. Setting **−mvis3** also sets **−mvis2** and **−mvis**.

−mvis3b
−mno−vis3b

With **−mvis3b**, GCC generates code that takes advantage of version 3.0 of the UltraSPARC Visual Instruction Set extensions, plus the additional VIS instructions introduced in the Oracle SPARC Architecture 2011. The default is **−mvis3b** when targeting a cpu that supports such instructions, such as niagara−7 and later. Setting **−mvis3b** also sets **−mvis3**, **−mvis2** and **−mvis**.

−mvis4

−mno−vis4

With **−mvis4**, GCC generates code that takes advantage of version 4.0 of the UltraSPARC Visual Instruction Set extensions. The default is **−mvis4** when targeting a cpu that supports such instructions, such as niagara−7 and later. Setting **−mvis4** also sets **−mvis3b**, **−mvis3**, **−mvis2** and **−mvis**.

−mvis4b

−mno−vis4b

With **−mvis4b**, GCC generates code that takes advantage of version 4.0 of the UltraSPARC Visual Instruction Set extensions, plus the additional VIS instructions introduced in the Oracle SPARC Architecture 2017. The default is **−mvis4b** when targeting a cpu that supports such instructions, such as m8 and later. Setting **−mvis4b** also sets **−mvis4**, **−mvis3b**, **−mvis3**, **−mvis2** and **−mvis**.

−mcbcond

−mno−cbcond

With **−mcbcond**, GCC generates code that takes advantage of the UltraSPARC Compare−and−Branch−on−Condition instructions. The default is **−mcbcond** when targeting a CPU that supports such instructions, such as Niagara−4 and later.

−mfmaf

−mno−fmaf

With **−mfmaf**, GCC generates code that takes advantage of the UltraSPARC Fused Multiply−Add Floating−point instructions. The default is **−mfmaf** when targeting a CPU that supports such instructions, such as Niagara−3 and later.

−mfsmuld

−mno−fsmuld

With **−mfsmuld**, GCC generates code that takes advantage of the Floating−point Multiply Single to Double (FsMULd) instruction. The default is **−mfsmuld** when targeting a CPU supporting the architecture versions V8 or V9 with FPU except **−mcpu=leon**.

−mpopc

−mno−popc

With **−mpopc**, GCC generates code that takes advantage of the UltraSPARC Population Count instruction. The default is **−mpopc** when targeting a CPU that supports such an instruction, such as Niagara−2 and later.

−msubxc

−mno−subxc

With **−msubxc**, GCC generates code that takes advantage of the UltraSPARC Subtract−Extended−with−Carry instruction. The default is **−msubxc** when targeting a CPU that supports such an instruction, such as Niagara−7 and later.

−mfix−at697f

Enable the documented workaround for the single erratum of the Atmel AT697F processor (which corresponds to erratum #13 of the AT697E processor).

−mfix−ut699

Enable the documented workarounds for the floating−point errata and the data cache nullify errata of the UT699 processor.

−mfix−ut700

Enable the documented workaround for the back−to−back store errata of the UT699E/UT700 processor.

−mfix−gr712rc

Enable the documented workaround for the back−to−back store errata of the GR712RC processor.

These **−m** options are supported in addition to the above on SPARC−V9 processors in 64−bit environments:

−m32

−m64

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

−mcmodel=*which*

Set the code model to one of

medlow

The Medium/Low code model: 64−bit addresses, programs must be linked in the low 32 bits of memory. Programs can be statically or dynamically linked.

medmid

The Medium/Middle code model: 64−bit addresses, programs must be linked in the low 44 bits of memory, the text and data segments must be less than 2GB in size and the data segment must be located within 2GB of the text segment.

medany

The Medium/Anywhere code model: 64−bit addresses, programs may be linked anywhere in memory, the text and data segments must be less than 2GB in size and the data segment must be located within 2GB of the text segment.

embmedany

The Medium/Anywhere code model for embedded systems: 64−bit addresses, the text and data segments must be less than 2GB in size, both starting anywhere in memory (determined at link time). The global register %g4 points to the base of the data segment. Programs are statically linked and PIC is not supported.

−mmemory−model=*mem−model*

Set the memory model in force on the processor to one of

default

The default memory model for the processor and operating system.

rmo

Relaxed Memory Order

pso Partial Store Order

tso Total Store Order

sc Sequential Consistency

These memory models are formally defined in Appendix D of the SPARC−V9 architecture manual, as set in the processor's PSTATE.MM field.

−mstack−bias

−mno−stack−bias

With **−mstack−bias**, GCC assumes that the stack pointer, and frame pointer if present, are offset by −2047 which must be added back when making stack frame references. This is the default in 64−bit mode. Otherwise, assume no such offset is present.

Options for System V

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

−G Create a shared object. It is recommended that **−symbolic** or **−shared** be used instead.

−Qy

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

−Qn

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

−YP,*dirs*

> Search the directories *dirs*, and no others, for libraries specified with **−l**.

−Ym,*dir*

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

V850 Options

These **−m** options are defined for V850 implementations:

−mlong−calls
−mno−long−calls

> Treat all calls as being far away (near). If calls are assumed to be far away, the compiler always loads the function's address into a register, and calls indirect through the pointer.

−mno−ep
−mep

> Do not optimize (do optimize) basic blocks that use the same index pointer 4 or more times to copy pointer into the ep register, and use the shorter sld and sst instructions. The **−mep** option is on by default if you optimize.

−mno−prolog−function
−mprolog−function

> Do not use (do use) external functions to save and restore registers at the prologue and epilogue of a function. The external functions are slower, but use less code space if more than one function saves the same number of registers. The **−mprolog−function** option is on by default if you optimize.

−mspace

> Try to make the code as small as possible. At present, this just turns on the **−mep** and **−mprolog−function** options.

−mtda=*n*

> Put static or global variables whose size is *n* bytes or less into the tiny data area that register ep points to. The tiny data area can hold up to 256 bytes in total (128 bytes for byte references).

−msda=*n*

> Put static or global variables whose size is *n* bytes or less into the small data area that register gp points to. The small data area can hold up to 64 kilobytes.

−mzda=*n*

> Put static or global variables whose size is *n* bytes or less into the first 32 kilobytes of memory.

−mv850

> Specify that the target processor is the V850.

−mv850e3v5

> Specify that the target processor is the V850E3V5. The preprocessor constant __v850e3v5__ is defined if this option is used.

−mv850e2v4

> Specify that the target processor is the V850E3V5. This is an alias for the **−mv850e3v5** option.

−mv850e2v3

> Specify that the target processor is the V850E2V3. The preprocessor constant __v850e2v3__ is defined if this option is used.

−mv850e2

> Specify that the target processor is the V850E2. The preprocessor constant __v850e2__ is defined if this option is used.

−mv850e1

> Specify that the target processor is the V850E1. The preprocessor constants __v850e1__ and __v850e__ are defined if this option is used.

−mv850es

Specify that the target processor is the V850ES. This is an alias for the **−mv850e1** option.

−mv850e

Specify that the target processor is the V850E. The preprocessor constant `__v850e__` is defined if this option is used.

If neither **−mv850** nor **−mv850e** nor **−mv850e1** nor **−mv850e2** nor **−mv850e2v3** nor **−mv850e3v5** are defined then a default target processor is chosen and the relevant **`__v850*__`** preprocessor constant is defined.

The preprocessor constants `__v850` and `__v851__` are always defined, regardless of which processor variant is the target.

−mdisable−callt
−mno−disable−callt

This option suppresses generation of the `CALLT` instruction for the v850e, v850e1, v850e2, v850e2v3 and v850e3v5 flavors of the v850 architecture.

This option is enabled by default when the RH850 ABI is in use (see **−mrh850−abi**), and disabled by default when the GCC ABI is in use. If `CALLT` instructions are being generated then the C preprocessor symbol `__V850_CALLT__` is defined.

−mrelax
−mno−relax

Pass on (or do not pass on) the **−mrelax** command−line option to the assembler.

−mlong−jumps
−mno−long−jumps

Disable (or re−enable) the generation of PC−relative jump instructions.

−msoft−float
−mhard−float

Disable (or re−enable) the generation of hardware floating point instructions. This option is only significant when the target architecture is **V850E2V3** or higher. If hardware floating point instructions are being generated then the C preprocessor symbol `__FPU_OK__` is defined, otherwise the symbol `__NO_FPU__` is defined.

−mloop

Enables the use of the e3v5 LOOP instruction. The use of this instruction is not enabled by default when the e3v5 architecture is selected because its use is still experimental.

−mrh850−abi
−mghs

Enables support for the RH850 version of the V850 ABI. This is the default. With this version of the ABI the following rules apply:

*   Integer sized structures and unions are returned via a memory pointer rather than a register.

*   Large structures and unions (more than 8 bytes in size) are passed by value.

*   Functions are aligned to 16−bit boundaries.

*   The **−m8byte−align** command−line option is supported.

*   The **−mdisable−callt** command−line option is enabled by default. The **−mno−disable−callt** command−line option is not supported.

When this version of the ABI is enabled the C preprocessor symbol `__V850_RH850_ABI__` is defined.

−mgcc−abi

Enables support for the old GCC version of the V850 ABI. With this version of the ABI the following rules apply:

      *   Integer sized structures and unions are returned in register `r10`.

      *   Large structures and unions (more than 8 bytes in size) are passed by reference.

      *   Functions are aligned to 32−bit boundaries, unless optimizing for size.

      *   The **−m8byte−align** command−line option is not supported.

      *   The **−mdisable−callt** command−line option is supported but not enabled by default.

When this version of the ABI is enabled the C preprocessor symbol `__V850_GCC_ABI__` is defined.

−m8byte−align
−mno−8byte−align
> Enables support for `double` and `long long` types to be aligned on 8−byte boundaries. The default is to restrict the alignment of all objects to at most 4−bytes. When **−m8byte−align** is in effect the C preprocessor symbol `__V850_8BYTE_ALIGN__` is defined.

−mbig−switch
> Generate code suitable for big switch tables. Use this option only if the assembler/linker complain about out of range branches within a switch table.

−mapp−regs
> This option causes r2 and r5 to be used in the code generated by the compiler. This setting is the default.

−mno−app−regs
> This option causes r2 and r5 to be treated as fixed registers.

VAX Options
These **−m** options are defined for the VAX:

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

−mgnu
> Do output those jump instructions, on the assumption that the GNU assembler is being used.

−md
−md−float
> Use the D_floating data format for double−precision floating−point numbers instead of G_floating.

−mg
−mg−float
> Use the G_floating data format for double−precision floating−point numbers instead of D_floating.

−mlra
−mno−lra
> Enable Local Register Allocation. This is still experimental for the VAX, so by default the compiler uses standard reload.

Visium Options
−mdebug
> A program which performs file I/O and is destined to run on an MCM target should be linked with this option. It causes the libraries libc.a and libdebug.a to be linked. The program should be run on the target under the control of the GDB remote debugging stub.

−msim
> A program which performs file I/O and is destined to run on the simulator should be linked with option. This causes libraries libc.a and libsim.a to be linked.

−mfpu

−mhard−float

  Generate code containing floating−point instructions. This is the default.

−mno−fpu

−msoft−float

  Generate code containing library calls for floating−point.

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

−mcpu=*cpu_type*

  Set the instruction set, register set, and instruction scheduling parameters for machine type *cpu_type*. Supported values for *cpu_type* are **mcm**, **gr5** and **gr6**.

  mcm is a synonym of **gr5** present for backward compatibility.

  By default (unless configured otherwise), GCC generates code for the GR5 variant of the Visium architecture.

  With **−mcpu=gr6**, GCC generates code for the GR6 variant of the Visium architecture. The only difference from GR5 code is that the compiler will generate block move instructions.

−mtune=*cpu_type*

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

−msv−mode

  Generate code for the supervisor mode, where there are no restrictions on the access to general registers. This is the default.

−muser−mode

  Generate code for the user mode, where the access to some general registers is forbidden: on the GR5, registers r24 to r31 cannot be accessed in this mode; on the GR6, only registers r29 to r31 are affected.

VMS Options

  These **−m** options are defined for the VMS implementations:

−mvms−return−codes

  Return VMS condition codes from `main`. The default is to return POSIX−style condition (e.g. error) codes.

−mdebug−main=*prefix*

  Flag the first routine whose name starts with *prefix* as the main routine for the debugger.

−mmalloc64

  Default to 64−bit memory allocation routines.

−mpointer−size=*size*

  Set the default size of pointers. Possible options for *size* are **32** or **short** for 32 bit pointers, **64** or **long** for 64 bit pointers, and **no** for supporting only 32 bit pointers. The later option disables `pragma pointer_size`.

VxWorks Options

  The options in this section are defined for all VxWorks targets. Options specific to the target hardware are listed with the other options for that target.

−mrtp

  GCC can generate code for both VxWorks kernels and real time processes (RTPs). This option switches from the former to the latter. It also defines the preprocessor macro `__RTP__`.

−msmp

Select SMP runtimes for linking. Not available on architectures other than PowerPC, nor on VxWorks version 7 or later, in which the selection is part of the VxWorks build configuration and the library paths are the same for either choice.

−non−static

Link an RTP executable against shared libraries rather than static libraries. The options **−static** and **−shared** can also be used for RTPs; **−static** is the default.

−Bstatic
−Bdynamic

These options are passed down to the linker. They are defined for compatibility with Diab.

−Xbind−lazy

Enable lazy binding of function calls. This option is equivalent to **−Wl,−z,now** and is defined for compatibility with Diab.

−Xbind−now

Disable lazy binding of function calls. This option is the default and is defined for compatibility with Diab.

x86 Options

These **−m** options are defined for the x86 family of computers.

−march=_cpu−type_

Generate instructions for the machine type _cpu−type_. In contrast to **−mtune=**_cpu−type_, which merely tunes the generated code for the specified _cpu−type_, **−march=**_cpu−type_ allows GCC to generate code that may not run at all on processors other than the one indicated. Specifying **−march=**_cpu−type_ implies **−mtune=**_cpu−type_, except where noted otherwise.

The choices for _cpu−type_ are:

native

This selects the CPU to generate code for at compilation time by determining the processor type of the compiling machine. Using **−march=native** enables all instruction subsets supported by the local machine (hence the result might not run on different machines). Using **−mtune=native** produces code optimized for the local machine under the constraints of the selected instruction set.

x86−64

A generic CPU with 64−bit extensions, MMX, SSE, SSE2, and FXSR instruction set support.

x86−64−v2
x86−64−v3
x86−64−v4

These choices for _cpu−type_ select the corresponding micro−architecture level from the x86−64 psABI. On ABIs other than the x86−64 psABI they select the same CPU features as the x86−64 psABI documents for the particular micro−architecture level.

Since these _cpu−type_ values do not have a corresponding **−mtune** setting, using **−march** with these values enables generic tuning. Specific tuning can be enabled using the **−mtune=**_other−cpu−type_ option with an appropriate _other−cpu−type_ value.

i386

Original Intel i386 CPU.

i486

Intel i486 CPU. (No scheduling is implemented for this chip.)

i586
pentium

Intel Pentium CPU with no MMX support.

lakemont
Intel Lakemont MCU, based on Intel Pentium CPU.

pentium−mmx
Intel Pentium MMX CPU, based on Pentium core with MMX instruction set support.

pentiumpro
Intel Pentium Pro CPU with no MMX support.

i686
When used with **−march**, the Pentium Pro instruction set is used, so the code runs on all i686 family chips. When used with **−mtune**, it has the same meaning as **generic**.

pentium2
Intel Pentium II CPU, based on Pentium Pro core with MMX and FXSR instruction set support.

pentium3
pentium3m
Intel Pentium III CPU, based on Pentium Pro core with MMX, FXSR and SSE instruction set support.

pentium−m
Intel Pentium M; low−power version of Intel Pentium III CPU with MMX, SSE, SSE2 and FXSR instruction set support. Used by Centrino notebooks.

pentium4
pentium4m
Intel Pentium 4 CPU with MMX, SSE, SSE2 and FXSR instruction set support.

prescott
Improved version of Intel Pentium 4 CPU with MMX, SSE, SSE2, SSE3 and FXSR instruction set support.

nocona
Improved version of Intel Pentium 4 CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3 and FXSR instruction set support.

core2
Intel Core 2 CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, CX16, SAHF and FXSR instruction set support.

nehalem
corei7
Intel Nehalem CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF and FXSR instruction set support.

westmere
Intel Westmere CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR and PCLMUL instruction set support.

sandybridge
corei7−avx
Intel Sandy Bridge CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE and PCLMUL instruction set support.

ivybridge
core−avx−i
Intel Ivy Bridge CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND and F16C instruction set support.

haswell
core−avx2
> Intel Haswell CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE and HLE instruction set support.

broadwell
> Intel Broadwell CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX and PREFETCHW instruction set support.

skylake
> Intel Skylake CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES and SGX instruction set support.

skylake−avx512
> Intel Skylake Server CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, CLWB, AVX512VL, AVX512BW, AVX512DQ and AVX512CD instruction set support.

cascadelake
> Intel Cascade Lake CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, CLWB, AVX512VL, AVX512BW, AVX512DQ, AVX512CD and AVX512VNNI instruction set support.

cannonlake
> Intel Cannon Lake Server CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, PKU, AVX512VBMI, AVX512IFMA and SHA instruction set support.

cooperlake
> Intel Cooper Lake CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, CLWB, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, AVX512VNNI and AVX512BF16 instruction set support.

icelake−client
> Intel Ice Lake Client CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2 , VPCLMULQDQ, AVX512BITALG, RDPID and AVX512VPOPCNTDQ instruction set support.

icelake−server
> Intel Ice Lake Server CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND,

F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2 , VPCLMULQDQ, AVX512BITALG, RDPID, AVX512VPOPCNTDQ, PCONFIG, WBNOINVD and CLWB instruction set support.

tigerlake

Intel Tiger Lake CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2, VPCLMULQDQ, AVX512BITALG, RDPID, AVX512VPOPCNTDQ, MOVDIRI, MOVDIR64B, CLWB, AVX512VP2INTERSECT and KEYLOCKER instruction set support.

rocketlake

Intel Rocket Lake CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2, VPCLMULQDQ, AVX512BITALG, RDPID and AVX512VPOPCNTDQ instruction set support.

alderlake
raptorlake
meteorlake
gracemont

Intel Alder Lake/Raptor Lake/Meteor Lake/Gracemont CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PREFETCHW, PCLMUL, RDRND, XSAVE, XSAVEC, XSAVES, XSAVEOPT, FSGSBASE, PTWRITE, RDPID, SGX, GFNI−SSE, CLWB, MOVDIRI, MOVDIR64B, CLDEMOTE, WAITPKG, ADCX, AVX, AVX2, BMI, BMI2, F16C, FMA, LZCNT, PCONFIG, PKU, VAES, VPCLMULQDQ, SERIALIZE, HRESET, KL, WIDEKL and AVX−VNNI instruction set support.

arrowlake

Intel Arrow Lake CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PREFETCHW, PCLMUL, RDRND, XSAVE, XSAVEC, XSAVES, XSAVEOPT, FSGSBASE, PTWRITE, RDPID, SGX, GFNI−SSE, CLWB, MOVDIRI, MOVDIR64B, CLDEMOTE, WAITPKG, ADCX, AVX, AVX2, BMI, BMI2, F16C, FMA, LZCNT, PCONFIG, PKU, VAES, VPCLMULQDQ, SERIALIZE, HRESET, KL, WIDEKL, AVX−VNNI, UINTR, AVXIFMA, AVXVNNIINT8, AVXNECONVERT and CMPCCXADD instruction set support.

arrowlake−s
lunarlake

Intel Arrow Lake S/Lunar Lake CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PREFETCHW, PCLMUL, RDRND, XSAVE, XSAVEC, XSAVES, XSAVEOPT, FSGSBASE, PTWRITE, RDPID, SGX, GFNI−SSE, CLWB, MOVDIRI, MOVDIR64B, CLDEMOTE, WAITPKG, ADCX, AVX, AVX2, BMI, BMI2, F16C, FMA, LZCNT, PCONFIG, PKU, VAES, VPCLMULQDQ, SERIALIZE, HRESET, KL, WIDEKL, AVX−VNNI, UINTR, AVXIFMA, AVXVNNIINT8, AVXNECONVERT, CMPCCXADD, AVXVNNIINT16, SHA512, SM3 and SM4 instruction set support.

pantherlake

Intel Panther Lake CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PREFETCHW, PCLMUL, RDRND, XSAVE, XSAVEC,

XSAVES, XSAVEOPT, FSGSBASE, PTWRITE, RDPID, SGX, GFNI−SSE, CLWB, MOVDIRI, MOVDIR64B, CLDEMOTE, WAITPKG, ADCX, AVX, AVX2, BMI, BMI2, F16C, FMA, LZCNT, PCONFIG, PKU, VAES, VPCLMULQDQ, SERIALIZE, HRESET, KL, WIDEKL, AVX−VNNI, UINTR, AVXIFMA, AVXVNNIINT8, AVXNECONVERT, CMPCCXADD, AVXVNNIINT16, SHA512, SM3, SM4 and PREFETCHI instruction set support.

sapphirerapids
emeraldrapids
Intel Sapphire Rapids/Emerald Rapids CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2, VPCLMULQDQ, AVX512BITALG, RDPID, AVX512VPOPCNTDQ, PCONFIG, WBNOINVD, CLWB, MOVDIRI, MOVDIR64B, ENQCMD, CLDEMOTE, PTWRITE, WAITPKG, SERIALIZE, TSXLDTRK, UINTR, AMX−BF16, AMX−TILE, AMX−INT8, AVX−VNNI, AVX512−FP16 and AVX512BF16 instruction set support.

graniterapids
Intel Granite Rapids CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2, VPCLMULQDQ, AVX512BITALG, RDPID, AVX512VPOPCNTDQ, PCONFIG, WBNOINVD, CLWB, MOVDIRI, MOVDIR64B, ENQCMD, CLDEMOTE, PTWRITE, WAITPKG, SERIALIZE, TSXLDTRK, UINTR, AMX−BF16, AMX−TILE, AMX−INT8, AVX−VNNI, AVX512−FP16, AVX512BF16, AMX−FP16 and PREFETCHI instruction set support.

graniterapids−d
Intel Granite Rapids D CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2, VPCLMULQDQ, AVX512BITALG, RDPID, AVX512VPOPCNTDQ, PCONFIG, WBNOINVD, CLWB, MOVDIRI, MOVDIR64B, ENQCMD, CLDEMOTE, PTWRITE, WAITPKG, SERIALIZE, TSXLDTRK, UINTR, AMX−BF16, AMX−TILE, AMX−INT8, AVX−VNNI, AVX512FP16, AVX512BF16, AMX−FP16, PREFETCHI and AMX−COMPLEX instruction set support.

diamondrapids
Intel Diamond Rapids CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, AVX, XSAVE, PCLMUL, FSGSBASE, RDRND, F16C, AVX2, BMI, BMI2, LZCNT, FMA, MOVBE, HLE, RDSEED, ADCX, PREFETCHW, AES, CLFLUSHOPT, XSAVEC, XSAVES, SGX, AVX512F, AVX512VL, AVX512BW, AVX512DQ, AVX512CD, PKU, AVX512VBMI, AVX512IFMA, SHA, AVX512VNNI, GFNI, VAES, AVX512VBMI2, VPCLMULQDQ, AVX512BITALG, RDPID, AVX512VPOPCNTDQ, PCONFIG, WBNOINVD, CLWB, MOVDIRI, MOVDIR64B, ENQCMD, CLDEMOTE, PTWRITE, WAITPKG, SERIALIZE, TSXLDTRK, UINTR, AMX−BF16, AMX−TILE, AMX−INT8, AVX−VNNI, AVX512FP16, AVX512BF16, AMX−FP16, PREFETCHI, AMX−COMPLEX, AVX10.1−512, AVX−IFMA, AVX−NE−CONVERT, AVX−VNNI−INT16, AVX−VNNI−INT8, CMPccXADD, SHA512, SM3, SM4, AVX10.2−512, APX_F, AMX−AVX512, AMX−FP8, AMX−TF32, AMX−TRANSPOSE, MOVRS, AMX−MOVRS and USER_MSR instruction set support.

bonnell

atom
Intel Bonnell CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support.

silvermont

slm Intel Silvermont CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, PCLMUL, PREFETCHW and RDRND instruction set support.

goldmont
Intel Goldmont CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, PCLMUL, PREFETCHW, RDRND, AES, SHA, RDSEED, XSAVE, XSAVEC, XSAVES, XSAVEOPT, CLFLUSHOPT and FSGSBASE instruction set support.

goldmont−plus
Intel Goldmont Plus CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, PCLMUL, PREFETCHW, RDRND, AES, SHA, RDSEED, XSAVE, XSAVEC, XSAVES, XSAVEOPT, CLFLUSHOPT, FSGSBASE, PTWRITE, RDPID and SGX instruction set support.

tremont
Intel Tremont CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, CX16, SAHF, FXSR, PCLMUL, PREFETCHW, RDRND, AES, SHA, RDSEED, XSAVE, XSAVEC, XSAVES, XSAVEOPT, CLFLUSHOPT, FSGSBASE, PTWRITE, RDPID, SGX, CLWB, GFNI−SSE, MOVDIRI, MOVDIR64B, CLDEMOTE and WAITPKG instruction set support.

sierraforest
Intel Sierra Forest CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PREFETCHW, PCLMUL, RDRND, XSAVE, XSAVEC, XSAVES, XSAVEOPT, FSGSBASE, PTWRITE, RDPID, SGX, GFNI−SSE, CLWB, MOVDIRI, MOVDIR64B, CLDEMOTE, WAITPKG, ADCX, AVX, AVX2, BMI, BMI2, F16C, FMA, LZCNT, PCONFIG, PKU, VAES, VPCLMULQDQ, SERIALIZE, HRESET, KL, WIDEKL, AVX−VNNI, AVXIFMA, AVXVNNIINT8, AVXNECONVERT, CMPCCXADD, ENQCMD and UINTR instruction set support.

grandridge
Intel Grand Ridge CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PREFETCHW, PCLMUL, RDRND, XSAVE, XSAVEC, XSAVES, XSAVEOPT, FSGSBASE, PTWRITE, RDPID, SGX, GFNI−SSE, CLWB, MOVDIRI, MOVDIR64B, CLDEMOTE, WAITPKG, ADCX, AVX, AVX2, BMI, BMI2, F16C, FMA, LZCNT, PCONFIG, PKU, VAES, VPCLMULQDQ, SERIALIZE, HRESET, KL, WIDEKL, AVX−VNNI, AVXIFMA, AVXVNNIINT8, AVXNECONVERT, CMPCCXADD, ENQCMD and UINTR instruction set support.

clearwaterforest
Intel Clearwater Forest CPU with 64−bit extensions, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PREFETCHW, PCLMUL, RDRND, XSAVE, XSAVEC, XSAVES, XSAVEOPT, FSGSBASE, PTWRITE, RDPID, SGX, GFNI−SSE, CLWB, MOVDIRI, MOVDIR64B, CLDEMOTE, WAITPKG, ADCX, AVX, AVX2, BMI, BMI2, F16C, FMA, LZCNT, PCONFIG, PKU, VAES, VPCLMULQDQ, SERIALIZE, HRESET, KL, WIDEKL, AVX−VNNI, ENQCMD, UINTR, AVXIFMA, AVXVNNIINT8, AVXNECONVERT, CMPCCXADD, AVXVNNIINT16, SHA512, SM3, SM4, USER_MSR and PREFETCHI instruction set support.

k6   AMD K6 CPU with MMX instruction set support.

k6−2
k6−3
>   Improved versions of AMD K6 CPU with MMX and 3DNow! instruction set support.

athlon
athlon−tbird
>   AMD Athlon CPU with MMX, 3dNOW!, enhanced 3DNow! and SSE prefetch instructions support.

athlon−4
athlon−xp
athlon−mp
>   Improved AMD Athlon CPU with MMX, 3DNow!, enhanced 3DNow! and full SSE instruction set support.

k8
opteron
athlon64
athlon−fx
>   Processors based on the AMD K8 core with x86−64 instruction set support, including the AMD Opteron, Athlon 64, and Athlon 64 FX processors. (This supersets MMX, SSE, SSE2, 3DNow!, enhanced 3DNow! and 64−bit instruction set extensions.)

k8−sse3
opteron−sse3
athlon64−sse3
>   Improved versions of AMD K8 cores with SSE3 instruction set support.

amdfam10
barcelona
>   CPUs based on AMD Family 10h cores with x86−64 instruction set support. (This supersets MMX, SSE, SSE2, SSE3, SSE4A, 3DNow!, enhanced 3DNow!, ABM and 64−bit instruction set extensions.)

bdver1
>   CPUs based on AMD Family 15h cores with x86−64 instruction set support. (This supersets FMA4, AVX, XOP, LWP, AES, PCLMUL, CX16, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64−bit instruction set extensions.)

bdver2
>   AMD Family 15h core based CPUs with x86−64 instruction set support. (This supersets BMI, TBM, F16C, FMA, FMA4, AVX, XOP, LWP, AES, PCLMUL, CX16, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64−bit instruction set extensions.)

bdver3
>   AMD Family 15h core based CPUs with x86−64 instruction set support. (This supersets BMI, TBM, F16C, FMA, FMA4, FSGSBASE, AVX, XOP, LWP, AES, PCLMUL, CX16, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64−bit instruction set extensions.)

bdver4
>   AMD Family 15h core based CPUs with x86−64 instruction set support. (This supersets BMI, BMI2, TBM, F16C, FMA, FMA4, FSGSBASE, AVX, AVX2, XOP, LWP, AES, PCLMUL, CX16, MOVBE, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64−bit instruction set extensions.)

znver1
>   AMD Family 17h core based CPUs with x86−64 instruction set support. (This supersets BMI, BMI2, F16C, FMA, FSGSBASE, AVX, AVX2, ADCX, RDSEED, MWAITX, SHA, CLZERO, AES, PCLMUL, CX16, MOVBE, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2,

ABM, XSAVEC, XSAVES, CLFLUSHOPT, POPCNT, and 64−bit instruction set extensions.)

znver2

AMD Family 17h core based CPUs with x86−64 instruction set support. (This supersets BMI, BMI2, CLWB, F16C, FMA, FSGSBASE, AVX, AVX2, ADCX, RDSEED, MWAITX, SHA, CLZERO, AES, PCLMUL, CX16, MOVBE, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM, XSAVEC, XSAVES, CLFLUSHOPT, POPCNT, RDPID, WBNOINVD, and 64−bit instruction set extensions.)

znver3

AMD Family 19h core based CPUs with x86−64 instruction set support. (This supersets BMI, BMI2, CLWB, F16C, FMA, FSGSBASE, AVX, AVX2, ADCX, RDSEED, MWAITX, SHA, CLZERO, AES, PCLMUL, CX16, MOVBE, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM, XSAVEC, XSAVES, CLFLUSHOPT, POPCNT, RDPID, WBNOINVD, PKU, VPCLMULQDQ, VAES, and 64−bit instruction set extensions.)

znver4

AMD Family 19h core based CPUs with x86−64 instruction set support. (This supersets BMI, BMI2, CLWB, F16C, FMA, FSGSBASE, AVX, AVX2, ADCX, RDSEED, MWAITX, SHA, CLZERO, AES, PCLMUL, CX16, MOVBE, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM, XSAVEC, XSAVES, CLFLUSHOPT, POPCNT, RDPID, WBNOINVD, PKU, VPCLMULQDQ, VAES, AVX512F, AVX512DQ, AVX512IFMA, AVX512CD, AVX512BW, AVX512VL, AVX512BF16, AVX512VBMI, AVX512VBMI2, AVX512VNNI, AVX512BITALG, AVX512VPOPCNTDQ, GFNI and 64−bit instruction set extensions.)

znver5

AMD Family 1ah core based CPUs with x86−64 instruction set support. (This supersets BMI, BMI2, CLWB, F16C, FMA, FSGSBASE, AVX, AVX2, ADCX, RDSEED, MWAITX, SHA, CLZERO, AES, PCLMUL, CX16, MOVBE, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM, XSAVEC, XSAVES, CLFLUSHOPT, POPCNT, RDPID, WBNOINVD, PKU, VPCLMULQDQ, VAES, AVX512F, AVX512DQ, AVX512IFMA, AVX512CD, AVX512BW, AVX512VL, AVX512BF16, AVX512VBMI, AVX512VBMI2, AVX512VNNI, AVX512BITALG, AVX512VPOPCNTDQ, GFNI, AVXVNNI, MOVDIRI, MOVDIR64B, AVX512VP2INTERSECT, PREFETCHI and 64−bit instruction set extensions.)

btver1

CPUs based on AMD Family 14h cores with x86−64 instruction set support. (This supersets MMX, SSE, SSE2, SSE3, SSSE3, SSE4A, CX16, ABM and 64−bit instruction set extensions.)

btver2

CPUs based on AMD Family 16h cores with x86−64 instruction set support. This includes MOVBE, F16C, BMI, AVX, PCLMUL, AES, SSE4.2, SSE4.1, CX16, ABM, SSE4A, SSSE3, SSE3, SSE2, SSE, MMX and 64−bit instruction set extensions.

winchip−c6

IDT WinChip C6 CPU, dealt in same way as i486 with additional MMX instruction set support.

winchip2

IDT WinChip 2 CPU, dealt in same way as i486 with additional MMX and 3DNow! instruction set support.

c3 VIA C3 CPU with MMX and 3DNow! instruction set support. (No scheduling is implemented for this chip.)

c3−2

VIA C3−2 (Nehemiah/C5XL) CPU with MMX and SSE instruction set support. (No scheduling is implemented for this chip.)

c7 VIA C7 (Esther) CPU with MMX, SSE, SSE2 and SSE3 instruction set support. (No scheduling is implemented for this chip.)

samuel−2

VIA Eden Samuel 2 CPU with MMX and 3DNow! instruction set support. (No scheduling is implemented for this chip.)

nehemiah

VIA Eden Nehemiah CPU with MMX and SSE instruction set support. (No scheduling is implemented for this chip.)

esther

VIA Eden Esther CPU with MMX, SSE, SSE2 and SSE3 instruction set support. (No scheduling is implemented for this chip.)

eden−x2

VIA Eden X2 CPU with x86−64, MMX, SSE, SSE2 and SSE3 instruction set support. (No scheduling is implemented for this chip.)

eden−x4

VIA Eden X4 CPU with x86−64, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX and AVX2 instruction set support. (No scheduling is implemented for this chip.)

nano

Generic VIA Nano CPU with x86−64, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support. (No scheduling is implemented for this chip.)

nano−1000

VIA Nano 1xxx CPU with x86−64, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support. (No scheduling is implemented for this chip.)

nano−2000

VIA Nano 2xxx CPU with x86−64, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support. (No scheduling is implemented for this chip.)

nano−3000

VIA Nano 3xxx CPU with x86−64, MMX, SSE, SSE2, SSE3, SSSE3 and SSE4.1 instruction set support. (No scheduling is implemented for this chip.)

nano−x2

VIA Nano Dual Core CPU with x86−64, MMX, SSE, SSE2, SSE3, SSSE3 and SSE4.1 instruction set support. (No scheduling is implemented for this chip.)

nano−x4

VIA Nano Quad Core CPU with x86−64, MMX, SSE, SSE2, SSE3, SSSE3 and SSE4.1 instruction set support. (No scheduling is implemented for this chip.)

lujiazui

ZHAOXIN lujiazui CPU with x86−64, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, POPCNT, AES, PCLMUL, RDRND, XSAVE, XSAVEOPT, FSGSBASE, CX16, ABM, BMI, BMI2, FXSR, RDSEED instruction set support. While the CPUs do support AVX and F16C, these aren't enabled by −march=lujiazui for performance reasons.

yongfeng

ZHAOXIN yongfeng CPU with x86−64, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, POPCNT, AES, PCLMUL, RDRND, XSAVE, XSAVEOPT, FSGSBASE, CX16, ABM, BMI, BMI2, F16C, FXSR, RDSEED, AVX2, FMA, SHA, LZCNT instruction set support.

shijidadao

ZHAOXIN shijidadao CPU with x86−64, MOVBE, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, POPCNT, AES, PCLMUL, RDRND, XSAVE, XSAVEOPT, FSGSBASE, CX16, ABM, BMI, BMI2, F16C, FXSR, RDSEED, AVX2, FMA, SHA, LZCNT instruction set support.

geode

AMD Geode embedded processor with MMX and 3DNow! instruction set support.

−mtune=_cpu−type_

    Tune to *cpu−type* everything applicable about the generated code, except for the ABI and the set of available instructions. While picking a specific *cpu−type* schedules things appropriately for that particular chip, the compiler does not generate any code that cannot run on the default machine type unless you use a **−march=**_cpu−type_ option. For example, if GCC is configured for i686−pc−linux−gnu then **−mtune=pentium4** generates code that is tuned for Pentium 4 but still runs on i686 machines.

    The choices for *cpu−type* are the same as for **−march**. In addition, **−mtune** supports 2 extra choices for *cpu−type*:

generic

    Produce code optimized for the most common IA32/AMD64/EM64T processors. If you know the CPU on which your code will run, then you should use the corresponding **−mtune** or **−march** option instead of **−mtune=generic**. But, if you do not know exactly what CPU users of your application will have, then you should use this option.

    As new processors are deployed in the marketplace, the behavior of this option will change. Therefore, if you upgrade to a newer version of GCC, code generation controlled by this option will change to reflect the processors that are most common at the time that version of GCC is released.

    There is no **−march=generic** option because **−march** indicates the instruction set the compiler can use, and there is no generic instruction set applicable to all processors. In contrast, **−mtune** indicates the processor (or, in this case, collection of processors) for which the code is optimized.

intel

    Produce code optimized for the most current Intel processors, which are Haswell and Silvermont for this version of GCC. If you know the CPU on which your code will run, then you should use the corresponding **−mtune** or **−march** option instead of **−mtune=intel**. But, if you want your application performs better on both Haswell and Silvermont, then you should use this option.

    As new Intel processors are deployed in the marketplace, the behavior of this option will change. Therefore, if you upgrade to a newer version of GCC, code generation controlled by this option will change to reflect the most current Intel processors at the time that version of GCC is released.

    There is no **−march=intel** option because **−march** indicates the instruction set the compiler can use, and there is no common instruction set applicable to all processors. In contrast, **−mtune** indicates the processor (or, in this case, collection of processors) for which the code is optimized.

−mcpu=_cpu−type_

    A deprecated synonym for **−mtune**.

−mfpmath=_unit_

    Generate floating−point arithmetic for selected unit *unit*. The choices for *unit* are:

387 Use the standard 387 floating−point coprocessor present on the majority of chips and emulated otherwise. Code compiled with this option runs almost everywhere. The temporary results are computed in 80−bit precision instead of the precision specified by the type, resulting in slightly different results compared to most of other chips. See **−ffloat−store** for more detailed description.

    This is the default choice for non−Darwin x86−32 targets.

sse Use scalar floating−point instructions present in the SSE instruction set. This instruction set is supported by Pentium III and newer chips, and in the AMD line by Athlon−4, Athlon XP and Athlon MP chips. The earlier version of the SSE instruction set supports only single−precision arithmetic, thus the double and extended−precision arithmetic are still done using 387. A later version, present only in Pentium 4 and AMD x86−64 chips, supports double−precision arithmetic too.

For the x86−32 compiler, you must use **−march=***cpu−type*, **−msse** or **−msse2** switches to enable SSE extensions and make this option effective. For the x86−64 compiler, these extensions are enabled by default.

The resulting code should be considerably faster in the majority of cases and avoid the numerical instability problems of 387 code, but may break some existing code that expects temporaries to be 80 bits.

This is the default choice for the x86−64 compiler, Darwin x86−32 targets, and the default choice for x86−32 targets with the SSE2 instruction set when **−ffast−math** is enabled.

sse,387
sse+387
both

Attempt to utilize both instruction sets at once. This effectively doubles the amount of available registers, and on chips with separate execution units for 387 and SSE the execution resources too. Use this option with care, as it is still experimental, because the GCC register allocator does not model separate functional units well, resulting in unstable performance.

−masm=*dialect*

Output assembly instructions using selected *dialect*. Also affects which dialect is used for basic `asm` and extended `asm`. Supported choices (in dialect order) are **att** or **intel**. The default is **att**. Darwin does not support **intel**.

−mieee−fp
−mno−ieee−fp

Control whether or not the compiler uses IEEE floating−point comparisons. These correctly handle the case where the result of a comparison is unordered.

−m80387
−mhard−float

Generate output containing 80387 instructions for floating point.

−mno−80387
−msoft−float

Generate output containing library calls for floating point.

Warning: the requisite libraries are not part of GCC. Normally the facilities of the machine's usual C compiler are used, but this cannot be done directly in cross−compilation. You must make your own arrangements to provide suitable library functions for cross−compilation.

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

−mno−fp−ret−in−387

Do not use the FPU registers for return values of functions.

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

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

−mno−fancy−math−387

Some 387 emulators do not support the `sin`, `cos` and `sqrt` instructions for the 387. Specify this option to avoid generating those instructions. This option is overridden when **−march** indicates that the target CPU always has an FPU and so the instruction does not need emulation. These instructions are not generated unless you also use the **−funsafe−math−optimizations** switch.

−malign−double
−mno−align−double

Control whether GCC aligns `double`, `long double`, and `long long` variables on a two−word boundary or a one−word boundary. Aligning `double` variables on a two−word boundary produces

code that runs somewhat faster on a Pentium at the expense of more memory.

On x86−64, **−malign−double** is enabled by default.

Warning: if you use the **−malign−double** switch, structures containing the above types are aligned differently than the published application binary interface specifications for the x86−32 and are not binary compatible with structures in code compiled without that switch.

−m96bit−long−double
−m128bit−long−double
> These switches control the size of `long double` type. The x86−32 application binary interface specifies the size to be 96 bits, so **−m96bit−long−double** is the default in 32−bit mode.
>
> Modern architectures (Pentium and newer) prefer `long double` to be aligned to an 8− or 16−byte boundary. In arrays or structures conforming to the ABI, this is not possible. So specifying **−m128bit−long−double** aligns `long double` to a 16−byte boundary by padding the `long double` with an additional 32−bit zero.
>
> In the x86−64 compiler, **−m128bit−long−double** is the default choice as its ABI specifies that `long double` is aligned on 16−byte boundary.
>
> Notice that neither of these options enable any extra precision over the x87 standard of 80 bits for a `long double`.
>
> **Warning:** if you override the default value for your target ABI, this changes the size of structures and arrays containing `long double` variables, as well as modifying the function calling convention for functions taking `long double`. Hence they are not binary−compatible with code compiled without that switch.

−mlong−double−64
−mlong−double−80
−mlong−double−128
> These switches control the size of `long double` type. A size of 64 bits makes the `long double` type equivalent to the `double` type. This is the default for 32−bit Bionic C library. A size of 128 bits makes the `long double` type equivalent to the `__float128` type. This is the default for 64−bit Bionic C library.
>
> **Warning:** if you override the default value for your target ABI, this changes the size of structures and arrays containing `long double` variables, as well as modifying the function calling convention for functions taking `long double`. Hence they are not binary−compatible with code compiled without that switch.

−malign−data=*type*
> Control how GCC aligns variables. Supported values for *type* are **compat** uses increased alignment value compatible uses GCC 4.8 and earlier, **abi** uses alignment value as specified by the psABI, and **cacheline** uses increased alignment value to match the cache line size. **compat** is the default.

−mlarge−data−threshold=*threshold*
> When **−mcmodel=medium** or **−mcmodel=large** is specified, data objects larger than *threshold* are placed in large data sections. The default is 65535.

−mrtd
> Use a different function−calling convention, in which functions that take a fixed number of arguments return with the `ret num` instruction, which pops their arguments while returning. This saves one instruction in the caller since there is no need to pop the arguments there.
>
> You can specify that an individual function is called with this calling sequence with the function attribute `stdcall`. You can also override the **−mrtd** option by using the function attribute `cdecl`.
>
> **Warning:** this calling convention is incompatible with the one normally used on Unix, so you cannot use it if you need to call libraries compiled with the Unix compiler.

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

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

−mregparm=*num*
> Control how many registers are used to pass integer arguments. By default, no registers are used to pass arguments, and at most 3 registers can be used. You can control this behavior for a specific function by using the function attribute `regparm`.
>
> **Warning:** if you use this switch, and *num* is nonzero, then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

−msseregparm
> Use SSE register passing conventions for float and double arguments and return values. You can control this behavior for a specific function by using the function attribute `sseregparm`.
>
> **Warning:** if you use this switch then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

−mvect8−ret−in−mem
> Return 8−byte vectors in memory instead of MMX registers. This is the default on VxWorks to match the ABI of the Sun Studio compilers until version 12. *Only* use this option if you need to remain compatible with existing code produced by those previous compiler versions or older versions of GCC.

−mpc32
−mpc64
−mpc80
> Set 80387 floating−point precision to 32, 64 or 80 bits. When **−mpc32** is specified, the significands of results of floating−point operations are rounded to 24 bits (single precision); **−mpc64** rounds the significands of results of floating−point operations to 53 bits (double precision) and **−mpc80** rounds the significands of results of floating−point operations to 64 bits (extended double precision), which is the default. When this option is used, floating−point operations in higher precisions are not available to the programmer without setting the FPU control word explicitly.
>
> Setting the rounding of floating−point operations to less than the default 80 bits can speed some programs by 2% or more. Note that some mathematical libraries assume that extended−precision (80−bit) floating−point operations are enabled by default; routines in such libraries could suffer significant loss of accuracy, typically through so−called "catastrophic cancellation", when this option is used to set the precision to less than extended precision.

−mdaz−ftz
> The flush−to−zero (FTZ) and denormals−are−zero (DAZ) flags in the MXCSR register are used to control floating−point calculations.SSE and AVX instructions including scalar and vector instructions could benefit from enabling the FTZ and DAZ flags when **−mdaz−ftz** is specified. Don't set FTZ/DAZ flags when **−mno−daz−ftz** or **−shared** is specified, **−mdaz−ftz** will set FTZ/DAZ flags even with **−shared**.

−mstackrealign
> Realign the stack at entry. On the x86, the **−mstackrealign** option generates an alternate prologue and epilogue that realigns the run−time stack if necessary. This supports mixing legacy codes that keep 4−byte stack alignment with modern codes that keep 16−byte stack alignment for SSE compatibility. See also the attribute `force_align_arg_pointer`, applicable to individual functions.

−mpreferred−stack−boundary=*num*
> Attempt to keep the stack boundary aligned to a 2 raised to *num* byte boundary. If **−mpreferred−stack−boundary** is not specified, the default is 4 (16 bytes or 128 bits).
>
> **Warning:** When generating code for the x86−64 architecture with SSE extensions disabled,

−mpreferred−stack−boundary=3 can be used to keep the stack boundary aligned to 8 byte boundary. Since x86−64 ABI require 16 byte stack alignment, this is ABI incompatible and intended to be used in controlled environment where stack space is important limitation. This option leads to wrong code when functions compiled with 16 byte stack alignment (such as functions from a standard library) are called with misaligned stack. In this case, SSE instructions may lead to misaligned memory access traps. In addition, variable arguments are handled incorrectly for 16 byte aligned objects (including x87 long double and __int128), leading to wrong results. You must build all modules with **−mpreferred−stack−boundary=3**, including any libraries. This includes the system libraries and startup modules.

−mincoming−stack−boundary=*num*

Assume the incoming stack is aligned to a 2 raised to *num* byte boundary. If **−mincoming−stack−boundary** is not specified, the one specified by **−mpreferred−stack−boundary** is used.

On Pentium and Pentium Pro, `double` and `long double` values should be aligned to an 8−byte boundary (see **−malign−double**) or suffer significant run time performance penalties. On Pentium III, the Streaming SIMD Extension (SSE) data type `__m128` may not work properly if it is not 16−byte aligned.

To ensure proper alignment of this values on the stack, the stack boundary must be as aligned as that required by any value stored on the stack. Further, every function must be generated such that it keeps the stack aligned. Thus calling a function compiled with a higher preferred stack boundary from a function compiled with a lower preferred stack boundary most likely misaligns the stack. It is recommended that libraries that use callbacks always use the default setting.

This extra alignment does consume extra stack space, and generally increases code size. Code that is sensitive to stack space usage, such as embedded systems and operating system kernels, may want to reduce the preferred alignment to **−mpreferred−stack−boundary=2**.

−mmmx
−msse
−msse2
−msse3
−mssse3
−msse4
−msse4a
−msse4.1
−msse4.2
−mavx
−mavx2
−mavx512f
−mavx512cd
−mavx512vl
−mavx512bw
−mavx512dq
−mavx512ifma
−mavx512vbmi
−msha
−maes
−mpclmul
−mclflushopt
−mclwb
−mfsgsbase
−mptwrite

−mrdrnd
−mf16c
−mfma
−mpconfig
−mwbnoinvd
−mfma4
−mprfchw
−mrdpid
−mrdseed
−msgx
−mxop
−mlwp
−m3dnow
−m3dnowa
−mpopcnt
−mabm
−madx
−mbmi
−mbmi2
−mlzcnt
−mfxsr
−mxsave
−mxsaveopt
−mxsavec
−mxsaves
−mrtm
−mhle
−mtbm
−mmwaitx
−mclzero
−mpku
−mavx512vbmi2
−mavx512bf16
−mavx512fp16
−mgfni
−mvaes
−mwaitpkg
−mvpclmulqdq
−mavx512bitalg
−mmovdiri
−mmovdir64b
−menqcmd
−muintr
−mtsxldtrk
−mavx512vpopcntdq
−mavx512vp2intersect
−mavx512vnni
−mavxvnni
−mcldemote
−mserialize
−mamx−tile
−mamx−int8

−mamx−bf16
−mhreset
−mkl
−mwidekl
−mavxifma
−mavxvnniint8
−mavxneconvert
−mcmpccxadd
−mamx−fp16
−mprefetchi
−mraoint
−mamx−complex
−mavxvnniint16
−msm3
−msha512
−msm4
−mapxf
−musermsr
−mavx10.1
−mavx10.1−256
−mavx10.1−512
−mavx10.2
−mamx−avx512
−mamx−tf32
−mamx−transpose
−mamx−fp8
−mmovrs
−mamx−movrs

These switches enable the use of instructions in the MMX, SSE, AVX512CD, AVX512VL, AVX512BW, AVX512DQ, AVX512IFMA, AVX512VBMI, SHA, AES, PCLMUL, CLFLUSHOPT, CLWB, FSGSBASE, PTWRITE, RDRND, F16C, FMA, PCONFIG, WBNOINVD, FMA4, PREFETCHW, RDPID, RDSEED, SGX, XOP, LWP, 3DNow!, enhanced 3DNow!, POPCNT, ABM, ADX, BMI, BMI2, LZCNT, FXSR, XSAVE, XSAVEOPT, XSAVEC, XSAVES, RTM, HLE, TBM, MWAITX, CLZERO, PKU, AVX512VBMI2, GFNI, VAES, WAITPKG, VPCLMULQDQ, AVX512BITALG, MOVDIRI, MOVDIR64B, AVX512BF16, ENQCMD, AVX512VPOPCNTDQ, AVX512VNNI, SERIALIZE, UINTR, HRESET, AMXTILE, AMXINT8, AMXBF16, KL, WIDEKL, AVXVNNI, AVX512−FP16, AVXIFMA, AVXVNNIINT8, AVXNECONVERT, CMPCCXADD, AMX−FP16, PREFETCHI, RAOINT, AMX−COMPLEX, AVXVNNIINT16, SM3, SHA512, SM4, APX_F, USER_MSR, AVX10.1, AVX10.2, AMX−AVX512, AMX−TF32, AMX−TRANSPOSE, AMX−FP8, MOVRS, AMX−MOVRS or CLDEMOTE extended instruction sets. Each has a corresponding **−mno−** option to disable use of these instructions.

These extensions are also available as built−in functions: see **x86 Built−in Functions**, for details of the functions enabled and disabled by these switches.

Note that **−msse4** enables both SSE4.1 and SSE4.2 support, while **−mno−sse4** turns off those features; neither form of the option affects SSE4A support, controlled separately by **−msse4a**.

To generate SSE/SSE2 instructions automatically from floating−point code (as opposed to 387 instructions), see **−mfpmath=sse**.

GCC depresses SSEx instructions when **−mavx** is used. Instead, it generates new AVX instructions or AVX equivalence for all SSEx instructions when needed.

These options enable GCC to use these extended instructions in generated code, even without **−mfpmath=sse**. Applications that perform run−time CPU detection must compile separate files for each supported architecture, using the appropriate flags. In particular, the file containing the CPU

detection code should be compiled without these options.

−mdump−tune−features

This option instructs GCC to dump the names of the x86 performance tuning features and default settings. The names can be used in **−mtune−ctrl=***feature−list*.

−mtune−ctrl=*feature−list*

This option is used to do fine grain control of x86 code generation features. *feature−list* is a comma separated list of *feature* names. See also **−mdump−tune−features**. When specified, the *feature* is turned on if it is not preceded with ˆ, otherwise, it is turned off. **−mtune−ctrl=***feature−list* is intended to be used by GCC developers. Using it may lead to code paths not covered by testing and can potentially result in compiler ICEs or runtime errors.

−mno−default

This option instructs GCC to turn off all tunable features. See also **−mtune−ctrl=***feature−list* and **−mdump−tune−features**.

−mcld

This option instructs GCC to emit a `cld` instruction in the prologue of functions that use string instructions. String instructions depend on the DF flag to select between autoincrement or autodecrement mode. While the ABI specifies the DF flag to be cleared on function entry, some operating systems violate this specification by not clearing the DF flag in their exception dispatchers. The exception handler can be invoked with the DF flag set, which leads to wrong direction mode when string instructions are used. This option can be enabled by default on 32−bit x86 targets by configuring GCC with the **−−enable−cld** configure option. Generation of `cld` instructions can be suppressed with the **−mno−cld** compiler option in this case.

−mvzeroupper

This option instructs GCC to emit a `vzeroupper` instruction before a transfer of control flow out of the function to minimize the AVX to SSE transition penalty as well as remove unnecessary `zeroupper` intrinsics.

−mprefer−avx128

This option instructs GCC to use 128−bit AVX instructions instead of 256−bit AVX instructions in the auto−vectorizer.

−mprefer−vector−width=*opt*

This option instructs GCC to use *opt*−bit vector width in instructions instead of default on the selected platform.

−mpartial−vector−fp−math

This option enables GCC to generate floating−point operations that might affect the set of floating−point status flags on partial vectors, where vector elements reside in the low part of the 128−bit SSE register. Unless **−fno−trapping−math** is specified, the compiler guarantees correct behavior by sanitizing all input operands to have zeroes in the unused upper part of the vector register. Note that by using built−in functions or inline assembly with partial vector arguments, NaNs, denormal or invalid values can leak into the upper part of the vector, causing possible performance issues when **−fno−trapping−math** is in effect. These issues can be mitigated by manually sanitizing the upper part of the partial vector argument register or by using **−mdaz−ftz** to set denormals−are−zero (DAZ) flag in the MXCSR register.

This option is enabled by default.

−mmove−max=*bits*

This option instructs GCC to set the maximum number of bits can be moved from memory to memory efficiently to *bits*. The valid *bits* are 128, 256 and 512.

−mstore−max=*bits*

This option instructs GCC to set the maximum number of bits can be stored to memory efficiently to *bits*. The valid *bits* are 128, 256 and 512.

none

No extra limitations applied to GCC other than defined by the selected platform.

128   Prefer 128−bit vector width for instructions.

256   Prefer 256−bit vector width for instructions.

512   Prefer 512−bit vector width for instructions.

−mnoreturn−no−callee−saved−registers

This option optimizes functions with noreturn attribute or _Noreturn specifier by not saving in the function prologue callee−saved registers which are used in the function (except for the BP register). This option can interfere with debugging of the caller of the noreturn function or any function further up in the call stack, so it is not enabled by default.

−mcx16

This option enables GCC to generate CMPXCHG16B instructions in 64−bit code to implement compare−and−exchange operations on 16−byte aligned 128−bit objects. This is useful for atomic updates of data structures exceeding one machine word in size. The compiler uses this instruction to implement **__sync Builtins**. However, for **__atomic Builtins** operating on 128−bit integers, a library call is always used.

−msahf

This option enables generation of SAHF instructions in 64−bit code. Early Intel Pentium 4 CPUs with Intel 64 support, prior to the introduction of Pentium 4 G1 step in December 2005, lacked the LAHF and SAHF instructions which are supported by AMD64. These are load and store instructions, respectively, for certain status flags. In 64−bit mode, the SAHF instruction is used to optimize fmod, drem, and remainder built−in functions; see **Other Builtins** for details.

−mmovbe

This option enables use of the movbe instruction to optimize byte swapping of four and eight byte entities.

−mshstk

The **−mshstk** option enables shadow stack built−in functions from x86 Control−flow Enforcement Technology (CET).

−mcrc32

This option enables built−in functions __builtin_ia32_crc32qi, __builtin_ia32_crc32hi, __builtin_ia32_crc32si and __builtin_ia32_crc32di to generate the crc32 machine instruction.

−mmwait

This option enables built−in functions __builtin_ia32_monitor, and __builtin_ia32_mwait to generate the monitor and mwait machine instructions.

−mrecip

This option enables use of RCPSS and RSQRTSS instructions (and their vectorized variants RCPPS and RSQRTPS) with an additional Newton−Raphson step to increase precision instead of DIVSS and SQRTSS (and their vectorized variants) for single−precision floating−point arguments. These instructions are generated only when **−funsafe−math−optimizations** is enabled together with **−ffinite−math−only** and **−fno−trapping−math**. Note that while the throughput of the sequence is higher than the throughput of the non−reciprocal instruction, the precision of the sequence can be decreased by up to 2 ulp (i.e. the inverse of 1.0 equals 0.99999994).

Note that GCC implements 1.0f/sqrtf(*x*) in terms of RSQRTSS (or RSQRTPS) already with **−ffast−math** (or the above option combination), and doesn't need **−mrecip**.

Also note that GCC emits the above sequence with additional Newton−Raphson step for vectorized single−float division and vectorized sqrtf(*x*) already with **−ffast−math** (or the above option combination), and doesn't need **−mrecip**.

−mrecip=*opt*

This option controls which reciprocal estimate instructions may be used. *opt* is a comma−separated list of options, which may be preceded by a **!** to invert the option:

all   Enable all estimate instructions.

default
Enable the default instructions, equivalent to **−mrecip**.

none
Disable all estimate instructions, equivalent to **−mno−recip**.

div   Enable the approximation for scalar division.

vec−div
Enable the approximation for vectorized division.

sqrt
Enable the approximation for scalar square root.

vec−sqrt
Enable the approximation for vectorized square root.

So, for example, **−mrecip=all,!sqrt** enables all of the reciprocal approximations, except for square root.

−mveclibabi=*type*

Specifies the ABI type to use for vectorizing intrinsics using an external library. Supported values for *type* are **svml** for the Intel short vector math library, **aocl** for the math library (LibM) from AMD Optimizing CPU Libraries (AOCL) and **acml** for the end−of−life AMD core math library (to which AOCL−LibM is the successor). To use this option, both **−ftree−vectorize** and **−funsafe−math−optimizations** have to be enabled, and an SVML or ACML ABI−compatible library must be specified at link time.

GCC currently emits calls to `vmldExp2`, `vmldLn2`, `vmldLog102`, `vmldPow2`, `vmldTanh2`, `vmldTan2`, `vmldAtan2`, `vmldAtanh2`, `vmldCbrt2`, `vmldSinh2`, `vmldSin2`, `vmldAsinh2`, `vmldAsin2`, `vmldCosh2`, `vmldCos2`, `vmldAcosh2`, `vmldAcos2`, `vmlsExp4`, `vmlsLn4`, `vmlsLog104`, `vmlsPow4`, `vmlsTanh4`, `vmlsTan4`, `vmlsAtan4`, `vmlsAtanh4`, `vmlsCbrt4`, `vmlsSinh4`, `vmlsSin4`, `vmlsAsinh4`, `vmlsAsin4`, `vmlsCosh4`, `vmlsCos4`, `vmlsAcosh4` and `vmlsAcos4` for corresponding function type when **−mveclibabi=svml** is used, `amd_vrs4_acosf`, `amd_vrs16_acosf`, `amd_vrd8_asin`, `amd_vrs4_asinf`, `amd_vrs8_asinf`, `amd_vrs16_asinf`, `amd_vrd2_atan`, `amd_vrd8_atan`, `amd_vrs4_atanf`, `amd_vrs8_atanf`, `amd_vrs16_atanf`, `amd_vrd2_cos`, `amd_vrd4_cos`, `amd_vrd8_cos`, `amd_vrs4_cosf`, `amd_vrs8_cosf`, `amd_vrs16_cosf`, `amd_vrs4_coshf`, `amd_vrs8_coshf`, `amd_vrd2_erf`, `amd_vrd4_erf`, `amd_vrd8_erf`, `amd_vrs4_erff`, `amd_vrs8_erff`, `amd_vrs16_erff`, `amd_vrd2_exp`, `amd_vrd4_exp`, `amd_vrd8_exp`, `amd_vrs4_expf`, `amd_vrs8_expf`, `amd_vrs16_expf`, `amd_vrd2_exp10`, `amd_vrs4_exp10f`, `amd_vrd2_exp2`, `amd_vrd4_exp2`, `amd_vrd8_exp2`, `amd_vrs4_exp2f`, `amd_vrs8_exp2f`, `amd_vrs16_exp2f`, `amd_vrs4_expm1f`, `amd_vrd2_log`, `amd_vrd4_log`, `amd_vrd8_log`, `amd_vrs4_logf`, `amd_vrs8_logf`, `amd_vrs16_logf`, `amd_vrd2_log10`, `amd_vrs4_log10f`, `amd_vrs8_log10f`, `amd_vrs16_log10f`, `amd_vrd2_log1p`, `amd_vrs4_log1pf`, `amd_vrd2_log2`, `amd_vrd4_log2`, `amd_vrd8_log2`, `amd_vrs4_log2f`, `amd_vrs8_log2f`, `amd_vrs16_log2f`, `amd_vrd2_pow`, `amd_vrd4_pow`, `amd_vrd8_pow`, `amd_vrs4_powf`, `amd_vrs8_powf`, `amd_vrs16_powf`, `amd_vrd2_sin`, `amd_vrd4_sin`, `amd_vrd8_sin`, `amd_vrs4_sinf`, `amd_vrs8_sinf`, `amd_vrs16_sinf`, `amd_vrd2_tan`, `amd_vrd4_tan`, `amd_vrd8_tan`, `amd_vrs16_tanf`, `amd_vrs4_tanhf`, `amd_vrs8_tanhf`, `amd_vrs16_tanhf` for the corresponding function type when **−mveclibabi=aocl** is used, and `__vrd2_sin`, `__vrd2_cos`, `__vrd2_exp`, `__vrd2_log`, `__vrd2_log2`, `__vrd2_log10`, `__vrs4_sinf`, `__vrs4_cosf`, `__vrs4_expf`,

`__vrs4_logf`, `__vrs4_log2f`, `__vrs4_log10f` and `__vrs4_powf` for the corresponding function type when **−mveclibabi=acml** is used.

−mabi=*name*

Generate code for the specified calling convention. Permissible values are **sysv** for the ABI used on GNU/Linux and other systems, and **ms** for the Microsoft ABI. The default is to use the Microsoft ABI when targeting Microsoft Windows and the SysV ABI on all other systems. You can control this behavior for specific functions by using the function attributes `ms_abi` and `sysv_abi`.

−mforce−indirect−call

Force all calls to functions to be indirect. This is useful when using Intel Processor Trace where it generates more precise timing information for function calls.

−mmanual−endbr

Insert ENDBR instruction at function entry only via the `cf_check` function attribute. This is useful when used with the option **−fcf−protection=branch** to control ENDBR insertion at the function entry.

−mcet−switch

By default, CET instrumentation is turned off on switch statements that use a jump table and indirect branch track is disabled. Since jump tables are stored in read−only memory, this does not result in a direct loss of hardening. But if the jump table index is attacker−controlled, the indirect jump may not be constrained by CET. This option turns on CET instrumentation to enable indirect branch track for switch statements with jump tables which leads to the jump targets reachable via any indirect jumps.

−mcall−ms2sysv−xlogues

Due to differences in 64−bit ABIs, any Microsoft ABI function that calls a System V ABI function must consider RSI, RDI and XMM6−15 as clobbered. By default, the code for saving and restoring these registers is emitted inline, resulting in fairly lengthy prologues and epilogues. Using **−mcall−ms2sysv−xlogues** emits prologues and epilogues that use stubs in the static portion of libgcc to perform these saves and restores, thus reducing function size at the cost of a few extra instructions.

−mtls−dialect=*type*

Generate code to access thread−local storage using the **gnu** or **gnu2** conventions. **gnu** is the conservative default; **gnu2** is more efficient, but it may add compile− and run−time requirements that cannot be satisfied on all systems.

−mpush−args

−mno−push−args

Use PUSH operations to store outgoing parameters. This method is shorter and usually equally fast as method using SUB/MOV operations and is enabled by default. In some cases disabling it may improve performance because of improved scheduling and reduced dependencies.

−maccumulate−outgoing−args

If enabled, the maximum amount of space required for outgoing arguments is computed in the function prologue. This is faster on most modern CPUs because of reduced dependencies, improved scheduling and reduced stack usage when the preferred stack boundary is not equal to 2. The drawback is a notable increase in code size. This switch implies **−mno−push−args**.

−mthreads

Support thread−safe exception handling on MinGW. Programs that rely on thread−safe exception handling must compile and link all code with the **−mthreads** option. When compiling, **−mthreads** defines **−D_MT**; when linking, it links in a special thread helper library **−lmingwthrd** which cleans up per−thread exception−handling data.

−mms−bitfields

−mno−ms−bitfields

Enable/disable bit−field layout compatible with the native Microsoft Windows compiler.

If `packed` is used on a structure, or if bit−fields are used, it may be that the Microsoft ABI lays out the structure differently than the way GCC normally does. Particularly when moving packed data between functions compiled with GCC and the native Microsoft compiler (either via function call or

as data in a file), it may be necessary to access either format.

This option is enabled by default for Microsoft Windows targets. This behavior can also be controlled locally by use of variable or type attributes. For more information, see **x86 Variable Attributes** and **x86 Type Attributes**.

The Microsoft structure layout algorithm is fairly simple with the exception of the bit−field packing. The padding and alignment of members of structures and whether a bit−field can straddle a storage−unit boundary are determine by these rules:

1. Structure members are stored sequentially in the order in which they are
   declared: the first member has the lowest memory address and the last member the highest.

2. Every data object has an alignment requirement. The alignment requirement
   for all data except structures, unions, and arrays is either the size of the object or the current packing size (specified with either the `aligned` attribute or the `pack` pragma), whichever is less. For structures, unions, and arrays, the alignment requirement is the largest alignment requirement of its members. Every object is allocated an offset so that:

```
offset % alignment_requirement == 0
```

3. Adjacent bit−fields are packed into the same 1−, 2−, or 4−byte allocation
   unit if the integral types are the same size and if the next bit−field fits into the current allocation unit without crossing the boundary imposed by the common alignment requirements of the bit−fields.

MSVC interprets zero−length bit−fields in the following ways:

1. If a zero−length bit−field is inserted between two bit−fields that
   are normally coalesced, the bit−fields are not coalesced.

   For example:

```
struct
 {
   unsigned long bf_1 : 12;
   unsigned long : 0;
   unsigned long bf_2 : 12;
 } t1;
```

   The size of `t1` is 8 bytes with the zero−length bit−field. If the zero−length bit−field were removed, `t1`'s size would be 4 bytes.

2. If a zero−length bit−field is inserted after a bit−field, `foo`, and the
   alignment of the zero−length bit−field is greater than the member that follows it, `bar`, `bar` is aligned as the type of the zero−length bit−field.

   For example:

```
struct
 {
   char foo : 4;
   short : 0;
   char bar;
 } t2;

struct
 {
   char foo : 4;
   short : 0;
   double bar;
 } t3;
```

For `t2`, `bar` is placed at offset 2, rather than offset 1. Accordingly, the size of `t2` is 4. For `t3`, the zero−length bit−field does not affect the alignment of `bar` or, as a result, the size of the structure.

Taking this into account, it is important to note the following:

1. If a zero−length bit−field follows a normal bit−field, the type of the
   zero−length bit−field may affect the alignment of the structure as whole. For example, `t2` has a size of 4 bytes, since the zero−length bit−field follows a normal bit−field, and is of type short.

2. Even if a zero−length bit−field is not followed by a normal bit−field, it may
   still affect the alignment of the structure:

```
struct
 {
    char foo : 6;
    long : 0;
 } t4;
```

Here, `t4` takes up 4 bytes.

3. Zero−length bit−fields following non−bit−field members are ignored:

```
struct
 {
    char foo;
    long : 0;
    char bar;
 } t5;
```

Here, `t5` takes up 2 bytes.

−mno−align−stringops
Do not align the destination of inlined string operations. This switch reduces code size and improves performance in case the destination is already aligned, but GCC doesn't know about it.

−minline−all−stringops
By default GCC inlines string operations only when the destination is known to be aligned to least a 4−byte boundary. This enables more inlining and increases code size, but may improve performance of code that depends on fast `memcpy` and `memset` for short lengths. The option enables inline expansion of `strlen` for all pointer alignments.

−minline−stringops−dynamically
For string operations of unknown size, use run−time checks with inline code for small blocks and a library call for large blocks.

−mstringop−strategy=*alg*
Override the internal decision heuristic for the particular algorithm to use for inlining string operations. The allowed values for *alg* are:

rep_byte
rep_4byte
rep_8byte
Expand using i386 `rep` prefix of the specified size.

byte_loop
loop
unrolled_loop
Expand into an inline loop.

libcall
> Always use a library call.

−mmemcpy−strategy=*strategy*
> Override the internal decision heuristic to decide if `__builtin_memcpy` should be inlined and what inline algorithm to use when the expected size of the copy operation is known. *strategy* is a comma−separated list of *alg*:*max_size*:*dest_align* triplets. *alg* is specified in **−mstringop−strategy**, *max_size* specifies the max byte size with which inline algorithm *alg* is allowed. For the last triplet, the *max_size* must be −1. The *max_size* of the triplets in the list must be specified in increasing order. The minimal byte size for *alg* is 0 for the first triplet and `max_size + 1` of the preceding range.

−mmemset−strategy=*strategy*
> The option is similar to **−mmemcpy−strategy=** except that it is to control `__builtin_memset` expansion.

−momit−leaf−frame−pointer
> Don't keep the frame pointer in a register for leaf functions. This avoids the instructions to save, set up, and restore frame pointers and makes an extra register available in leaf functions. The option **−fomit−leaf−frame−pointer** removes the frame pointer for leaf functions, which might make debugging harder.

−mtls−direct−seg−refs
−mno−tls−direct−seg−refs
> Controls whether TLS variables may be accessed with offsets from the TLS segment register (`%gs` for 32−bit, `%fs` for 64−bit), or whether the thread base pointer must be added. Whether or not this is valid depends on the operating system, and whether it maps the segment to cover the entire TLS area.
>
> For systems that use the GNU C Library, the default is on.

−msse2avx
−mno−sse2avx
> Specify that the assembler should encode SSE instructions with VEX prefix. The option **−mavx** turns this on by default.

−mfentry
−mno−fentry
> If profiling is active (**−pg**), put the profiling counter call before the prologue. Note: On x86 architectures the attribute `ms_hook_prologue` isn't possible at the moment for **−mfentry** and **−pg**.

−mrecord−mcount
−mno−record−mcount
> If profiling is active (**−pg**), generate a `__mcount_loc` section that contains pointers to each profiling call. This is useful for automatically patching and out calls.

−mnop−mcount
−mno−nop−mcount
> If profiling is active (**−pg**), generate the calls to the profiling functions as NOPs. This is useful when they should be patched in later dynamically. This is likely only useful together with **−mrecord−mcount**.

−minstrument−return=*type*
> Instrument function exit in −pg −mfentry instrumented functions with call to specified function. This only instruments true returns ending with ret, but not sibling calls ending with jump. Valid types are *none* to not instrument, *call* to generate a call to `__return__`, or *nop5* to generate a 5 byte nop.

−mrecord−return
−mno−record−return
> Generate a `__return_loc` section pointing to all return instrumentation code.

−mfentry−name=*name*
> Set name of `__fentry__` symbol called at function entry for −pg −mfentry functions.

−mfentry−section=*name*
> Set name of section to record −mrecord−mcount calls (default _ _mcount_loc).

−mskip−rax−setup
−mno−skip−rax−setup
> When generating code for the x86−64 architecture with SSE extensions disabled, **−mskip−rax−setup** can be used to skip setting up RAX register when there are no variable arguments passed in vector registers.
>
> **Warning:** Since RAX register is used to avoid unnecessarily saving vector registers on stack when passing variable arguments, the impacts of this option are callees may waste some stack space, misbehave or jump to a random location. GCC 4.4 or newer don't have those issues, regardless the RAX register value.

−m8bit−idiv
−mno−8bit−idiv
> On some processors, like Intel Atom, 8−bit unsigned integer divide is much faster than 32−bit/64−bit integer divide. This option generates a run−time check. If both dividend and divisor are within range of 0 to 255, 8−bit unsigned integer divide is used instead of 32−bit/64−bit integer divide.

−mavx256−split−unaligned−load
−mavx256−split−unaligned−store
> Split 32−byte AVX unaligned load and store.

−mstack−protector−guard=*guard*
−mstack−protector−guard−reg=*reg*
−mstack−protector−guard−offset=*offset*
−mstack−protector−guard−symbol=*symbol*
> Generate stack protection code using canary at *guard*. Supported locations are **global** for global canary or **tls** for per−thread canary in the TLS block (the default). This option has effect only when **−fstack−protector** or **−fstack−protector−all** is specified.
>
> With the latter choice the options **−mstack−protector−guard−reg=***reg* and **−mstack−protector−guard−offset=***offset* furthermore specify which segment register (`%fs` or `%gs`) to use as base register for reading the canary, and from what offset from that base register. The default for those is as specified in the relevant ABI.
>
> **−mstack−protector−guard−symbol=***symbol* overrides the offset with a symbol reference to a canary in the TLS block.

−mgeneral−regs−only
> Generate code that uses only the general−purpose registers. This prevents the compiler from using floating−point, vector, mask and bound registers.

−mrelax−cmpxchg−loop
> When emitting a compare−and−swap loop for **_ _sync Builtins** and **_ _atomic Builtins** lacking a native instruction, optimize for the highly contended case by issuing an atomic load before the `CMPXCHG` instruction, and using the `PAUSE` instruction to save CPU power when restarting the loop.

−mindirect−branch=*choice*
> Convert indirect call and jump with *choice*. The default is **keep**, which keeps indirect call and jump unmodified. **thunk** converts indirect call and jump to call and return thunk. **thunk−inline** converts indirect call and jump to inlined call and return thunk. **thunk−extern** converts indirect call and jump to external call and return thunk provided in a separate object file. You can control this behavior for a specific function by using the function attribute `indirect_branch`.
>
> Note that **−mcmodel=large** is incompatible with **−mindirect−branch=thunk** and **−mindirect−branch=thunk−extern** since the thunk function may not be reachable in the large code model.
>
> Note that **−mindirect−branch=thunk−extern** is compatible with **−fcf−protection=branch** since the

external thunk can be made to enable control−flow check.

−mfunction−return=*choice*

    Convert function return with *choice*. The default is **keep**, which keeps function return unmodified. **thunk** converts function return to call and return thunk. **thunk−inline** converts function return to inlined call and return thunk. **thunk−extern** converts function return to external call and return thunk provided in a separate object file. You can control this behavior for a specific function by using the function attribute `function_return`.

    Note that **−mindirect−return=thunk−extern** is compatible with **−fcf−protection=branch** since the external thunk can be made to enable control−flow check.

    Note that **−mcmodel=large** is incompatible with **−mfunction−return=thunk** and **−mfunction−return=thunk−extern** since the thunk function may not be reachable in the large code model.

−mindirect−branch−register

    Force indirect call and jump via register.

−mharden−sls=*choice*

    Generate code to mitigate against straight line speculation (SLS) with *choice*. The default is **none** which disables all SLS hardening. **return** enables SLS hardening for function returns. **indirect−jmp** enables SLS hardening for indirect jumps. **all** enables all SLS hardening.

−mindirect−branch−cs−prefix

    Add CS prefix to call and jmp to indirect thunk with branch target in r8−r15 registers so that the call and jmp instruction length is 6 bytes to allow them to be replaced with **lfence; call *%r8−r15** or **lfence; jmp *%r8−r15** at run−time.

−mapx−inline−asm−use−gpr32

    For inline asm support with APX, by default the EGPR feature was disabled to prevent potential illegal instruction with EGPR occurs. To invoke egpr usage in inline asm, use new compiler option −mapx−inline−asm−use−gpr32 and user should ensure the instruction supports EGPR.

−mevex512
−mno−evex512

    Enables/disables 512−bit vector. It will be default on if AVX512F is enabled.

These **−m** switches are supported in addition to the above on x86−64 processors in 64−bit environments.

−m32
−m64
−mx32
−m16
−miamcu

    Generate code for a 16−bit, 32−bit or 64−bit environment. The **−m32** option sets `int`, `long`, and pointer types to 32 bits, and generates code that runs in 32−bit mode.

    The **−m64** option sets `int` to 32 bits and `long` and pointer types to 64 bits, and generates code for the x86−64 architecture. For Darwin only the **−m64** option also turns off the **−fno−pic** and **−mdynamic−no−pic** options.

    The **−mx32** option sets `int`, `long`, and pointer types to 32 bits, and generates code for the x86−64 architecture.

    The **−m16** option is the same as **−m32**, except for that it outputs the `.code16gcc` assembly directive at the beginning of the assembly output so that the binary can run in 16−bit mode.

    The **−miamcu** option generates code which conforms to Intel MCU psABI. It requires the **−m32** option to be turned on.

−mno−red−zone

Do not use a so−called "red zone" for x86−64 code. The red zone is mandated by the x86−64 ABI; it is a 128−byte area beyond the location of the stack pointer that is not modified by signal or interrupt handlers and therefore can be used for temporary data without adjusting the stack pointer. The flag **−mno−red−zone** disables this red zone.

−mcmodel=small

Generate code for the small code model: the program and its symbols must be linked in the lower 2 GB of the address space. Pointers are 64 bits. Programs can be statically or dynamically linked. This is the default code model.

−mcmodel=kernel

Generate code for the kernel code model. The kernel runs in the negative 2 GB of the address space. This model has to be used for Linux kernel code.

−mcmodel=medium

Generate code for the medium model: the program is linked in the lower 2 GB of the address space. Small symbols are also placed there. Symbols with sizes larger than **−mlarge−data−threshold** are put into large data or BSS sections and can be located above 2GB. Programs can be statically or dynamically linked.

−mcmodel=large

Generate code for the large model. This model makes no assumptions about addresses and sizes of sections.

−maddress−mode=long

Generate code for long address mode. This is only supported for 64−bit and x32 environments. It is the default address mode for 64−bit environments.

−maddress−mode=short

Generate code for short address mode. This is only supported for 32−bit and x32 environments. It is the default address mode for 32−bit and x32 environments.

−mneeded
−mno−needed

Emit GNU_PROPERTY_X86_ISA_1_NEEDED GNU property for Linux target to indicate the micro−architecture ISA level required to execute the binary.

−mno−direct−extern−access

Without **−fpic** nor **−fPIC**, always use the GOT pointer to access external symbols. With **−fpic** or **−fPIC**, treat access to protected symbols as local symbols. The default is **−mdirect−extern−access**.

Warning: shared libraries compiled with **−mno−direct−extern−access** and executable compiled with **−mdirect−extern−access** may not be binary compatible if protected symbols are used in shared libraries and executable.

−munroll−only−small−loops

Controls conservative small loop unrolling. It is default enabled by O2, and unrolls loop with less than 4 insns by 1 time. Explicit −f[no−]unroll−[all−]loops would disable this flag to avoid any unintended unrolling behavior that user does not want.

−mlam=_choice_

LAM(linear−address masking) allows special bits in the pointer to be used for metadata. The default is **none**. With **u48**, pointer bits in positions 62:48 can be used for metadata; With **u57**, pointer bits in positions 62:57 can be used for metadata.

x86 Windows Options
Xstormy16 Options

These options are defined for Xstormy16:

−msim

　　Choose startup files and linker script suitable for the simulator.

Xtensa Options

　　These options are supported for Xtensa targets:

−mconst16

−mno−const16

　　Enable or disable use of CONST16 instructions for loading constant values. The CONST16 instruction is currently not a standard option from Tensilica. When enabled, CONST16 instructions are always used in place of the standard L32R instructions. The use of CONST16 is enabled by default only if the L32R instruction is not available.

−mfused−madd

−mno−fused−madd

　　Enable or disable use of fused multiply/add and multiply/subtract instructions in the floating−point option. This has no effect if the floating−point option is not also enabled. Disabling fused multiply/add and multiply/subtract instructions forces the compiler to use separate instructions for the multiply and add/subtract operations. This may be desirable in some cases where strict IEEE 754−compliant results are required: the fused multiply add/subtract instructions do not round the intermediate result, thereby producing results with *more* bits of precision than specified by the IEEE standard. Disabling fused multiply add/subtract instructions also ensures that the program output is not sensitive to the compiler's ability to combine multiply and add/subtract operations.

−mserialize−volatile

−mno−serialize−volatile

　　When this option is enabled, GCC inserts MEMW instructions before volatile memory references to guarantee sequential consistency. The default is **−mserialize−volatile**. Use **−mno−serialize−volatile** to omit the MEMW instructions.

−mforce−no−pic

　　For targets, like GNU/Linux, where all user−mode Xtensa code must be position−independent code (PIC), this option disables PIC for compiling kernel code.

−mtext−section−literals

−mno−text−section−literals

　　These options control the treatment of literal pools. The default is **−mno−text−section−literals**, which places literals in a separate section in the output file. This allows the literal pool to be placed in a data RAM/ROM, and it also allows the linker to combine literal pools from separate object files to remove redundant literals and improve code size. With **−mtext−section−literals**, the literals are interspersed in the text section in order to keep them as close as possible to their references. This may be necessary for large assembly files. Literals for each function are placed right before that function.

−mauto−litpools

−mno−auto−litpools

　　These options control the treatment of literal pools. The default is **−mno−auto−litpools**, which places literals in a separate section in the output file unless **−mtext−section−literals** is used. With **−mauto−litpools** the literals are interspersed in the text section by the assembler. Compiler does not produce explicit .literal directives and loads literals into registers with MOVI instructions instead of L32R to let the assembler do relaxation and place literals as necessary. This option allows assembler to create several literal pools per function and assemble very big functions, which may not be possible with **−mtext−section−literals**.

−mtarget−align

−mno−target−align

　　When this option is enabled, GCC instructs the assembler to automatically align instructions to reduce branch penalties at the expense of some code density. The assembler attempts to widen density instructions to align branch targets and the instructions following call instructions. If there are not enough preceding safe density instructions to align a target, no widening is performed. The default is

−**mtarget**−**align**. These options do not affect the treatment of auto−aligned instructions like `LOOP`, which the assembler always aligns, either by widening density instructions or by inserting NOP instructions.

−**mlongcalls**
−**mno**−**longcalls**

When this option is enabled, GCC instructs the assembler to translate direct calls to indirect calls unless it can determine that the target of a direct call is in the range allowed by the call instruction. This translation typically occurs for calls to functions in other source files. Specifically, the assembler translates a direct `CALL` instruction into an `L32R` followed by a `CALLX` instruction. The default is −**mno**−**longcalls**. This option should be used in programs where the call target can potentially be out of range. This option is implemented in the assembler, not the compiler, so the assembly code generated by GCC still shows direct call instructions−−−look at the disassembled object code to see the actual instructions. Note that the assembler uses an indirect call for every cross−file call, not just those that really are out of range.

−**mabi=***name*

Generate code for the specified ABI. Permissible values are: **call0**, **windowed**. Default ABI is chosen by the Xtensa core configuration.

−**mabi=call0**

When this option is enabled function parameters are passed in registers `a2` through `a7`, registers `a12` through `a15` are caller−saved, and register `a15` may be used as a frame pointer. When this version of the ABI is enabled the C preprocessor symbol `__XTENSA_CALL0_ABI__` is defined.

−**mabi=windowed**

When this option is enabled function parameters are passed in registers `a10` through `a15`, and called function rotates register window by 8 registers on entry so that its arguments are found in registers `a2` through `a7`. Register `a7` may be used as a frame pointer. Register window is rotated 8 registers back upon return. When this version of the ABI is enabled the C preprocessor symbol `__XTENSA_WINDOWED_ABI__` is defined.

−**mextra**−**l32r**−**costs=***n*

Specify an extra cost of instruction RAM/ROM access for `L32R` instructions, in clock cycles. This affects, when optimizing for speed, whether loading a constant from literal pool using `L32R` or synthesizing the constant from a small one with a couple of arithmetic instructions. The default value is 0.

−**mstrict**−**align**
−**mno**−**strict**−**align**

Avoid or allow generating memory accesses that may not be aligned on a natural object boundary as described in the architecture specification. The default is −**mno**−**strict**−**align** for cores that support both unaligned loads and stores in hardware and −**mstrict**−**align** for all other cores.

zSeries Options

These are listed under

ENVIRONMENT

This section describes several environment variables that affect how GCC operates. Some of them work by specifying directories or prefixes to use when searching for various kinds of files. Some are used to specify other aspects of the compilation environment.

Note that you can also specify places to search using options such as −**B**, −**I** and −**L**. These take precedence over places specified using environment variables, which in turn take precedence over those specified by the configuration of GCC.

LANG
LC_CTYPE

LC_MESSAGES
LC_ALL

These environment variables control the way that GCC uses localization information which allows GCC to work with different national conventions. GCC inspects the locale categories **LC_CTYPE** and **LC_MESSAGES** if it has been configured to do so. These locale categories can be set to any value supported by your installation. A typical value is **en_GB.UTF−8** for English in the United Kingdom encoded in UTF−8.

The **LC_CTYPE** environment variable specifies character classification. GCC uses it to determine the character boundaries in a string; this is needed for some multibyte encodings that contain quote and escape characters that are otherwise interpreted as a string end or escape.

The **LC_MESSAGES** environment variable specifies the language to use in diagnostic messages.

If the **LC_ALL** environment variable is set, it overrides the value of **LC_CTYPE** and **LC_MESSAGES**; otherwise, **LC_CTYPE** and **LC_MESSAGES** default to the value of the **LANG** environment variable. If none of these variables are set, GCC defaults to traditional C English behavior.

TMPDIR

If **TMPDIR** is set, it specifies the directory to use for temporary files. GCC uses temporary files to hold the output of one stage of compilation which is to be used as input to the next stage: for example, the output of the preprocessor, which is the input to the compiler proper.

GCC_COMPARE_DEBUG

Setting **GCC_COMPARE_DEBUG** is nearly equivalent to passing **−fcompare−debug** to the compiler driver. See the documentation of this option for more details.

GCC_EXEC_PREFIX

If **GCC_EXEC_PREFIX** is set, it specifies a prefix to use in the names of the subprograms executed by the compiler. No slash is added when this prefix is combined with the name of a subprogram, but you can specify a prefix that ends with a slash if you wish.

If **GCC_EXEC_PREFIX** is not set, GCC attempts to figure out an appropriate prefix to use based on the pathname it is invoked with.

If GCC cannot find the subprogram using the specified prefix, it tries looking in the usual places for the subprogram.

The default value of **GCC_EXEC_PREFIX** is *prefix/lib/gcc/* where *prefix* is the prefix to the installed compiler. In many cases *prefix* is the value of `prefix` when you ran the *configure* script.

Other prefixes specified with **−B** take precedence over this prefix.

This prefix is also used for finding files such as *crt0.o* that are used for linking.

In addition, the prefix is used in an unusual way in finding the directories to search for header files. For each of the standard directories whose name normally begins with **/usr/local/lib/gcc** (more precisely, with the value of **GCC_INCLUDE_DIR**), GCC tries replacing that beginning with the specified prefix to produce an alternate directory name. Thus, with **−Bfoo/**, GCC searches *foo/bar* just before it searches the standard directory */usr/local/lib/bar*. If a standard directory begins with the configured *prefix* then the value of *prefix* is replaced by **GCC_EXEC_PREFIX** when looking for header files.

COMPILER_PATH

The value of **COMPILER_PATH** is a colon−separated list of directories, much like **PATH**. GCC tries the directories thus specified when searching for subprograms, if it cannot find the subprograms using **GCC_EXEC_PREFIX**.

LIBRARY_PATH

The value of **LIBRARY_PATH** is a colon−separated list of directories, much like **PATH**. When configured as a native compiler, GCC tries the directories thus specified when searching for special

linker files, if it cannot find them using **GCC_EXEC_PREFIX**. Linking using GCC also uses these directories when searching for ordinary libraries for the **−l** option (but directories specified with **−L** come first).

LANG

This variable is used to pass locale information to the compiler. One way in which this information is used is to determine the character set to be used when character literals, string literals and comments are parsed in C and C++. When the compiler is configured to allow multibyte characters, the following values for **LANG** are recognized:

C−JIS

Recognize JIS characters.

C−SJIS

Recognize SJIS characters.

C−EUCJP

Recognize EUCJP characters.

If **LANG** is not defined, or if it has some other value, then the compiler uses `mblen` and `mbtowc` as defined by the default locale to recognize and translate multibyte characters.

GCC_EXTRA_DIAGNOSTIC_OUTPUT

If **GCC_EXTRA_DIAGNOSTIC_OUTPUT** is set to one of the following values, then additional text will be emitted to stderr when fix−it hints are emitted. **−fdiagnostics−parseable−fixits** and **−fno−diagnostics−parseable−fixits** take precedence over this environment variable.

fixits−v1

Emit parseable fix−it hints, equivalent to **−fdiagnostics−parseable−fixits**. In particular, columns are expressed as a count of bytes, starting at byte 1 for the initial column.

fixits−v2

As `fixits-v1`, but columns are expressed as display columns, as per **−fdiagnostics−column−unit=display**.

Some additional environment variables affect the behavior of the preprocessor.

CPATH
C_INCLUDE_PATH
CPLUS_INCLUDE_PATH
OBJC_INCLUDE_PATH

Each variable's value is a list of directories separated by a special character, much like **PATH**, in which to look for header files. The special character, PATH_SEPARATOR, is target−dependent and determined at GCC build time. For Microsoft Windows−based targets it is a semicolon, and for almost all other targets it is a colon.

CPATH specifies a list of directories to be searched as if specified with **−I**, but after any paths given with **−I** options on the command line. This environment variable is used regardless of which language is being preprocessed.

The remaining environment variables apply only when preprocessing the particular language indicated. Each specifies a list of directories to be searched as if specified with **−isystem**, but after any paths given with **−isystem** options on the command line.

In all these variables, an empty element instructs the compiler to search its current working directory. Empty elements can appear at the beginning or end of a path. For instance, if the value of **CPATH** is `:/special/include`, that has the same effect as **−I. −I/special/include**.

DEPENDENCIES_OUTPUT

If this variable is set, its value specifies how to output dependencies for Make based on the non−system header files processed by the compiler. System header files are ignored in the dependency output.

The value of **DEPENDENCIES_OUTPUT** can be just a file name, in which case the Make rules are written to that file, guessing the target name from the source file name. Or the value can have the form *file target*, in which case the rules are written to file *file* using *target* as the target name.

In other words, this environment variable is equivalent to combining the options **−MM** and **−MF**, with an optional **−MT** switch too.

SUNPRO_DEPENDENCIES
This variable is the same as **DEPENDENCIES_OUTPUT** (see above), except that system header files are not ignored, so it implies **−M** rather than **−MM**. However, the dependence on the main input file is omitted.

SOURCE_DATE_EPOCH
If this variable is set, its value specifies a UNIX timestamp to be used in replacement of the current date and time in the `__DATE__` and `__TIME__` macros, so that the embedded timestamps become reproducible.

The value of **SOURCE_DATE_EPOCH** must be a UNIX timestamp, defined as the number of seconds (excluding leap seconds) since 01 Jan 1970 00:00:00 represented in ASCII; identical to the output of `date +%s` on GNU/Linux and other systems that support the `%s` extension in the `date` command.

The value should be a known timestamp such as the last modification time of the source or package and it should be set by the build process.

BUGS
For instructions on reporting bugs, see **<http://bugzilla.redhat.com/bugzilla>**.

FOOTNOTES
1. On some systems, **gcc −shared** needs to build supplementary stub code for constructors to work. On multi−libbed systems, **gcc −shared** must select the correct support libraries to link against. Failing to supply the correct flags may lead to subtle defects. Supplying them in cases where they are not necessary is innocuous. **−shared** suppresses the addition of startup code to alter the floating−point environment as done with **−ffast−math**, **−Ofast** or **−funsafe−math−optimizations** on some targets.

SEE ALSO
gpl (7), **gfdl** (7), **fsf−funding** (7), **cpp** (1), **gcov** (1), **as** (1), **ld** (1), **gdb** (1) and the Info entries for *gcc*, *cpp*, *as*, *ld*, *binutils* and *gdb*.

AUTHOR
See the Info entry for **gcc**, or **<https://gcc.gnu.org/onlinedocs/gcc/Contributors.html>**, for contributors to GCC.

COPYRIGHT
Copyright (c) 1988−2025 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with the Invariant Sections being "GNU General Public License" and "Funding Free Software", the Front−Cover texts being (a) (see below), and with the Back−Cover Texts being (b) (see below). A copy of the license is included in the **gfdl** (7) man page.

(a) The FSF's Front−Cover Text is:

    A GNU Manual

(b) The FSF's Back−Cover Text is:

    You have freedom to copy and modify this GNU Manual, like GNU
    software. Copies published by the Free Software Foundation raise
    funds for GNU development.