NAME

gcc − GNU project C and C++ compiler

SYNOPSIS

gcc [**−c**|**−S**|**−E**] [**−std=***standard*]
[**−g**] [**−pg**] [**−O***level*]
[**−W***warn*...] [**−pedantic**]
[**−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 stage of processing. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; most of these are not documented here, since you rarely need to use any of them.

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

The **gcc** program accepts options and file names as operands. Many options have multi-letter names; therefore multiple single-letter options may *not* be grouped: **−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** would be **−fno−foo**. This manual documents only one of these two forms, whichever one is not the default.

OPTIONS

Option Summary

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

Overall Options
−c −S −E −o *file* **−no−canonical−prefixes −pipe −pass−exit−codes −x** *language* **−v −###** **−−help**[=*class*[**,...**]] **−−target−help −−version −wrapper** @*file* **−fplugin=***file* **−fplugin−arg−***name***=***arg* **−fdump−ada−spec**[**−slim**] **−fdump−go−spec=***file*

C Language Options
−ansi −std=*standard* **−fgnu89−inline −aux−info** *filename* **−fallow−parameterless−variadic−functions −fno−asm −fno−builtin −fno−builtin−***function* **−fhosted −ffreestanding −fopenmp −fms−extensions −fplan9−extensions −trigraphs −no−integrated−cpp −traditional −traditional−cpp −fallow−single−precision −fcond−mismatch −flax−vector−conversions −fsigned−bitfields −fsigned−char −funsigned−bitfields −funsigned−char**

C++ Language Options
−fabi−version=*n* **−fno−access−control −fcheck−new −fconserve−space −fconstexpr−depth=***n* **−ffriend−injection −fno−elide−constructors −fno−enforce−eh−specs −ffor−scope −fno−for−scope −fno−gnu−keywords −fno−implicit−templates −fno−implicit−inline−templates −fno−implement−inlines −fms−extensions −fno−nonansi−builtins −fnothrow−opt −fno−operator−names −fno−optional−diags −fpermissive −fno−pretty−templates −frepo**

−fno−rtti −fstats −ftemplate−depth=*n* **−fno−threadsafe−statics −fuse−cxa−atexit −fno−weak −nostdinc++ −fno−default−inline −fvisibility−inlines−hidden −fvisibility−ms−compat −Wabi −Wconversion−null −Wctor−dtor−privacy −Wdelete−non−virtual−dtor −Wnarrowing −Wnoexcept −Wnon−virtual−dtor −Wreorder −Weffc++ −Wstrict−null−sentinel −Wno−non−template−friend −Wold−style−cast −Woverloaded−virtual −Wno−pmf−conversions −Wsign−promo**

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 −freplace−objc−classes −fzero−link −gen−decls −Wassign−intercept −Wno−protocol −Wselector −Wstrict−selector−match −Wundeclared−selector**

Language Independent Options

−fmessage−length=*n* **−fdiagnostics−show−location=**[**once**|**every-line**] **−fno−diagnostics−show−option**

Warning Options

−fsyntax−only −fmax−errors=*n* **−pedantic −pedantic−errors −w −Wextra −Wall −Waddress −Waggregate−return −Warray−bounds −Wno−attributes −Wno−builtin−macro−redefined −Wc++−compat −Wc++11−compat −Wcast−align −Wcast−qual −Wchar−subscripts −Wclobbered −Wcomment −Wconversion −Wcoverage−mismatch −Wno−cpp −Wno−deprecated −Wno−deprecated−declarations −Wdisabled−optimization −Wno−div−by−zero −Wdouble−promotion −Wempty−body −Wenum−compare −Wno−endif−labels −Werror −Werror=* −Wfatal−errors −Wfloat−equal −Wformat −Wformat=2 −Wno−format−contains−nul −Wno−format−extra−args −Wformat−nonliteral −Wformat−security −Wformat−y2k −Wframe−larger−than=***len* **−Wno−free−nonheap−object −Wjump−misses−init −Wignored−qualifiers −Wimplicit −Wimplicit−function−declaration −Wimplicit−int −Winit−self −Winline −Wmaybe−uninitialized −Wno−int−to−pointer−cast −Wno−invalid−offsetof −Winvalid−pch −Wlarger−than=***len* **−Wunsafe−loop−optimizations −Wlogical−op −Wlong−long −Wmain −Wmaybe−uninitialized −Wmissing−braces −Wmissing−field−initializers −Wmissing−format−attribute −Wmissing−include−dirs −Wno−mudflap −Wno−multichar −Wnonnull −Wno−overflow −Woverlength−strings −Wpacked −Wpacked−bitfield−compat −Wpadded −Wparentheses −Wpedantic−ms−format −Wno−pedantic−ms−format −Wpointer−arith −Wno−pointer−to−int−cast −Wredundant−decls −Wreturn−type −Wsequence−point −Wshadow −Wsign−compare −Wsign−conversion −Wstack−protector −Wstack−usage=***len* **−Wstrict−aliasing −Wstrict−aliasing=n −Wstrict−overflow −Wstrict−overflow=***n* **−Wsuggest−attribute=**[**pure**|**const**|**noreturn**] **−Wswitch −Wswitch−default −Wswitch−enum −Wsync−nand −Wsystem−headers −Wtrampolines −Wtrigraphs −Wtype−limits −Wundef −Wuninitialized −Wunknown−pragmas −Wno−pragmas −Wunsuffixed−float−constants −Wunused −Wunused−function −Wunused−label −Wunused−local−typedefs −Wunused−parameter −Wno−unused−result −Wunused−value −Wunused−variable −Wunused−but−set−parameter −Wunused−but−set−variable −Wvariadic−macros −Wvector−operation−performance −Wvla −Wvolatile−register−var −Wwrite−strings −Wzero−as−null−pointer−constant**

C and Objective-C-only Warning Options

−Wbad−function−cast −Wmissing−declarations −Wmissing−parameter−type −Wmissing−prototypes −Wnested−externs −Wold−style−declaration −Wold−style−definition −Wstrict−prototypes −Wtraditional −Wtraditional−conversion −Wdeclaration−after−statement −Wpointer−sign

Debugging Options

−d*letters* **−dumpspecs −dumpmachine −dumpversion −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−noaddr**

−fdump−unnumbered **−fdump−unnumbered−links** **−fdump−translation−unit**[−*n*] **−fdump−class−hierarchy**[−*n*] **−fdump−ipa−all** **−fdump−ipa−cgraph** **−fdump−ipa−inline** **−fdump−passes** **−fdump−statistics** **−fdump−tree−all** **−fdump−tree−original**[−*n*] **−fdump−tree−optimized**[−*n*] **−fdump−tree−cfg** **−fdump−tree−vcg** **−fdump−tree−alias** **−fdump−tree−ch** **−fdump−tree−ssa**[−*n*] **−fdump−tree−pre**[−*n*] **−fdump−tree−ccp**[−*n*] **−fdump−tree−dce**[−*n*] **−fdump−tree−gimple**[−raw] **−fdump−tree−mudflap**[−*n*] **−fdump−tree−dom**[−*n*] **−fdump−tree−dse**[−*n*] **−fdump−tree−phiprop**[−*n*] **−fdump−tree−phiopt**[−*n*] **−fdump−tree−forwprop**[−*n*] **−fdump−tree−copyrename**[−*n*] **−fdump−tree−nrv** **−fdump−tree−vect** **−fdump−tree−sink** **−fdump−tree−sra**[−*n*] **−fdump−tree−forwprop**[−*n*] **−fdump−tree−fre**[−*n*] **−fdump−tree−vrp**[−*n*] **−ftree−vectorizer−verbose=***n* **−fdump−tree−storeccp**[−*n*] **−fdump−final−insns=***file* **−fcompare−debug**[=*opts*] **−fcompare−debug−second** **−feliminate−dwarf2−dups** **−feliminate−unused−debug−types** **−feliminate−unused−debug−symbols** **−femit−class−debug−always** **−fenable−***kind***−***pass* **−fenable−***kind***−***pass***=***range-list* **−fdebug−types−section** **−fmem−report** **−fpre−ipa−mem−report** **−fpost−ipa−mem−report** **−fprofile−arcs** **−frandom−seed=***string* **−fsched−verbose=***n* **−fsel−sched−verbose** **−fsel−sched−dump−cfg** **−fsel−sched−pipelining−verbose** **−fstack−usage** **−ftest−coverage** **−ftime−report** **−fvar−tracking** **−fvar−tracking−assignments** **−fvar−tracking−assignments−toggle** **−g** **−g***level* **−gtoggle** **−gcoff** **−gdwarf−***version* **−ggdb** **−grecord−gcc−switches** **−gno−record−gcc−switches** **−gstabs** **−gstabs+** **−gstrict−dwarf** **−gno−strict−dwarf** **−gvms** **−gxcoff** **−gxcoff+** **−fno−merge−debug−strings** **−fno−dwarf2−cfi−asm** **−fdebug−prefix−map=***old***=***new* **−femit−struct−debug−baseonly** **−femit−struct−debug−reduced** **−femit−struct−debug−detailed**[=*spec-list*] **−p** **−pg** **−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*]

Optimization Options

−falign−functions[=*n***]** **−falign−jumps[=***n***]** **−falign−labels[=***n***]** **−falign−loops[=***n***]** **−fassociative−math** **−fauto−inc−dec** **−fbranch−probabilities** **−fbranch−target−load−optimize** **−fbranch−target−load−optimize2** **−fbtr−bb−exclusive** **−fcaller−saves** **−fcheck−data−deps** **−fcombine−stack−adjustments** **−fconserve−stack** **−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** **−fdse** **−fearly−inlining** **−fipa−sra** **−fexpensive−optimizations** **−ffat−lto−objects** **−ffast−math** **−ffinite−math−only** **−ffloat−store** **−fexcess−precision=***style* **−fforward−propagate** **−ffp−contract=***style* **−ffunction−sections** **−fgcse** **−fgcse−after−reload** **−fgcse−las** **−fgcse−lm** **−fgraphite−identity** **−fgcse−sm** **−fif−conversion** **−fif−conversion2** **−findirect−inlining** **−finline−functions** **−finline−functions−called−once** **−finline−limit=***n* **−finline−small−functions** **−fipa−cp** **−fipa−cp−clone** **−fipa−matrix−reorg** **−fipa−pta** **−fipa−profile** **−fipa−pure−const** **−fipa−reference** **−fira−algorithm=***algorithm* **−fira−region=***region* **−fira−loop−pressure** **−fno−ira−share−save−slots** **−fno−ira−share−spill−slots** **−fira−verbose=***n* **−fivopts** **−fkeep−inline−functions** **−fkeep−static−consts** **−floop−block** **−floop−flatten** **−floop−interchange** **−floop−strip−mine** **−floop−parallelize−all** **−flto** **−flto−compression−level** **−flto−partition=***alg* **−flto−report** **−fmerge−all−constants** **−fmerge−constants** **−fmodulo−sched** **−fmodulo−sched−allow−regmoves** **−fmove−loop−invariants** **fmudflap** **−fmudflapir** **−fmudflapth** **−fno−branch−count−reg** **−fno−default−inline** **−fno−defer−pop** **−fno−function−cse** **−fno−guess−branch−probability** **−fno−inline** **−fno−math−errno** **−fno−peephole** **−fno−peephole2** **−fno−sched−interblock** **−fno−sched−spec** **−fno−signed−zeros** **−fno−toplevel−reorder** **−fno−trapping−math** **−fno−zero−initialized−in−bss** **−fomit−frame−pointer** **−foptimize−register−move** **−foptimize−sibling−calls** **−fpartial−inlining** **−fpeel−loops** **−fpredictive−commoning** **−fprefetch−loop−arrays** **−fprofile−correction** **−fprofile−dir=***path* **−fprofile−generate** **−fprofile−generate=***path* **−fprofile−use** **−fprofile−use=***path* **−fprofile−values** **−freciprocal−math** **−free** **−fregmove** **−frename−registers** **−freorder−blocks** **−freorder−blocks−and−partition** **−freorder−functions** **−frerun−cse−after−loop**

−freschedule−modulo−scheduled−loops −frounding−math −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−insns −fschedule−insns2 −fsection−anchors −fselective−scheduling −fselective−scheduling2 −fsel−sched−pipelining −fsel−sched−pipelining−outer−loops −fshrink−wrap −fsignaling−nans −fsingle−precision−constant −fsplit−ivs−in−unroller −fsplit−wide−types −fstack−protector −fstack−protector−all −fstrict−aliasing −fstrict−overflow −fthread−jumps −ftracer −ftree−bit−ccp −ftree−builtin−call−dce −ftree−ccp −ftree−ch −ftree−copy−prop −ftree−copyrename −ftree−dce −ftree−dominator−opts −ftree−dse −ftree−forwprop −ftree−fre −ftree−loop−if−convert −ftree−loop−if−convert−stores −ftree−loop−im −ftree−phiprop −ftree−loop−distribution −ftree−loop−distribute−patterns −ftree−loop−ivcanon −ftree−loop−linear −ftree−loop−optimize −ftree−parallelize−loops=***n* **−ftree−pre −ftree−pta −ftree−reassoc −ftree−sink −ftree−sra −ftree−switch−conversion −ftree−tail−merge −ftree−ter −ftree−vect−loop−version −ftree−vectorize −ftree−vrp −funit−at−a−time −funroll−all−loops −funroll−loops −funsafe−loop−optimizations −funsafe−math−optimizations −funswitch−loops −fvariable−expansion−in−unroller −fvect−cost−model −fvpt −fweb −fwhole−program −fwpa −fuse−linker−plugin −−param** *name*=*value* **−O −O0 −O1 −O2 −O3 −Os −Ofast**

Preprocessor Options

−A*question*=*answer* **−A−***question*[=*answer*] **−C −dD −dI −dM −dN −D***macro*[=*defn*] **−E −H −idirafter** *dir* **−include** *file* **−imacros** *file* **−iprefix** *file* **−iwithprefix** *dir* **−iwithprefixbefore** *dir* **−isystem** *dir* **−imultilib** *dir* **−isysroot** *dir* **−M −MM −MF −MG −MP −MQ −MT −nostdinc −P −fdebug−cpp −ftrack−macro−expansion −fworking−directory −remap −trigraphs −undef −U***macro* **−Wp,***option* **−Xpreprocessor** *option*

Assembler Option

−Wa,*option* **−Xassembler** *option*

Linker Options

object-file-name **−l***library* **−nostartfiles −nodefaultlibs −nostdlib −pie −rdynamic −s −static −static−libgcc −static−libstdc++ −shared −shared−libgcc −symbolic −T** *script* **−Wl,***option* **−Xlinker** *option* **−u** *symbol*

Directory Options

−B*prefix* **−I***dir* **−iplugindir=***dir* **−iquote***dir* **−L***dir* **−specs=***file* **−I− −−sysroot=***dir*

Machine Dependent 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*

ARM Options **−mapcs−frame −mno−apcs−frame −mabi=***name* **−mapcs−stack−check −mno−apcs−stack−check −mapcs−float −mno−apcs−float −mapcs−reentrant −mno−apcs−reentrant −msched−prolog −mno−sched−prolog −mlittle−endian −mbig−endian −mwords−little−endian −mfloat−abi=***name* **−mfpe −mfp16−format=***name* **−mthumb−interwork −mno−thumb−interwork −mcpu=***name* **−march=***name* **−mfpu=***name* **−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 −mcirrus−fix−invalid−insns −mno−cirrus−fix−invalid−insns −mpoke−function−name −mthumb −marm −mtpcs−frame −mtpcs−leaf−frame −mcaller−super−interworking −mcallee−super−interworking −mtp=***name* **−mtls−dialect=***dialect* **−mword−relocations −mfix−cortex−m3−ldrd −munaligned−access**

AVR Options **−mmcu=***mcu* **−maccumulate−args −mbranch−cost=***cost* **−mcall−prologues −mint8 −mno−interrupts −mrelax −mshort−calls −mstrict−X −mtiny−stack**

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 **−mbig−endian −mlittle−endian −march=***cpu* **−msim −msdata=***sdata-type*

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

CR16 Options **−mmac −mcr16cplus −mcr16c −msim −mint32 −mbit−ops −mdata−model=***model*

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 −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 **−mno−fp−regs −msoft−float −malpha−as −mgas −mieee −mieee−with−inexact −mieee−conformant −mfp−trap−mode=***mode* **−mfp−rounding−mode=***mode* **−mtrap−precision=***mode* **−mbuild−constants −mcpu=***cpu-type* **−mtune=***cpu-type* **−mbwx −mmax −mfix −mcix −mfloat−vax −mfloat−ieee −mexplicit−relocs −msmall−data −mlarge−data −msmall−text −mlarge−text −mmemory−latency=***time*

DEC Alpha/VMS Options **−mvms−return−codes −mdebug−main=***prefix* **−mmalloc64**

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

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 **−mglibc −muclibc −mbionic −mandroid −tno−android−cc −tno−android−ld**

H8/300 Options **−mrelax −mh −ms −mn −mint32 −malign−300**

HPPA Options **−march=***architecture-type* **−mbig−switch −mdisable−fpregs −mdisable−indexing −mfast−indirect−calls −mgas −mgnu−ld −mhp−ld −mfixed−range=***register-range*

−mjump−in−delay −mlinker−opt −mlong−calls −mlong−load−store −mno−big−switch −mno−disable−fpregs −mno−disable−indexing −mno−fast−indirect−calls −mno−gas −mno−jump−in−delay −mno−long−load−store −mno−portable−runtime −mno−soft−float −mno−space−regs −msoft−float −mpa−risc−1−0 −mpa−risc−1−1 −mpa−risc−2−0 −mportable−runtime −mschedule=*cpu-type* **−mspace−regs −msio −mwsio −munix=***unix-std* **−nolibdld −static −threads**

i386 and x86−64 Options **−mtune=***cpu-type* **−march=***cpu-type* **−mfpmath=***unit* **−masm=***dialect* **−mno−fancy−math−387 −mno−fp−ret−in−387 −msoft−float −mno−wide−multiply −mrtd −malign−double −mpreferred−stack−boundary=***num* **−mincoming−stack−boundary=***num* **−mcld −mcx16 −msahf −mmovbe −mcrc32 −mrecip −mrecip=***opt* **−mvzeroupper −mmmx −msse −msse2 −msse3 −mssse3 −msse4.1 −msse4.2 −msse4 −mavx −mavx2 −maes −mpclmul −mfsgsbase −mrdrnd −mf16c −mfma −msse4a −m3dnow −mpopcnt −mabm −mbmi −mtbm −mfma4 −mxop −mlzcnt −mbmi2 −mlwp −mthreads −mno−align−stringops −minline−all−stringops −minline−stringops−dynamically −mstringop−strategy=***alg* **−mpush−args −maccumulate−outgoing−args −m128bit−long−double −m96bit−long−double −mregparm=***num* **−msseregparm −mveclibabi=***type* **−mvect8−ret−in−mem −mpc32 −mpc64 −mpc80 −mstackrealign −momit−leaf−frame−pointer −mno−red−zone −mno−tls−direct−seg−refs −mcmodel=***code-model* **−mabi=***name* **−m32 −m64 −mx32 −mlarge−data−threshold=***num* **−msse2avx −mfentry −m8bit−idiv −mavx256−split−unaligned−load −mavx256−split−unaligned−store**

i386 and x86−64 Windows Options **−mconsole −mcygwin −mno−cygwin −mdll −mnop−fun−dllimport −mthread −municode −mwin32 −mwindows −fno−set−stack−executable**

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*

IA−64/VMS Options **−mvms−return−codes −mdebug−main=***prefix* **−mmalloc64**

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

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*

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

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**

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

MeP Options **−mabsdiff −mall−opts −maverage −mbased=***n* **−mbitops −mc=***n* **−mclip −mconfig=***name* **−mcop −mcop32 −mcop64 −mivc2 −mdc −mdiv −meb −mel −mio−volatile −ml −mleadz −mm −mminmax −mmult −mno−opts −mrepeat −ms −msatur −msdram −msim −msimnovec −mtf −mtiny=***n*

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 −mxl−mode−***app-model*

MIPS Options **−EL −EB −march=***arch* **−mtune=***arch* **−mips1 −mips2 −mips3 −mips4 −mips32 −mips32r2 −mips64 −mips64r2 −mips16 −mno−mips16 −mflip−mips16 −minterlink−mips16 −mno−interlink−mips16 −mabi=***abi* **−mabicalls −mno−abicalls −mshared −mno−shared −mplt −mno−plt −mxgot −mno−xgot −mgp32 −mgp64 −mfp32 −mfp64 −mhard−float −msoft−float −msingle−float −mdouble−float −mdsp −mno−dsp −mdspr2 −mno−dspr2 −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 −mcheck−zero−division −mno−check−zero−division −mdivide−traps −mdivide−breaks −mmemcpy −mno−memcpy −mlong−calls −mno−long−calls −mmad −mno−mad −mfused−madd −mno−fused−madd −nocpp −mfix−24k −mno−fix−24k −mfix−r4000 −mno−fix−r4000 −mfix−r4400 −mno−fix−r4400 −mfix−r10000 −mno−fix−r10000 −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 −mfp−exceptions −mno−fp−exceptions −mvr4130−align −mno−vr4130−align −msynci −mno−synci −mrelax−pic−calls −mno−relax−pic−calls −mmcount−ra−address**

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

PDP−11 Options **−mfpu −msoft−float −mac0 −mno−ac0 −m40 −m45 −m10 −mbcopy −mbcopy−builtin −mint32 −mno−int16 −mint16 −mno−int32 −mfloat32 −mno−float64 −mfloat64 −mno−float32 −mabshi −mno−abshi −mbranch−expensive −mbranch−cheap −munix−asm −mdec−asm**

picoChip Options **−mae=***ae_type* **−mvliw−lookahead=***N* **−msymbol−as−address −mno−inefficient−warnings**

PowerPC Options See RS/6000 and PowerPC Options.

RL78 Options **−msim −mmul=none −mmul=g13 −mmul=rl78**

RS/6000 and PowerPC Options **−mcpu=***cpu-type* **−mtune=***cpu-type* **−mcmodel=***code-model* **−mpower −mno−power −mpower2 −mno−power2 −mpowerpc −mpowerpc64 −mno−powerpc −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 −mmfpgpr −mno−mfpgpr −mhard−dfp −mno−hard−dfp −mnew−mnemonics −mold−mnemonics −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 −msingle−float −mdouble−float −msimple−fpu −mstring −mno−string −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 −maltivec −mswdiv −msingle−pic−base −mprioritize−restricted−insns=*priority* **−msched−costly−dep=***dependence_type* **−minsert−sched−nops=***scheme* **−mcall−sysv −mcall−netbsd −maix−struct−return −msvr4−struct−return −mabi=***abi-type* **−msecure−plt −mbss−plt −mblock−move−inline−limit=***num* **−misel −mno−isel −misel=yes −misel=no −mspe −mno−spe −mspe=yes −mspe=no −mpaired −mgen−cell−microcode −mwarn−cell−microcode −mvrsave −mno−vrsave −mmulhw −mno−mulhw −mdlmzb −mno−dlmzb −mfloat−gprs=yes −mfloat−gprs=no −mfloat−gprs=single −mfloat−gprs=double −mprototype −mno−prototype −msim −mmvme −mads −myellowknife −memb −msdata −msdata=***opt* **−mvxworks −G** *num* **−pthread −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**

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 −msave−acc−in−interrupts**

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 −mtpf−trace −mno−tpf−trace −mfused−madd −mno−fused−madd −mwarn−framesize −mwarn−dynamicstack −mstack−size −mstack−guard**

Score Options **−meb −mel −mnhwloop −muls −mmac −mscore5 −mscore5u −mscore7 −mscore7d**

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 −m5−64media −m5−64media−nofpu −m5−32media −m5−32media−nofpu −m5−compact −m5−compact−nofpu −mb −ml −mdalign −mrelax −mbigtable −mfmovd −mhitachi −mrenesas −mno−renesas −mnomacsave −mieee −mno−ieee −mbitops −misize −minline−ic_invalidate −mpadstruct −mspace −mprefergot −musermode −multcost=***number* **−mdiv=***strategy* **−mdivsi3_libfunc=***name* **−mfixed−range=***register-range* **−madjust−unroll −mindexed−addressing −mgettrcost=***number* **−mpt−fixed −maccumulate−outgoing−args −minvalid−symbols −msoft−atomic −mbranch−cost=***num* **−mcbranchdi −mcmpeqdi −mfused−madd −mpretend−cmove**

Solaris 2 Options **−mimpure−text −mno−impure−text −pthreads −pthread**

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 −mlittle−endian −mstack−bias −mno−stack−bias −munaligned−doubles −mno−unaligned−doubles −mv8plus −mno−v8plus −mvis −mno−vis −mvis2 −mno−vis2 −mvis3 −mno−vis3 −mfmaf −mno−fmaf −mpopc −mno−popc −mfix−at697f**

SPU Options **−mwarn−reloc −merror−reloc −msafe−dma −munsafe−dma −mbranch−hints −msmall−mem −mlarge−mem −mstdmain −mfixed−range=***register-range* **−mea32 −mea64**

−maddress−space−conversion −mno−address−space−conversion −mcache−size=*cache-size* **−matomic−updates −mno−atomic−updates**

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

TILE-Gx Options **−mcpu=***cpu* **−m32 −m64**

TILEPro Options **−mcpu=***cpu* **−m32**

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 −mbig−switch**

VAX Options **−mg −mgnu −munix**

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

x86−64 Options See i386 and x86−64 Options.

Xstormy16 Options **−msim**

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 −mtarget−align −mno−target−align −mlongcalls −mno−longcalls**

zSeries Options See S/390 and zSeries Options.

Code Generation Options
−fcall−saved−*reg* **−fcall−used−***reg* **−ffixed−***reg* **−fexceptions −fnon−call−exceptions −funwind−tables −fasynchronous−unwind−tables −finhibit−size−directive −finstrument−functions −finstrument−functions−exclude−function−list=***sym***,***sym***,... −finstrument−functions−exclude−file−list=***file***,***file***,... −fno−common −fno−ident −fpcc−struct−return −fpic −fPIC −fpie −fPIE −fno−jump−tables −frecord−gcc−switches −freg−struct−return −fshort−enums −fshort−double −fshort−wchar −fverbose−asm −fpack−struct[=***n***] −fstack−check −fstack−limit−register=***reg* **−fstack−limit−symbol=***sym* **−fno−stack−limit −fsplit−stack −fleading−underscore −ftls−model=***model* **−ftrapv −fwrapv −fbounds−check −fvisibility −fstrict−volatile−bitfields**

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**
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**
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.**go**
Go source code.

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++−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
f77  f77−cpp−input f95  f95−cpp−input
go
java
```

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

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

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

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.

−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.

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

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

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

−−help={*class*|[ˆ]*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
This will display all of the optimization options supported by the compiler.

warnings
This will display all of the options controlling warning messages produced by the compiler.

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

params
This will display the values recognized by the **−−param** option.

language

This will display the options supported for *language*, where *language* is the name of one of the languages supported in this version of GCC.

common

This will 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 the following can be used:

```
        −−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 by so much that there is nothing to display. One case where it does work however is when one of the classes is *target*. So for example to display all the target-specific optimization options the following can be used:

```
        −−help=target,optimizers
```

The **−−help=** option can be repeated on the command line. Each successive use will display its requested class of options, skipping those that have already been displayed.

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
```

−no−canonical−prefixes
Do not expand any symbolic links, resolve references to **/../** or **/./**, or make the path absolute when generating a relative prefix.

−−version
Display the version number and copyrights of the invoked GCC.

−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 will invoke all subprograms of **gcc** under **gdb −−args**, thus the invocation of **cc1** will be **gdb −−args cc1 ...**.

−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.

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

However, the use of **gcc** does not add the C++ library. **g++** is a program that calls GCC and treats **.c**, **.h** and **.i** files as C++ source files instead of C source files unless **−x** is used, and automatically specifies linking against the C++ library. 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**.

This turns off certain features of GCC that are incompatible with ISO C90 (when compiling C code), or of standard C++ (when compiling C++ code), such as the `asm` and `typeof` keywords, and predefined macros such as `unix` and `vax` that identify the type of system you are using. It also enables the undesirable and rarely used ISO trigraph feature. For the C compiler, it disables recognition of C++ style // comments as well as the `inline` keyword.

The alternate keywords `__asm__`, `__extension__`, `__inline__` and `__typeof__` continue to work despite **−ansi**. You would not want to use them in an ISO C program, of course, but it is useful to put them in header files that might be included in compilations done with **−ansi**. Alternate predefined macros such as `__unix__` and `__vax__` are also available, with or without **−ansi**.

The **−ansi** option does not cause non-ISO programs to be rejected gratuitously. For that, **−pedantic** is required in addition to **−ansi**.

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

Functions that would normally be built in but do not have semantics defined by ISO C (such as `alloca` and `ffs`) are not built-in functions when **−ansi** is used.

−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**. By specifying a base standard, the compiler will accept all programs following that standard and 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, by specifying a GNU dialect of a standard, all features the compiler support are enabled, even when those features change the meaning of the base standard and some strict-conforming programs may be rejected. The particular standard is used by **−pedantic** to identify which features are GNU extensions given that version of the standard. For example **−std=gnu90 −pedantic** would warn about C++ style // comments, while **−std=gnu99 −pedantic** would 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. Note that this standard is not yet fully supported; see <**http://gcc.gnu.org/gcc−4.7/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. Support is incomplete and experimental. The name **c1x** is deprecated.

gnu90
gnu89
GNU dialect of ISO C90 (including some C99 features). This is the default for C code.

gnu99
gnu9x
GNU dialect of ISO C99. When ISO C99 is fully implemented in GCC, this will become the default. The name **gnu9x** is deprecated.

gnu11
gnu1x
GNU dialect of ISO C11. Support is incomplete and experimental. The name **gnu1x** is deprecated.

c++98
The 1998 ISO C++ standard plus amendments. Same as **−ansi** for C++ code.

gnu++98
GNU dialect of **−std=c++98**. This is the default for C++ code.

c++11
The 2011 ISO C++ standard plus amendments. Support for C++11 is still experimental, and may change in incompatible ways in future releases.

gnu++11
GNU dialect of **−std=c++11**. Support for C++11 is still experimental, and may change in incompatible ways in future releases.

−fgnu89−inline
The option **−fgnu89−inline** tells GCC to use the traditional GNU semantics for `inline` functions when in C99 mode.
This option is accepted and ignored by GCC versions 4.1.3 up to but not including 4.3. In GCC versions 4.3 and later it changes the behavior of GCC 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 was first supported in GCC 4.3. 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.

−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.

−fallow−parameterless−variadic−functions

Accept variadic functions without named parameters.

Although it is possible to define such a function, this is not very useful as it is not possible to read the arguments. This is only supported for C as this construct is allowed by C++.

−fno−asm

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

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

−fno−builtin
−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))
```

−fhosted

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

−ffreestanding

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

−fopenmp

Enable handling of OpenMP directives `#pragma omp` in C/C++ and `!$omp` in Fortran. When **−fopenmp** is specified, the compiler generates parallel code according to the OpenMP Application Program Interface v3.0 <**http://www.openmp.org/**>. This option implies **−pthread**, and thus is only supported on targets that have support for **−pthread**.

−fgnu−tm

When the option **−fgnu−tm** is specified, the compiler will generate 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**).

−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.

−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++.

−trigraphs

Support ISO C trigraphs. The **−ansi** option (and **−std** options for strict ISO C conformance) implies **−trigraphs**.

−no−integrated−cpp

Performs a compilation in two passes: preprocessing and compiling. This option allows a user supplied "cc1", "cc1plus", or "cc1obj" via the **−B** option. The user supplied compilation step can then add in an additional preprocessing step after normal preprocessing but before compiling. The default is to use the integrated cpp (internal cpp)

The semantics of this option will change if "cc1", "cc1plus", and "cc1obj" are merged.

−traditional
−traditional−cpp

Formerly, these options caused GCC to attempt to emulate a pre-standard C compiler. They are now only supported with the **−E** switch. The preprocessor continues to support a pre-standard mode. See the GNU CPP manual for details.

−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++.

−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.

−funsigned−char

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

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

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

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

−fsigned−char

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

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

−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.

Options Controlling C++ Dialect

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

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

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

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

−fabi−version=*n*

Use version *n* of the C++ ABI. Version 2 is the version of the C++ ABI that first appeared in G++ 3.4. Version 1 is the version of the C++ ABI that first appeared in G++ 3.2. Version 0 will always be the version that conforms most closely to the C++ ABI specification. Therefore, the ABI obtained using version 0 will change as ABI bugs are fixed.

The default is version 2.

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.

See also **−Wabi**.

−fno−access−control

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

−fcheck−new

Check that the pointer returned by `operator new` is non-null before attempting to modify the storage allocated. This check is normally unnecessary because the C++ standard specifies that `operator new` will only return `0` if it is declared ***throw()***, in which case the compiler will always check 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)**.

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

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

−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.

−fdeduce−init−list
Enable deduction of a template type parameter as std::initializer_list from a brace-enclosed initializer list, i.e.

```
template <class T> auto forward(T t) −> decltype (realfn (t))
{
  return realfn (t);
}

void f()
{
  forward({1,2}); // call forward<std::initializer_list<int>>
}
```

This deduction was implemented as a possible extension to the originally proposed semantics for the C++11 standard, but was not part of the final standard, so it is disabled by default. This option is deprecated, and may be removed in a future version of G++.

−ffriend−injection
Inject friend functions into the enclosing namespace, so that they are visible outside the scope of the class in which they are declared. Friend functions were documented to work this way in the old Annotated C++ Reference Manual, and versions of G++ before 4.1 always worked that way. However, in ISO C++ a friend function that is not declared in an enclosing scope can only be found using argument dependent lookup. This option causes friends to be injected as they were in earlier releases.

This option is for compatibility, and may be removed in a future release of G++.

−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.

−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 will still optimize based on the specifications, so throwing an unexpected exception will result in undefined behavior.

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

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

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

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

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

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

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

−fno−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 though 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 will actually make 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 will result 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.

−fpermissive
Downgrade some diagnostics about nonconformant code from errors to warnings. Thus, using **−fpermissive** will allow some nonconforming code to compile.

−fno−pretty−templates
When an error message refers to a specialization of a function template, the compiler will normally print 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 will omit 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, using **−fno−pretty−templates** will disable them.

−frepo
Enable automatic template instantiation at link time. This option also implies **−fno−implicit−templates**.

−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 it will generate 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.

−fstats
Emit statistics about front-end processing at the end of the compilation. This information is generally only useful to the G++ development team.

−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.

−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 will only work if your C library supports `__cxa_atexit`.

−fno−use−cxa−get−exception−ptr
Don't use the `__cxa_get_exception_ptr` runtime routine. This will cause `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 were 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 will have 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 different shared object: those declarations are permitted if they would have been permitted when this option was 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 will be different, so changing one will 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++ will use weak symbols if they are available. This option exists only for testing, and should not be used by end-users; it will result in inferior code and has no benefits. This option may be removed in a future release of G++.

−nostdinc++

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

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

−fno−default−inline

Do not assume **inline** for functions defined inside a class scope.
Note that these functions will have linkage like inline functions; they just won't be inlined by default.

−Wabi (C, Objective-C, C++ and Objective−C++ only)

Warn when G++ generates code that is probably not compatible with the vendor-neutral C++ ABI. 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 will be 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.

The known incompatibilities in **−fabi−version=2** (the default) include:

- A template with a non-type template parameter of reference type is mangled incorrectly:

  ```
  extern int N;
  template <int &> struct S {};
  void n (S<N>) {2}
  ```

  This is fixed in **−fabi−version=3**.

- SIMD vector types declared using `__attribute ((vector_size))` are mangled in a non-standard way that does not allow for overloading of functions taking vectors of different sizes.

  The mangling is changed in **−fabi−version=4**.

The known incompatibilities in **−fabi−version=1** include:

- Incorrect handling of tail-padding for bit-fields. G++ may attempt to pack data into the same byte as a base class. For example:

  ```
  struct A { virtual void f(); int f1 : 1; };
  struct B : public A { int f2 : 1; };
  ```

  In this case, G++ will place `B::f2` into the same byte as`A::f1`; other compilers will not. You can avoid this problem by explicitly padding `A` so that its size is a multiple of the byte size on your platform; that will cause G++ and other compilers to layout `B` identically.

- Incorrect handling of tail-padding for virtual bases. G++ does not use tail padding when laying out virtual bases. For example:

  ```
  struct A { virtual void f(); char c1; };
  struct B { B(); char c2; };
  struct C : public A, public virtual B {};
  ```

  In this case, G++ will not place `B` into the tail-padding for `A`; other compilers will. You can avoid this problem by explicitly padding `A` so that its size is a multiple of its alignment (ignoring virtual base classes); that will cause G++ and other compilers to layout `C` identically.

- Incorrect handling of bit-fields with declared widths greater than that of their underlying types, when the bit-fields appear in a union. For example:

  ```
  union U { int i : 4096; };
  ```

  Assuming that an `int` does not have 4096 bits, G++ will make the union too small by the number of bits in an `int`.

- Empty classes can be placed at incorrect offsets. For example:

  ```
  struct A {};

  struct B {
    A a;
    virtual void f ();
  };

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

  G++ will place the `A` base class of `C` at a nonzero offset; it should be placed at offset zero. G++ mistakenly believes that the `A` data member of `B` is already at offset zero.

- Names of template functions whose types involve `typename` or template template parameters can be mangled incorrectly.

  ```
  template <typename Q>
  void f(typename Q::X) {}

  template <template <typename> class Q>
  void f(typename Q<int>::X) {}
  ```

  Instantiations of these templates may be mangled incorrectly.

It also warns psABI related changes. The known psABI changes at this point include:

- For SYSV/x86−64, when passing union with long double, it is changed to pass in memory as specified in psABI. For example:

  ```
  union U {
    long double ld;
    int i;
  };
  ```

`union U` will always be passed in memory.

−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.

−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**.

−Wnarrowing (C++ and Objective−C++ only)
: 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**.

With −std=c++11, **−Wno−narrowing** suppresses the diagnostic required by the standard. Note that this does not affect the meaning of well-formed code; narrowing conversions are still considered ill-formed in SFINAE context.

−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.

−Wnon−virtual−dtor (C++ and Objective−C++ only)
: Warn when a class has virtual functions and accessible non-virtual destructor, in which case it would be possible but unsafe to delete an instance of a derived class through a pointer to the base class. This warning is also enabled if **−Weffc++** is specified.

−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 will rearrange 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**.

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

−Weffc++ (C++ and Objective−C++ only)
: Warn about violations of the following style guidelines from Scott Meyers' *Effective C++, Second Edition* book:

- Item 11: Define a copy constructor and an assignment operator for classes with dynamically allocated memory.
- Item 12: Prefer initialization to assignment in constructors.
- Item 14: Make destructors virtual in base classes.
- Item 15: Have `operator=` return a reference to `*this`.
- Item 23: Don't try to return a reference when you must return an object.

Also warn about violations of the following style guidelines from Scott Meyers' *More Effective C++* book:

- Item 6: Distinguish between prefix and postfix forms of increment and decrement operators.
- Item 7: Never overload `&&`, `||`, or `,`.

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.

−Wstrict−null−sentinel (C++ and Objective−C++ only)
Warn also 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 not 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-templatized friend functions are declared within a template. Since the advent of explicit template specification support in G++, if the name of the friend is an unqualified-id (i.e., **friend foo(int)**), the C++ language specification demands that the friend declare or define an ordinary, nontemplate function. (Section 14.5.3). Before G++ implemented explicit specification, unqualified-ids could be interpreted as a particular specialization of a templatized function. Because this non-conforming behavior is no longer the default behavior for G++, **−Wnon−template−friend** allows the compiler to check existing code for potential trouble spots and is on by default. This new compiler behavior can be turned off with **−Wno−non−template−friend**, which keeps the conformant compiler code but disables the helpful warning.

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

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

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

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

```
        B* b;
        b−>f();
```

will fail to compile.

−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++ would try to preserve unsignedness, but the standard mandates the current behavior.

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

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

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

Options 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, but 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=*class-name*

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). The **−fconstant−cfstrings** option, if also present, will override the **−fconstant−string−class** setting and cause `@"..."` literals to be laid out as constant CoreFoundation strings.

−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 and Mac OS X. 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 will run 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 will run all such default destructors, in reverse order.

The `− (id) .cxx_construct` and `− (void) .cxx_destruct` methods thusly generated will 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 will be invoked by the runtime immediately after a new object instance is allocated; the `− (void) .cxx_destruct` methods will be 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++ and Java. 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.

−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−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.

−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 will omit 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 to Control Diagnostic Messages Formatting

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

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

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

−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.

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
Make all warnings into errors.

−Werror=
Make 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 **−W**, for example **−Wimplicit** to request warnings on implicit declarations. Each of these specific warning options also has a negative form beginning **−Wno−** to turn off warnings; for example, **−Wno−implicit**. This manual lists only one of the two forms, whichever is not the default. For further, language-specific options also refer to **C++ Dialect Options** and **Objective-C and Objective−C++ Dialect Options**.

When an unrecognized warning option is requested (e.g., **−Wunknown−warning**), GCC will emit a diagnostic stating that the option is not recognized. However, if the **−Wno−** form is used, the behavior is slightly different: No diagnostic will be 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 will warn that an unrecognized option was used.

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

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

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

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

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

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 **−pedantic** are given where they are required by the base standard. (It would 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
Like **−pedantic**, except that errors are produced rather than warnings.

−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 **−Warray−bounds** (only with **−O2**) **−Wc++11−compat** **−Wchar−subscripts** **−Wenum−compare** (in C/Objc; this is on by default in C++) **−Wimplicit−int** (C and Objective-C only) **−Wimplicit−function−declaration** (C and Objective-C only) **−Wcomment** **−Wformat** **−Wmain** (only for C/ObjC and unless **−ffreestanding**) **−Wmaybe−uninitialized** **−Wmissing−braces** **−Wnonnull** **−Wparentheses** **−Wpointer−sign** **−Wreorder** **−Wreturn−type** **−Wsequence−point** **−Wsign−compare** (only in C++) **−Wstrict−aliasing** **−Wstrict−overflow=1** **−Wswitch** **−Wtrigraphs** **−Wuninitialized** **−Wunknown−pragmas** **−Wunused−function** **−Wunused−label** **−Wunused−value** **−Wunused−variable** **−Wvolatile−register−var**

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.)

−Wclobbered **−Wempty−body** **−Wignored−qualifiers** **−Wmissing−field−initializers** **−Wmissing−parameter−type** (C only) **−Wold−style−declaration** (C only) **−Woverride−init** **−Wsign−compare** **−Wtype−limits** **−Wuninitialized** **−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 a derived class' copy constructor.

−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**.

−Wcomment
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**.

−Wno−coverage−mismatch
Warn if feedback profiles do not match when using the **−fprofile−use** option. If a source file was changed between **−fprofile−gen** and **−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−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 will perform the entire computation with `double` because the floating-point literal is a `double`.

−Wformat
Check calls to `printf` and `scanf`, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense. This includes standard functions, and others specified by format attributes, in the `printf`, `scanf`, `strftime` and `strfmon` (an X/Open extension, not in the C standard) families (or other target-specific families). Which functions are checked without format attributes having been specified depends on the standard version selected, and such checks of functions without the attribute specified are disabled by **−ffreestanding** or **−fno−builtin**.

The formats are checked against the format features supported by GNU libc version 2.2. These include all ISO C90 and C99 features, as well as features from the Single Unix Specification and some BSD and GNU extensions. Other library implementations may not support all these features; GCC does not support warning about features that go beyond a particular library's limitations. However, if **−pedantic** is used with **−Wformat**, warnings will be given about format features not in the selected standard version (but not for `strfmon` formats, since those are not in any version of the C standard).

Since **−Wformat** also checks for null format arguments for several functions, **−Wformat** also implies **−Wnonnull**.

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

−Wformat−y2k
If **−Wformat** is specified, also warn about `strftime` formats that may yield only a two-digit year.

−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 will suppress the warning if the unused arguments are all pointers, since the Single Unix Specification says that such unused arguments are allowed.

−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=2
Enable **−Wformat** plus format checks not included in **−Wformat**. Currently equivalent to **−Wformat −Wformat−nonliteral −Wformat−security −Wformat−y2k**.

−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.

−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 will warn about `i` being uninitialized in the following snippet only when **−Winit−self** has been specified:

```
int f()
{
  int i = i;
  return i;
}
```

−Wimplicit−int (C and Objective-C only)
Warn when a declaration does not specify a type. This warning is enabled by **−Wall**.

−Wimplicit−function−declaration (C and Objective-C only)
Give a warning whenever a function is used before being declared. In C99 mode (**−std=c99** or **−std=gnu99**), this warning is enabled by default and it is made into an error by **−pedantic−errors**. This warning is also enabled by **−Wall**.

−Wimplicit (C and Objective-C only)
Same as **−Wimplicit−int** and **−Wimplicit−function−declaration**. This warning is enabled by **−Wall**.

−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**.

−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 **−pedantic**.

−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 and Objective−C++ only)
Warn if a user-supplied include directory does not exist.

−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 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 will issue a warning when this flag is specified. To eliminate the warning, add explicit braces around the innermost `if` statement so there is no way the `else` could belong to the enclosing `if`. The resulting code would look like this:

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

Also warn for dangerous uses of the ?: with omitted middle operand GNU extension. When the condition in the ?: operator is a boolean expression the omitted value will be always 1. Often the user expects it to be a value computed inside the conditional expression instead.

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 defines 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 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 <**http://gcc.gnu.org/readings.html**>.

This warning is enabled by **−Wall** for C and C++.

−Wreturn−type

Warn whenever a function is defined with a return-type that defaults to `int`. Also warn about any `return` statement with no return-value in a function whose return-type is not `void` (falling off the end of the function body is considered returning without a value), and about a `return` statement with an expression in a function whose return-type is `void`.

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

This warning is enabled by **−Wall**.

−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 outside the enumeration range also provoke warnings when this option is used (even if there is a `default` label). 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 outside the enumeration range also provoke warnings when this option is used. 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.

−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.

−Wtrigraphs
Warn if any trigraphs are encountered that might change the meaning of the program (trigraphs within comments are not warned about). This warning is enabled by **−Wall**.

−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.

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

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 variable or non-constant static variable is unused aside from its declaration. This warning is enabled by **−Wall**.

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

−Wunused−value

Warn whenever a statement computes a result that is explicitly not used. To suppress this warning cast the 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]** will cause a warning, while **x[(void)i,j]** will not.

This warning is enabled by **−Wall**.

−Wunused

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

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**.

−Wuninitialized

Warn if an automatic variable is used without first being initialized or if a variable may be clobbered by a `setjmp` call. In C++, warn if a non-static reference or non-static **const** member appears in a class without constructors.

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 or clobbered elements of structure, union or array variables as well as for variables that are uninitialized or clobbered 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 will 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.

−Wmaybe−uninitialized

For an automatic variable, if there exists a path from the function entry to a use of the variable that is initialized, but there exist some other paths the variable is not initialized, the compiler will emit a warning if it can not prove the uninitialized paths do not happen at run time. These warnings are made optional because GCC is not smart enough to see all the reasons why the code might be correct despite appearing to have an error. Here is one example of how this can happen:

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

If the value of `y` is always 1, 2 or 3, then `x` is always initialized, but GCC doesn't know this. To suppress the warning, the user needs 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`. These warnings as well are possible only in optimizing compilation.

The compiler sees only the calls to `setjmp`. It cannot know where `longjmp` will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact no problem because `longjmp` cannot in fact be called at the place 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 will even be issued for unknown pragmas in system header files. This is not the case if the warnings were only enabled by the **−Wall** command-line option.

−Wno−pragmas
Do not warn about misuses of pragmas, such as incorrect parameters, invalid syntax, or conflicts between pragmas. See also **−Wunknown−pragmas**.

−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=n**, with n=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 **−fstrict−overflow** is active. 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 will require, 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: `x + 1 > x`; with **−fstrict−overflow**, the compiler will simplify this 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 **−fstrict−overflow** is in effect, 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` will be simplified to `x > 0`.

−Wstrict−overflow=4
Also warn about other simplifications not covered by the above cases. For example: `(x * 10) / 5` will be 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` will be 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 will give a very large number of false positives.

−Wsuggest−attribute=[**pure**|**const**|**noreturn**]
Warn for cases where adding an attribute may be beneficial. The attributes currently supported are listed below.

−Wsuggest−attribute=pure
−Wsuggest−attribute=const
−Wsuggest−attribute=noreturn
Warn about functions that might be candidates for attributes `pure`, `const` or `noreturn`. 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 nor returns 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.

−Warray−bounds
This option is only active when **−ftree−vrp** is active (default for **−O2** and above). It warns about subscripts to arrays that are always out of bounds. This warning is enabled by **−Wall**.

−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 will *not* warn about unknown pragmas in system headers−−−for that, **−Wunknown−pragmas** must also be used.

−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 would check to see whether the two values have ranges that overlap; and this is done with the relational operators, so equality comparisons are probably mistaken.

−Wtraditional (C 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 does not in ISO C.
- In traditional C, some preprocessor directives did not exist. Traditional preprocessors would only consider a line to be a directive if the **#** appeared in column 1 on the line. Therefore **−Wtraditional** warns about directives that traditional C understands but would ignore because the **#** does not appear as the first character on the line. It also suggests you hide directives like **#pragma** not understood by traditional C by indenting them. Some traditional implementations would not recognize **#elif**, so it suggests avoiding it altogether.
- A function-like macro that appears without arguments.
- The unary plus operator.
- The **U** integer constant suffix, or the **F** or **L** floating-point constant suffixes. (Traditional C does support the **L** suffix on integer constants.) Note, these suffixes appear in macros defined in the system headers of most modern systems, e.g. the **_MIN**/**_MAX** macros in `<limits.h>`. Use of these macros 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 would cause 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 will 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 and was not supported by GCC versions before GCC 3.0.

−Wundef
Warn if an undefined identifier is evaluated in an **#if** directive.

−Wno−endif−labels
Do not warn whenever an **#else** or an **#endif** are followed by text.

−Wshadow
Warn whenever a local variable or type declaration shadows another variable, parameter, type, or class member (in C++), or whenever a built-in function is shadowed. Note that in C++, the compiler will not warn if a local variable shadows a struct/class/enum, but will warn if it shadows an explicit typedef.

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

−Wframe−larger−than=*len*
Warn if the size of a function frame is larger than *len* bytes. The computation done to determine the stack frame size is approximate and not conservative. The actual requirements may be somewhat greater than *len* 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.

−Wno−free−nonheap−object
Do not warn when attempting to free an object that was not allocated on the heap.

−Wstack−usage=*len*
Warn if the stack usage of a function might be larger than *len* bytes. 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
```

−Wunsafe−loop−optimizations
Warn if the loop cannot be optimized because the compiler could not assume anything on the bounds of the loop indices. With **−funsafe−loop−optimizations** warn if the compiler made such assumptions.

−Wno−pedantic−ms−format (MinGW targets only)
Disables the warnings about non-ISO `printf` / `scanf` format width specifiers `I32`, `I64`, and `I` used on Windows targets depending on the MS runtime, when you are using the options **−Wformat** and **−pedantic** without gnu-extensions.

−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 **−pedantic**.

−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**.

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

−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**.

−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.

−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 will get a warning. These warnings will help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using `const` in declarations and prototypes. Otherwise, it will just be a nuisance. This is why we did not make **−Wall** request these warnings.

When compiling C++, warn about the deprecated conversion from string literals to `char *`. This warning is enabled by default for C++ programs.

−Wclobbered
Warn for variables that might be changed by **longjmp** or **vfork**. This warning is also enabled by **−Wextra**.

−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 will 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.

−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.

−Wzero−as−null−pointer−constant (C++ and Objective−C++ only)
Warn when a literal ’0’ is used as null pointer constant. This can be useful to facilitate the conversion to `nullptr` in C++11.

−Wempty−body
Warn if an empty body occurs in an **if**, **else** or **do while** statement. This warning is also enabled by **−Wextra**.

−Wenum−compare
Warn about a comparison between values of different enumerated types. In C++ enumeral mismatches in conditional expressions are also diagnosed and the warning is enabled by default. In C this warning is enabled by **−Wall**.

−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. This warning is also enabled by **−Wextra**; to get the other warnings of **−Wextra** without this warning, use **−Wextra −Wno−sign−compare**.

−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**.

−Waddress
Warn about suspicious uses of memory addresses. These include using the address of a function in a conditional expression, such as `void func(void); if (func)`, and comparisons against the memory address of a string literal, such as `if (x == "abc")`. Such uses typically indicate a programmer error: the address of a function always evaluates to true, so their use in a conditional usually indicate that the programmer forgot the parentheses in a function call; and comparisons against string literals result in unspecified behavior and are not portable in C, so they usually indicate that the programmer intended to use `strcmp`. This warning is enabled by **−Wall**.

−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.

−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−attributes
Do not warn if an unexpected `__attribute__` is used, such as unrecognized attributes, function attributes applied to variables, etc. This will not stop errors for incorrect use of supported attributes.

−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__`.

−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.

−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**.

−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. The aim is to detect global functions that are not declared in header files.

−Wmissing−declarations
Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files. 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 would cause such a warning, because `x.h` is implicitly zero:

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

This option does not warn about designated initializers, so the following modification would not trigger a warning:

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

This warning is included in **−Wextra**. To get other **−Wextra** warnings without this one, use **−Wextra −Wno−missing−field−initializers**.

−Wmissing−format−attribute
Warn about function pointers that might be candidates for `format` attributes. Note these are only possible candidates, not absolute ones. GCC will guess 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 will also warn about function definitions that might be candidates for `format` attributes. Again, these are only possible candidates. GCC will guess that `format` attributes might be appropriate for any function that calls a function like `vprintf` or `vscanf`, but this might not always be the case, and some functions for which `format` attributes are appropriate may not be detected.

−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.

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**. You would only want to do this if you were 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’’, will display 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 will warn 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 is unable to be fixed to display these characters distinctly.

−Wno−deprecated
Do not warn about usage of deprecated features.

−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.

−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**.

−Wpacked

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

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

−Wpacked−bitfield−compat

The 4.1, 4.2 and 4.3 series of GCC ignore the `packed` attribute on bit-fields of type `char`. This has been 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.

−Wpadded

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

−Wredundant−decls

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

−Wnested−externs (C and Objective-C only)

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

−Winline

Warn if a function can not be inlined and it was declared as inline. Even with this option, the compiler will 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.

−Wno−invalid−offsetof (C++ and Objective−C++ only)

Suppress warnings from applying the **offsetof** macro to a non-POD type. According to the 1998 ISO C++ standard, applying **offsetof** to a non-POD type is undefined. In existing C++ implementations, however, **offsetof** typically gives meaningful results even when applied to certain kinds of non-POD types. (Such as a simple **struct** that fails to be a POD type only by virtue of having a constructor.) 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.

−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 can't be used.

−Wlong−long
Warn if **long long** type is used. This is enabled by either **−pedantic** or **−Wtraditional** in ISO C90 and C++98 modes. To inhibit the warning messages, use **−Wno−long−long**.

−Wvariadic−macros
Warn if variadic macros are used in pedantic ISO C90 mode, or the GNU alternate syntax when in pedantic ISO C99 mode. This is default. To inhibit the warning messages, use **−Wno−variadic−macros**.

−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 variable length array is used in the code. **−Wno−vla** will prevent the **−pedantic** warning of the variable length array.

−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**.

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

−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 **−pedantic**, which can be disabled with **−Wno−pointer−sign**.

−Wstack−protector
This option is only active when **−fstack−protector** is active. It warns about functions that will not be protected against stack smashing.

−Wno−mudflap
Suppress warnings about constructs that cannot be instrumented by **−fmudflap**.

−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 **−pedantic**, and can be disabled with **−Wno−overlength−strings**.

−Wunsuffixed−float−constants (C and Objective-C only)

GCC will issue a warning for any floating constant that does not have a suffix. When used together with **−Wsystem−headers** it will warn 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.

Options for Debugging Your Program or GCC

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

−g Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF 2). GDB can work with this debugging information.

On most systems that use stabs format, **−g** enables use of extra debugging information that only GDB can use; this extra information makes debugging work better in GDB but will probably make other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use **−gstabs+**, **−gstabs**, **−gxcoff+**, **−gxcoff**, or **−gvms** (see below).

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

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

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

−ggdb

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

−gstabs

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

−feliminate−unused−debug−symbols

Produce debugging information in stabs format (if that is supported), for only symbols that are actually used.

−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 will increase the size of debugging information by as much as a factor of two.

−fno−debug−types−section

By default when using DWARF v4 or higher type DIEs will 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 sections since the linker will then be able to remove duplicates. But not all DWARF consumers support .debug_types sections yet.

−gstabs+

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

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

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

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

−gdwarf−*version*
Produce debugging information in DWARF format (if that is supported). This is the format used by DBX on IRIX 6. The value of *version* may be either 2, 3 or 4; the default version is 2.

Note that with DWARF version 2 some ports require, and will 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.

−grecord−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. See also **−frecord−gcc−switches** for another way of storing compiler options into the object file.

−gno−record−gcc−switches
Disallow appending command-line options to the DW_AT_producer attribute in DWARF debugging information. This is the default.

−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*.

−gvms
Produce debugging information in VMS debug format (if that is supported). This is the format used by DEBUG on VMS systems.

−g*level*
−ggdb*level*
−gstabs*level*
−gcoff*level*
−gxcoff*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, but no information about local variables and no line numbers.

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

−gdwarf−2 does not accept a concatenated debug level, because GCC used to support an option

−gdwarf that meant to generate debug information in version 1 of the DWARF format (which is very different from version 2), and it would have been too confusing. That debug format is long obsolete, but the option cannot be changed now. Instead use an additional **−g***level* option to change the debug level for DWARF.

−gtoggle
Turn off generation of debug info, if leaving out this option would have generated 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**.

−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 will be determined by appending `.gkd` to the compilation output file name.

−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 will reject 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 side-effect compiler outputs to files or to the standard output. 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.

−feliminate−dwarf2−dups
Compress DWARF2 debugging information by eliminating duplicated information about each symbol. This option only makes sense when generating DWARF2 debugging information with **−gdwarf−2**.

−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 was 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 2.

−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 was 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 2.

−femit−struct−debug−detailed[=*spec-list*]

Specify the struct-like types for which the compiler will generate 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 will serve 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 would be legal, 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 will 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 types declared in *foo.c* and *foo.h* will have debug information, but types declared in other header will not. 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 2.

−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 in directory *old*, record debugging information describing them as in *new* instead.

−fno−dwarf2−cfi−asm

Emit DWARF 2 unwind info as compiler generated `.eh_frame` section instead of using GAS `.cfi_*` directives.

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

−pg

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

−Q 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 about the time consumed by each pass when it finishes.

−fmem−report
Makes the compiler print some statistics about permanent memory allocation when it finishes.

−fpre−ipa−mem−report
−fpost−ipa−mem−report
Makes the compiler print some statistics about permanent memory allocation before or after interprocedural optimization.

−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.

−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. 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**).

−−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.
- 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. Also `fork` calls are detected and correctly handled (double counting will not happen).

- 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.

−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 will match the source files more closely, if you do not optimize.

−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 upper bound. *counter-value-list* is a comma-separated list of *name*:*value* pairs which sets the upper bound of each debug counter *name* to *value*. All debug counters have the initial upper bound of *UINT_MAX*, thus *dbg_cnt()* returns true always unless the upper bound is set by this option. e.g. With −fdbg−cnt=dce:10,tail_call:0 dbg_cnt(dce) will return true only for first 10 invocations

−fenable−*kind*−*pass*
−fdisable−*kind*−*pass*=*range-list*
This is a set of debugging options that are used to explicitly disable/enable optimization passes. For compiler users, regular options for enabling/disabling passes should be used 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 seperated list of function ranges or assembler names. Each range is a number pair seperated 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 cgraph node's *uid* is falling 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.

```
        # 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
```

−d*letters*
−fdump−rtl−*pass*

Says to make debugging dumps during compilation at times specified by *letters*. This is used for debugging the RTL-based passes of the compiler. The file names for most of the dumps are made by appending a pass number and a word to the *dumpname*, and the files are created in the directory of the output file. Note that the pass number is computed statically as passes get registered into the pass manager. Thus the numbering is not related to the dynamic order of execution of passes. In particular, a pass installed by a plugin could have a number over 200 even if it executed quite early. *dumpname* 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. These switches may have different effects 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 sub-expression 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−regmove
Dump after the register move pass.

−fdump−rtl−sched1
−fdump−rtl−sched2
−fdump−rtl−sched1 and **−fdump−rtl−sched2** enable dumping after the basic block scheduling passes.

−fdump−rtl−see
Dump after sign extension elimination.

−fdump−rtl−seqabstr
Dump after common sequence discovery.

−fdump−rtl−shorten
Dump after shortening branches.

−fdump−rtl−sibling
Dump after sibling call optimizations.

−fdump−rtl−split1
−fdump−rtl−split2
−fdump−rtl−split3
−fdump−rtl−split4
−fdump−rtl−split5
−fdump−rtl−split1, **−fdump−rtl−split2**, **−fdump−rtl−split3**, **−fdump−rtl−split4** and **−fdump−rtl−split5** 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−unshare
: Dump after all rtl has been unshared.

−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 was used. The length of each instruction is also printed.

−dP
: Dump the RTL in the assembler output as a comment before each instruction. Also turns on **−dp** annotation.

−dv
: For each of the other indicated dump files (**−fdump−rtl−***pass*), dump a representation of the control flow graph suitable for viewing with VCG to *file.pass.vcg*.

−dx
: Just generate RTL for a function instead of compiling it. Usually used with **−fdump−rtl−expand**.

−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.

−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−translation−unit (C++ only)
−fdump−translation−unit−*options* (C++ only)
Dump a representation of the tree structure for the entire translation unit to a file. The file name is made by appending *.tu* to the source file name, and the file is created in the same directory as the output file. If the *−options* form is used, *options* controls the details of the dump as described for the **−fdump−tree** options.

−fdump−class−hierarchy (C++ only)
−fdump−class−hierarchy−*options* (C++ only)
Dump a representation of each class's hierarchy and virtual function table layout to a file. The file name is made by appending *.class* to the source file name, and the file is created in the same directory as the output file. If the *−options* form is used, *options* controls the details of the dump as described for the **−fdump−tree** options.

−fdump−ipa−*switch*
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.

−fdump−passes
Dump 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** will cause counters to be summed over the whole compilation unit while **−details** will dump every event as the passes generate them. The default with no option is to sum counters for each function compiled.

−fdump−tree−*switch*
−fdump−tree−*switch−options*
Control the dumping at various stages of processing the intermediate 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. 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 will be 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
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.

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).

stats
Enable dumping various statistics about the pass (not honored by every dump option).

blocks
Enable showing basic block boundaries (disabled in raw dumps).

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.

all Turn on all options, except **raw**, **slim**, **verbose** and **lineno**.

The following tree dumps are possible:

original
Dump before any tree based optimization, to *file.original*.

optimized
Dump after all tree based optimization, to *file.optimized*.

gimple
Dump each function before and after the gimplification pass to a file. The file name is made by appending *.gimple* to the source file name.

cfg Dump the control flow graph of each function to a file. The file name is made by appending *.cfg* to the source file name.

vcg Dump the control flow graph of each function to a file in VCG format. The file name is made by appending *.vcg* to the source file name. Note that if the file contains more than one function, the generated file cannot be used directly by VCG. You will need to cut and paste each function's graph into its own separate file first.

ch Dump each function after copying loop headers. The file name is made by appending *.ch* to the source file name.

ssa Dump SSA related information to a file. The file name is made by appending *.ssa* to the source file name.

alias
Dump aliasing information for each function. The file name is made by appending *.alias* to the source file name.

ccp Dump each function after CCP. The file name is made by appending *.ccp* to the source file name.

storeccp
Dump each function after STORE-CCP. The file name is made by appending *.storeccp* to the source file name.

pre Dump trees after partial redundancy elimination. The file name is made by appending *.pre* to the source file name.

fre Dump trees after full redundancy elimination. The file name is made by appending *.fre* to the source file name.

copyprop
Dump trees after copy propagation. The file name is made by appending *.copyprop* to the source file name.

store_copyprop
Dump trees after store copy-propagation. The file name is made by appending *.store_copyprop* to the source file name.

dce Dump each function after dead code elimination. The file name is made by appending *.dce* to the source file name.

mudflap
Dump each function after adding mudflap instrumentation. The file name is made by appending *.mudflap* to the source file name.

sra Dump each function after performing scalar replacement of aggregates. The file name is made by appending *.sra* to the source file name.

sink
Dump each function after performing code sinking. The file name is made by appending *.sink* to the source file name.

dom
Dump each function after applying dominator tree optimizations. The file name is made by appending *.dom* to the source file name.

dse Dump each function after applying dead store elimination. The file name is made by appending *.dse* to the source file name.

phiopt
Dump each function after optimizing PHI nodes into straightline code. The file name is made by appending *.phiopt* to the source file name.

forwprop
Dump each function after forward propagating single use variables. The file name is made by appending *.forwprop* to the source file name.

copyrename
Dump each function after applying the copy rename optimization. The file name is made by appending *.copyrename* to the source file name.

nrv Dump each function after applying the named return value optimization on generic trees. The file name is made by appending *.nrv* to the source file name.

vect
Dump each function after applying vectorization of loops. The file name is made by appending *.vect* to the source file name.

slp Dump each function after applying vectorization of basic blocks. The file name is made by appending *.slp* to the source file name.

vrp Dump each function after Value Range Propagation (VRP). The file name is made by appending *.vrp* to the source file name.

all Enable all the available tree dumps with the flags provided in this option.

−ftree−vectorizer−verbose=*n*

This option controls the amount of debugging output the vectorizer prints. This information is written to standard error, unless **−fdump−tree−all** or **−fdump−tree−vect** is specified, in which case it is output to the usual dump listing file, *.vect*. For *n*=0 no diagnostic information is reported. If *n*=1 the vectorizer reports each loop that got vectorized, and the total number of loops that got vectorized. If *n*=2 the vectorizer also reports non-vectorized loops that passed the first analysis phase (vect_analyze_loop_form) − i.e. countable, inner-most, single-bb, single−entry/exit loops. This is the same verbosity level that **−fdump−tree−vect−stats** uses. Higher verbosity levels mean either more information dumped for each reported loop, or same amount of information reported for more loops: if *n*=3, vectorizer cost model information is reported. If *n*=4, alignment related information is added to the reports. If *n*=5, data-references related information (e.g. memory dependences, memory access-patterns) is added to the reports. If *n*=6, the vectorizer reports also non-vectorized inner-most loops that did not pass the first analysis phase (i.e., may not be countable, or may have complicated control-flow). If *n*=7, the vectorizer reports also non-vectorized nested loops. If *n*=8, SLP related information is added to the reports. For *n*=9, all the information the vectorizer generates during its analysis and transformation is reported. This is the same verbosity level that **−fdump−tree−vect−details** uses.

−frandom−seed=*string*

This option provides a seed that GCC uses when it would otherwise use random numbers. It is used to generate 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* should be different for every file you compile.

−fsched−verbose=*n*

On targets that use instruction scheduling, this option controls the amount of debugging output the scheduler prints. This information is written to standard error, unless **−fdump−rtl−sched1** or **−fdump−rtl−sched2** is specified, in which case it is output to the usual dump listing file, *.sched1* or *.sched2* respectively. However for *n* greater than nine, the output is always printed to standard error.

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.

−save−temps
−save−temps=cwd

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

When used in combination with the **−x** command-line option, **−save−temps** is sensible enough to avoid over writing 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**.

If you invoke GCC in parallel, compiling several different source files that share a common base name in different subdirectories or the same source file compiled for multiple output destinations, it is likely that the different parallel compilers will interfere with each other, and overwrite the temporary files. For instance:

```
gcc −save−temps −o outdir1/foo.o indir1/foo.c&
gcc −save−temps −o outdir2/foo.o indir2/foo.c&
```

may result in *foo.i* and *foo.o* being written to simultaneously by both compilers.

−save−temps=obj

Store the usual ‘‘temporary’’ intermediate files permanently. If the **−o** option is used, the temporary files are based on the object file. If the **−o** option is not used, the **−save−temps=obj** switch behaves like **−save−temps**.

For example:

```
        gcc −save−temps=obj −c foo.c
        gcc −save−temps=obj −c bar.c −o dir/xbar.o
        gcc −save−temps=obj foobar.c −o dir2/yfoobar
```

would create *foo.i, foo.s, dir/xbar.i, dir/xbar.s, dir2/yfoobar.i, dir2/yfoobar.s,* and *dir2/yfoobar.o*.

−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.

−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 will be created and maintained, but discarded at the end.

−fvar−tracking−assignments−toggle

Toggle **−fvar−tracking−assignments**, in the same way that **−gtoggle** toggles **−g**.

−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** will search−−−and don't do anything else.

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

−print−sysroot
Print the target sysroot directory that will be 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**)−−−and don't do anything else.

−dumpspecs
Print the compiler's built-in specs−−−and don't do anything else. (This is used when GCC itself is being built.)

−feliminate−unused−debug−types
Normally, when producing DWARF2 output, GCC will emit debugging information for all types declared in a compilation unit, regardless of whether or not they are actually used in that compilation unit. Sometimes this is useful, such as 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. With this option, GCC will avoid producing debug symbol output for types that are nowhere used in the source file being compiled.

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 would 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 only enabled if an **−O** level is set on the command line. Otherwise they are disabled, even if individual optimization flags are specified.

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 turns on the following optimization flags:

−fauto−inc−dec −fcompare−elim −fcprop−registers −fdce −fdefer−pop −fdelayed−branch −fdse −fguess−branch−probability −fif−conversion2 −fif−conversion −fipa−pure−const −fipa−profile −fipa−reference −fmerge−constants −fsplit−wide−types −ftree−bit−ccp −ftree−builtin−call−dce −ftree−ccp −ftree−ch −ftree−copyrename −ftree−dce −ftree−dominator−opts −ftree−dse −ftree−forwprop −ftree−fre −ftree−phiprop −ftree−sra −ftree−pta −ftree−ter −funit−at−a−time

−O also turns on **−fomit−frame−pointer** on machines where doing so does not interfere with debugging.

−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 **−O**. It also turns on the following optimization flags: **−fthread−jumps −falign−functions −falign−jumps −falign−loops −falign−labels −fcaller−saves −fcrossjumping −fcse−follow−jumps −fcse−skip−blocks −fdelete−null−pointer−checks −fdevirtualize −fexpensive−optimizations −fgcse −fgcse−lm −finline−small−functions −findirect−inlining −fipa−sra −foptimize−sibling−calls −fpartial−inlining −fpeephole2 −fregmove −freorder−blocks −freorder−functions −frerun−cse−after−loop −fsched−interblock −fsched−spec −fschedule−insns −fschedule−insns2 −fstrict−aliasing −fstrict−overflow −ftree−switch−conversion −ftree−tail−merge −ftree−pre −ftree−vrp**

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 **−finline−functions**, **−funswitch−loops**, **−fpredictive−commoning**, **−fgcse−after−reload**, **−ftree−vectorize** and **−fipa−cp−clone** options.

−O0

Reduce compilation time and make debugging produce the expected results. This is the default.

−Os

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

−Os disables the following optimization flags: **−falign−functions −falign−jumps −falign−loops −falign−labels −freorder−blocks −freorder−blocks−and−partition −fprefetch−loop−arrays −ftree−vect−loop−version**

−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** and the Fortran-specific **−fno−protect−parens** and **−fstack−arrays**.

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

Options of the form **−f***flag* specify machine-independent flags. Most flags have both positive and negative forms; the negative form of **−ffoo** would be **−fno−foo**. In the table below, only one of the forms is listed−−−the one you typically will 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−default−inline

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

−fno−defer−pop

Always pop the arguments to each function call as soon as that function returns. For machines that must pop arguments after a function call, the compiler normally lets arguments accumulate on the stack for several function calls and pops them all at once.

Disabled at levels **−O**, **−O2**, **−O3**, **−Os**.

−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 **−O**, **−O2**, **−O3**, **−Os**.

−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 currently not implemented and treated equal to **−ffp−contract=off**.

The default is **−ffp−contract=fast**.

−fomit−frame−pointer

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

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

Starting with GCC version 4.6, the default setting (when not optimizing for size) for 32−bit Linux x86 and 32−bit Darwin x86 targets has been changed to **−fomit−frame−pointer**. The default can be reverted to **−fno−omit−frame−pointer** by configuring GCC with the **−−enable−frame−pointer** configure option.

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

−foptimize−sibling−calls
Optimize sibling and tail recursive calls.

Enabled at levels **−O2**, **−O3**, **−Os**.

−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 level **−O2**.

−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 level **−O2**.

−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 level **−O3**.

−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**.

−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−consts

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

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

−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 **−O**, **−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 will result 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 will be deleted which will trigger the generation of reg-moves based on the life-range analysis. This option is effective only with **−fmodulo−sched** enabled.

−fno−branch−count−reg
Do not use ''decrement and branch'' instructions on a count register, but instead generate a sequence of instructions that decrement a register, compare it against zero, 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.

The default is **−fbranch−count−reg**.

−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**.

−fmudflap −fmudflapth −fmudflapir
For front-ends that support it (C and C++), instrument all risky pointer/array dereferencing operations, some standard library string/heap functions, and some other associated constructs with range/validity tests. Modules so instrumented should be immune to buffer overflows, invalid heap use, and some other classes of C/C++ programming errors. The instrumentation relies on a separate runtime library (*libmudflap*), which will be linked into a program if **−fmudflap** is given at link time. Run-time behavior of the instrumented program is controlled by the **MUDFLAP_OPTIONS** environment variable. See `env MUDFLAP_OPTIONS=-help a.out` for its options.

Use **−fmudflapth** instead of **−fmudflap** to compile and to link if your program is multi-threaded. Use **−fmudflapir**, in addition to **−fmudflap** or **−fmudflapth**, if instrumentation should ignore pointer reads. This produces less instrumentation (and therefore faster execution) and still provides some protection against outright memory corrupting writes, but allows erroneously read data to propagate within a program.

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

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

−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 will follow 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 has been 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 will attempt 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 gcse is enabled.

−fgcse−sm
When **−fgcse−sm** is enabled, a store motion pass is run after global common subexpression elimination. This pass will attempt 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 cleanup redundant spilling.

−funsafe−loop−optimizations
If given, the loop optimizer will assume that loop indices do not overflow, and that the loops with nontrivial exit condition are not infinite. This enables a wider range of loop optimizations even if the loop optimizer itself cannot prove that these assumptions are valid. Using **−Wunsafe−loop−optimizations**, the compiler will warn you if it finds this kind of loop.

−fcrossjumping
Perform cross-jumping transformation. This transformation unifies equivalent code and save 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 **−O** and higher on architectures that support this.

−fdce
Perform dead code elimination (DCE) on RTL. Enabled by default at **−O** and higher.

−fdse
Perform dead store elimination (DSE) on RTL. Enabled by default at **−O** and higher.

−fif−conversion
Attempt to transform conditional jumps into branch-less equivalents. This include 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 `if−conversion2`.

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

−fif−conversion2
Use conditional execution (where available) to transform conditional jumps into branch-less equivalents.

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

−fdelete−null−pointer−checks
Assume that programs cannot safely dereference null pointers, and that no code or data element resides there. This 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 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.

Some targets, especially embedded ones, disable this option at all levels. Otherwise it is enabled at all levels: **−O0**, **−O1**, **−O2**, **−O3**, **−Os**. Passes that use the 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**.

−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 x86 at levels **−O2**, **−O3**.

−foptimize−register−move

−fregmove

Attempt to reassign register numbers in move instructions and as operands of other simple instructions in order to maximize the amount of register tying. This is especially helpful on machines with two-operand instructions.

Note **−fregmove** and **−foptimize−register−move** are the same optimization.

Enabled at levels **−O2**, **−O3**, **−Os**.

−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−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.

−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.

−fdelayed−branch

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

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

−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.

Enabled at levels **−O2**, **−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

Don't schedule instructions across basic blocks. This is normally enabled by default when scheduling before register allocation, i.e. with **−fschedule−insns** or at **−O2** or higher.

−fno−sched−spec

Don't allow speculative motion of non-load instructions. This is normally enabled by default when scheduling before register allocation, i.e. with **−fschedule−insns** or at **−O2** or higher.

−fsched−pressure

Enable register pressure sensitive insn scheduling before the 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 as a consequence register spills in the 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 will be 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) will be examined for a dependency on a stalled insn that is 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, do use superblock scheduling algorithm. Superblock scheduling allows motion across basic block boundaries resulting on 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
The modulo scheduling comes before the traditional scheduling, if a loop was modulo scheduled we may want to prevent the later scheduling passes from changing its schedule, we use this option to control that.

−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 until 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 until **−fsel−sched−pipelining** is turned on.

−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.

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

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

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.

−fconserve−stack
Attempt to minimize stack usage. The compiler will attempt 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 **−O** and higher.

−ftree−pre
Perform partial redundancy elimination (PRE) on trees. This flag is enabled by default at **−O2** and **−O3**.

−ftree−forwprop
Perform forward propagation on trees. This flag is enabled by default at **−O** 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 **−O** and higher.

−ftree−phiprop
Perform hoisting of loads from conditional pointers on trees. This pass is enabled by default at **−O** and higher.

−ftree−copy−prop
Perform copy propagation on trees. This pass eliminates unnecessary copy operations. This flag is enabled by default at **−O** and higher.

−fipa−pure−const
Discover which functions are pure or constant. Enabled by default at **−O** and higher.

−fipa−reference
Discover which static variables do not escape cannot escape the compilation unit. Enabled by default at **−O** and higher.

−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 **−O** 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**.

−fipa−cp−clone
Perform function cloning to make interprocedural constant propagation stronger. When enabled, interprocedural constant propagation will perform 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 ipcp−unit−growth=***value*). This flag is enabled by default at **−O3**.

−fipa−matrix−reorg
Perform matrix flattening and transposing. Matrix flattening tries to replace an m−dimensional matrix with its equivalent n−dimensional matrix, where n < m. This reduces the level of indirection needed for accessing the elements of the matrix. The second optimization is matrix transposing, which attempts to change the order of the matrix's dimensions in order to improve cache locality. Both optimizations need the **−fwhole−program** flag. Transposing is enabled only if profiling information is available.

−ftree−sink
Perform forward store motion on trees. This flag is enabled by default at **−O** 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 **−O** and higher. 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 **−O** and higher.

−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−dce
Perform dead code elimination (DCE) on trees. This flag is enabled by default at **−O** and higher.

−ftree−builtin−call−dce
Perform conditional dead code elimination (DCE) for calls to builtin functions that may set `errno` but are otherwise side-effect free. This flag is enabled by default at **−O2** and higher if **−Os** is not also specified.

−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 **−O** 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 **−O** 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 **−O** 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 **−O** and higher.

−ftree−loop−linear
Perform loop interchange transformations on tree. Same as **−floop−interchange**. To use this code transformation, GCC has to be configured with **−−with−ppl** and **−−with−cloog** to enable the Graphite loop transformation infrastructure.

−floop−interchange

Perform loop interchange transformations on loops. Interchanging two nested loops switches the inner and outer loops. For example, given a loop like:

```
DO J = 1, M
  DO I = 1, N
    A(J, I) = A(J, I) * C
  ENDDO
ENDDO
```

loop interchange will transform the loop as if the user had written:

```
DO I = 1, N
  DO J = 1, M
    A(J, I) = A(J, I) * C
  ENDDO
ENDDO
```

which can be beneficial when `N` is larger than the caches, because in Fortran, the elements of an array are stored in memory contiguously by column, and the original loop iterates over rows, potentially creating at each access a cache miss. This optimization applies to all the languages supported by GCC and is not limited to Fortran. To use this code transformation, GCC has to be configured with **−−with−ppl** and **−−with−cloog** to enable the Graphite loop transformation infrastructure.

−floop−strip−mine

Perform loop strip mining transformations on loops. Strip mining splits a loop into two nested loops. The outer loop has strides equal to the strip size and the inner loop has strides of the original loop within a strip. The strip length can be changed using the **loop-block-tile-size** parameter. For example, given a loop like:

```
DO I = 1, N
  A(I) = A(I) + C
ENDDO
```

loop strip mining will transform the loop as if the user had written:

```
DO II = 1, N, 51
  DO I = II, min (II + 50, N)
    A(I) = A(I) + C
  ENDDO
ENDDO
```

This optimization applies to all the languages supported by GCC and is not limited to Fortran. To use this code transformation, GCC has to be configured with **−−with−ppl** and **−−with−cloog** to enable the Graphite loop transformation infrastructure.

−floop−block

Perform loop blocking transformations on loops. Blocking strip mines each loop in the loop nest such that the memory accesses of the element loops fit inside caches. The strip length can be changed using the **loop-block-tile-size** parameter. For example, given a loop like:

```
DO I = 1, N
  DO J = 1, M
    A(J, I) = B(I) + C(J)
  ENDDO
ENDDO
```

loop blocking will transform the loop as if the user had written:

```
DO II = 1, N, 51
  DO JJ = 1, M, 51
    DO I = II, min (II + 50, N)
      DO J = JJ, min (JJ + 50, M)
        A(J, I) = B(I) + C(J)
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

which can be beneficial when `M` is larger than the caches, because the innermost loop will iterate over a smaller amount of data which can be kept in the caches. This optimization applies to all the languages supported by GCC and is not limited to Fortran. To use this code transformation, GCC has to be configured with **−−with−ppl** and **−−with−cloog** 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 CLooG, like index splitting and dead code elimination in loops.

−floop−flatten
Removes the loop nesting structure: transforms the loop nest into a single loop. This transformation can be useful as an enablement transform for vectorization and parallelization. This feature is experimental. To use this code transformation, GCC has to be configured with **−−with−ppl** and **−−with−cloog** to enable the Graphite loop transformation infrastructure.

−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.

−fcheck−data−deps
Compare the results of several data dependence analyzers. This option is used for debugging the data dependence analyzers.

−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−if−convert−stores
Attempt to also if-convert conditional jumps containing memory writes. This transformation can be unsafe for multi-threaded programs as it transforms conditional memory writes into unconditional memory writes. For example,

```
for (i = 0; i < N; i++)
  if (cond)
    A[i] = expr;
```

would be transformed to

```
for (i = 0; i < N; i++)
  A[i] = cond ? expr : A[i];
```

potentially producing data races.

−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
```

−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 **−O3**.

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.

−ftree−loop−im
Perform loop invariant motion on trees. This pass moves only invariants that would be 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.

−fivopts
Perform induction variable optimizations (strength reduction, induction variable merging and induction variable elimination) on trees.

−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 **−O** and higher.

−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 **−O** and higher.

−ftree−copyrename
Perform copy renaming on trees. This pass attempts to rename compiler temporaries to other variables at copy locations, usually resulting in variable names which more closely resemble the original variables. This flag is enabled by default at **−O** and higher.

−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 **−O** and higher.

−ftree−vectorize
Perform loop vectorization on trees. This flag is enabled by default at **−O3**.

−ftree−slp−vectorize
Perform basic block vectorization on trees. This flag is enabled by default at **−O3** and when **−ftree−vectorize** is enabled.

−ftree−vect−loop−version
Perform loop versioning when doing loop vectorization on trees. When a loop appears to be vectorizable except that data alignment or data dependence cannot be determined at compile time, then vectorized and non-vectorized versions of the loop are generated along with run-time checks for alignment or dependence to control which version is executed. This option is enabled by default except at level **−Os** where it is disabled.

−fvect−cost−model
Enable cost model for vectorization.

−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.

−ftracer
Perform tail duplication to enlarge superblock size. This transformation simplifies the control flow of the function allowing other optimizations to do better job.

−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**. This option makes code larger, and may or may not make it run faster.

−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**,

−fsplit−ivs−in−unroller
Enables expressing 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.

Combination of **−fweb** and CSE is often sufficient to obtain the same effect. However in cases the loop body is more complicated than a single basic block, this is not reliable. It also does not work at all on some of the architectures due to restrictions in the CSE pass.

This optimization is enabled by default.

−fvariable−expansion−in−unroller
With this option, the compiler will create multiple copies of some local variables when unrolling a loop which can result in superior code.

−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 level **−O2**.

−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**.

−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−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 will use 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 will be 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.

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 **−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, for linkonce sections, for functions with a user-defined section attribute and on any architecture that does not support named sections.

−freorder−functions
Reorder functions in the object file in order to improve code locality. 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.

Also profile feedback must be available in to make this option effective. See **−fprofile−arcs** for details.

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 will work 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**.

−fstrict−overflow

Allow the compiler to assume strict signed overflow rules, depending on the language being compiled. For C (and C++) this means that overflow when doing arithmetic with signed numbers is undefined, which means that the compiler may assume that it will not happen. This permits various optimizations. For example, the compiler will assume that an expression like `i + 10 > i` will always be true for signed `i`. This assumption is only valid if signed overflow is undefined, as the expression is false if `i + 10` overflows when using twos complement arithmetic. When this option is in effect any attempt to determine whether an operation on signed numbers will overflow must be written carefully to not actually involve overflow.

This option also allows the compiler to assume strict pointer semantics: given a pointer to an object, if adding an offset to that pointer does not produce a pointer to the same object, the addition is undefined. This permits the compiler to conclude that `p + u > p` is always true for a pointer `p` and unsigned integer `u`. This assumption is only valid because pointer wraparound is undefined, as the expression is false if `p + u` overflows using twos complement arithmetic.

See also the **−fwrapv** option. Using **−fwrapv** means that integer signed overflow is fully defined: it wraps. When **−fwrapv** is used, there is no difference between **−fstrict−overflow** and **−fno−strict−overflow** for integers. With **−fwrapv** certain types of overflow are permitted. For example, if the compiler gets an overflow when doing arithmetic on constants, the overflowed value can still be used with **−fwrapv**, but not otherwise.

The **−fstrict−overflow** option is enabled at levels **−O2**, **−O3**, **−Os**.

−falign−functions
−falign−functions=*n*

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

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

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

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

Enabled at levels **−O2**, **−O3**.

−falign−labels
−falign−labels=*n*

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

−fno−align−labels and **−falign−labels=1** are equivalent and mean that labels will not be 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.

Enabled at levels **−O2**, **−O3**.

−falign−loops
−falign−loops=*n*

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

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

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

Enabled at levels **−O2**, **−O3**.

−falign−jumps
−falign−jumps=*n*

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

−fno−align−jumps and **−falign−jumps=1** are equivalent and mean that loops will not be aligned.

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

Enabled at levels **−O2**, **−O3**.

−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 will not be removed. This option is intended to support existing code that relies on a particular ordering. For new code, it is better to use attributes.

Enabled at level **−O0**. When disabled explicitly, it also implies **−fno−section−anchors**, which is otherwise enabled at **−O0** on some targets.

−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 will 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. If **gold** is used as the linker plugin, `externally_visible` attributes are automatically added to functions (not variable yet due to a current **gold** issue) that are accessed outside of LTO objects according to resolution file produced by **gold**. For other linkers that cannot generate resolution file, explicit `externally_visible` attributes are still necessary. While this option is equivalent to proper use of the `static` keyword for programs consisting of a single file, in combination with option **−flto** this flag can be used to compile many smaller scale programs since the functions and variables become local for the whole combined compilation unit, not for the single source file itself.

This option implies **−fwhole−file** for Fortran programs.

−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** needs to be specified at compile time and during the final link. 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 only important thing to keep in mind is that to enable link-time optimizations the **−flto** flag needs to be passed to both the compile and the link commands.

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.

Note that when a file is compiled with **−flto**, the generated object file is larger than a regular object file because it contains GIMPLE bytecodes and the usual final code. This means that object files with LTO information can be linked as normal object files; if **−flto** is not passed to the linker, no interprocedural optimizations are applied.

Additionally, the optimization flags used to compile individual files are not necessarily related to those used at link time. For instance,

```
        gcc -c -O0 -flto foo.c
        gcc -c -O0 -flto bar.c
        gcc -o myprog -flto -O3 foo.o bar.o
```

This produces individual object files with unoptimized assembler code, but the resulting binary *myprog* is optimized at **−O3**. If, instead, the final binary is generated without **−flto**, then *myprog* is not optimized.

When producing the final binary with **−flto**, 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.

There are some code generation flags preserved by GCC when generating bytecodes, as they need to be used during the final link stage. Currently, the following options are saved into the GIMPLE bytecode files: **−fPIC**, **−fcommon** and all the **−m** target flags.

At link time, these options are read in and reapplied. Note that the current implementation makes no attempt to recognize conflicting values for these options. If different files have conflicting option values (e.g., one file is compiled with **−fPIC** and another isn't), the compiler simply uses the last value read from the bytecode files. It is recommended, then, that you compile all the files participating in the same link with the same options.

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.

Another feature of LTO is that it is possible to apply interprocedural optimizations on files written in different languages. This requires support in the language front end. Currently, the C, C++ and Fortran front ends are capable of emitting GIMPLE bytecodes, so something like this should work:

```
        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; all you need to add is **−flto** to all the compile and link commands.

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 enable this feature, use the flag **−fuse−linker−plugin** at link time:

```
        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.

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 will not work with an older/newer version of GCC.

Link-time optimization does not work well with generation of debugging information. Combining **−flto** with **−g** is currently experimental and expected to produce wrong results.

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. The default value for *n* is 1.

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.

This option is disabled by default

−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). Specifying `none` as an algorithm disables partitioning and streaming completely. The default value is `balanced`.

−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**). Valid values are 0 (no compression) to 9 (maximum compression). Values outside this range are clamped to either 0 or 9. If the option is not given, a default balanced compression setting is used.

−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.

−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).

The default is **−ffat−lto−objects** but this default is intended to change in future releases when linker plugin enabled environments become more common.

−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 **−O**, **−O2**, **−O3**, **−Os**.

−fcprop−registers
After register allocation and post-register allocation instruction splitting, we perform a copy-propagation pass to try to reduce scheduling dependencies and occasionally eliminate the copy.

Enabled at levels **−O**, **−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 will use heuristics to correct or smooth out such inconsistencies. By default, GCC will emit an error message when an inconsistent profile is detected.

−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 will use the current directory as *path*, thus the profile data file will appear in the same directory as the object file.

−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`, `−fvpt`.

If *path* is specified, GCC will look at the *path* to find the profile feedback data files. See **−fprofile−dir**.

−fprofile−use
−fprofile−use=*path*

Enable profile feedback directed optimizations, and optimizations generally profitable only with profile feedback available.

The following options are enabled: `−fbranch−probabilities`, `−fvpt`, `−funroll−loops`, `−fpeel−loops`, `−ftracer`

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 **−Wcoverage−mismatch**. Note this may result in poorly optimized code.

If *path* is specified, GCC will look at the *path* to find the profile feedback data files. See **−fprofile−dir**.

The following options control compiler behavior regarding floating-point arithmetic. These options trade off between speed and correctness. All must be specifically enabled.

−ffloat−store

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

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

−fexcess−precision=*style*

This option allows further control over excess precision on machines where floating-point registers have more precision than the IEEE `float` and `double` types and the processor does not support operations rounding to those types. By default, **−fexcess−precision=fast** is in effect; this means that operations are carried out in the precision of the registers and that it is unpredictable when rounding to the types specified in the source code takes place. When compiling C, if **−fexcess−precision=standard** is specified then excess precision will follow the rules specified in ISO C99; 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 if a strict conformance option such as **−std=c99** is used.

−fexcess−precision=standard is not implemented for languages other than C, and has no effect if **−funsafe−math−optimizations** or **−ffast−math** is specified. On the x86, it also 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.

−ffast−math

Sets **−fno−math−errno**, **−funsafe−math−optimizations**, **−ffinite−math−only**, **−fno−rounding−math**, **−fno−signaling−nans** and **−fcx−limited−range**.

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 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 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 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 should never be turned on by any **−O** option 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**.

−frounding−math

Disable transformations and optimizations that assume default floating-point rounding behavior. This is round-to-zero for all floating point to integer conversions, and round-to-nearest for all other arithmetic truncations. 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 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.

−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 the 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.

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.c*, 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.

−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 with **−fprofile−generate** and **−fprofile−use**.

−fvpt

If combined with **−fprofile−arcs**, it instructs the compiler to add a 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 operation using the knowledge about the value of the denominator.

−frename−registers

Attempt to avoid false dependencies in scheduled code by making use of registers left over after register allocation. This optimization will most benefit processors with lots of registers. Depending on the debug information format adopted by the target, however, it can make debugging impossible, since variables will no longer stay in a ‘‘home register’’.

Enabled by default with **−funroll−loops** and **−fpeel−loops**.

−ftracer

Perform tail duplication to enlarge superblock size. This transformation simplifies the control flow of the function allowing other optimizations to do better job.

Enabled with **−fprofile−use**.

−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 small constant number of iterations). This option makes code larger, and may or may not make it run faster.

Enabled with **−fprofile−use**.

−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). It also turns on complete loop peeling (i.e. complete removal of loops with small constant number of iterations).

Enabled with **−fprofile−use**.

−fmove−loop−invariants
Enables the loop invariant motion pass in the RTL loop optimizer. Enabled at level **−O1**

−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).

−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 and SPARC processors running Solaris 2 have linkers with such optimizations. AIX may have these optimizations in the future.

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

−fbranch−target−load−optimize
Perform branch target register load optimization before prologue / epilogue threading. The use of target registers can typically be exposed only during reload, thus hoisting loads out of loops and doing inter-block scheduling needs a separate optimization pass.

−fbranch−target−load−optimize2
Perform branch target register load optimization after prologue / epilogue threading.

−fbtr−bb−exclusive
When performing branch target register load optimization, don't reuse branch target registers in within any basic block.

−fstack−protector
Emit extra code to check for buffer overflows, such as stack smashing attacks. This is done by adding a guard variable to functions with vulnerable objects. This includes functions that call alloca, and functions with buffers larger than 8 bytes. The guards are initialized when a function is entered and then checked when the function exits. If a guard check fails, an error message is printed and the program exits.

−fstack−protector−all
Like **−fstack−protector** except that all functions are protected.

−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; }
```

would usually calculate the addresses of all three variables, but if you compile it with **−fsection−anchors**, it will access 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.

−−param *name*=*value*

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

The names of specific parameters, and the meaning of the values, are tied to the internals of the compiler, and are subject to change without notice in future releases.

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

predictable-branch-outcome

When branch is predicted to be taken with probability lower than this threshold (in percent), then it is considered well predictable. The default is 10.

max-crossjump-edges

The maximum number of incoming edges to consider for crossjumping. The algorithm used by **−fcrossjumping** is O(Nˆ2) 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 crossjumping will be performed on them. This value is ignored in the case where all instructions in the block being crossjumped from are matched. The default value is 5.

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. The default value is 8.

max-goto-duplication-insns

The maximum number of instructions to duplicate to a block that jumps to a computed goto. To avoid O(Nˆ2) 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. The default value is 8.

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 is searched, the time savings from filling the delay slot will be 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 that will be allocated in order to perform the global common subexpression elimination optimization. If more memory than specified is required, the optimization will not be done.

max-gcse-insertion-ratio
If the ratio of expression insertions to deletions is larger than this value for any expression, then RTL PRE will insert or remove the expression and thus leave partially redundant computations in the instruction stream. The default value is 20.

max-pending-list-length
The maximum number of pending dependencies scheduling will allow 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-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 will consider for inlining. This only affects functions declared inline and methods implemented in a class declaration (C++). The default value is 400.

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 will be investigated. To those functions, a different (more restrictive) limit compared to functions declared inline can be applied. The default value is 40.

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. The default value is 2700.

large-function-growth
Specifies maximal growth of large function caused by inlining in percents. The default value is 100 which 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 (consider unit consisting of function A that is inline and B that just calls A three time. 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**. The default is 10000

inline-unit-growth
Specifies maximal overall growth of the compilation unit caused by inlining. The default value is 30 which limits unit growth to 1.3 times the original size.

ipcp-unit-growth
Specifies maximal overall growth of the compilation unit caused by interprocedural constant propagation. The default value is 10 which limits unit growth to 1.1 times the original size.

large-stack-frame
The limit specifying large stack frames. While inlining the algorithm is trying to not grow past this limit too much. Default value is 256 bytes.

large-stack-frame-growth
Specifies maximal growth of large stack frames caused by inlining in percents. The default value is 1000 which limits large stack frame growth to 11 times the original size.

max-inline-insns-recursive
max-inline-insns-recursive-auto

Specifies maximum number of instructions out-of-line copy of self recursive inline function can grow into by performing recursive inlining.

For functions declared inline **−−param max-inline-insns-recursive** is taken into account. For function not declared inline, recursive inlining happens only when **−finline−functions** (included in **−O3**) is enabled and **−−param max-inline-insns-recursive-auto** is used. The default value is 450.

max-inline-recursive-depth
max-inline-recursive-depth-auto

Specifies maximum recursion depth used by the recursive inlining.

For functions declared inline **−−param max-inline-recursive-depth** is taken into account. For function not declared inline, recursive inlining happens only when **−finline−functions** (included in **−O3**) is enabled and **−−param max-inline-recursive-depth-auto** is used. The default value is 8.

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 probability that function will recurse via given call expression. This parameter limits inlining only to call expression whose probability exceeds given threshold (in percents). The default value is 10.

early-inlining-insns

Specify growth that early inliner can make. In effect it increases amount of inlining for code having large abstraction penalty. The default value is 10.

max-early-inliner-iterations
max-early-inliner-iterations

Limit of iterations of early inliner. This basically bounds number of nested indirect calls early inliner can resolve. Deeper chains are still handled by late inlining.

comdat-sharing-probability
comdat-sharing-probability

Probability (in percent) that C++ inline function with comdat visibility will be shared across multiple compilation units. The default value is 20.

min-vect-loop-bound

The minimum number of iterations under which a loop will not get 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. The default value is 0.

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 will be with simple expressions, i.e., the expressions that have cost less than **gcse-unrestricted-cost**. Specifying 0 will disable hoisting of simple expressions. The default value is 10.

gcse-unrestricted-cost

Cost, roughly measured as the cost of a single typical machine instruction, at which GCSE optimizations will 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 will be. Specifying 0 will allow all expressions to travel unrestricted distances. The default value is 3.

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 will avoid limiting the search, but may slow down compilation of huge functions. The default value is 30.

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. The default value is 10.

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. The default value is 2.

max-unrolled-insns
The maximum number of instructions that a loop should have if that loop is unrolled, and if the loop is unrolled, it 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 should have if that loop is unrolled, and if the loop is unrolled, it 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 should have if that loop is peeled, and if the loop is peeled, it determines how many times the loop code is peeled.

max-peel-times
The maximum number of peelings of a single loop.

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-level
The maximum number of branches unswitched in a single loop.

lim-expensive
The minimum cost of an expensive expression in the loop invariant motion.

iv-consider-all-candidates-bound
Bound on number of candidates for induction variables below that all candidates are considered for each use in induction variable optimizations. Only the most relevant candidates are considered if there are more candidates, 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 number of candidates in the set is smaller than this value, we always try to remove unnecessary ivs from the set during its optimization when a new iv is added to the set.

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.

omega-max-vars
The maximum number of variables in an Omega constraint system. The default value is 128.

omega-max-geqs
The maximum number of inequalities in an Omega constraint system. The default value is 256.

omega-max-eqs
The maximum number of equalities in an Omega constraint system. The default value is 128.

omega-max-wild-cards
The maximum number of wildcard variables that the Omega solver will be able to insert. The default value is 18.

omega-hash-table-size
The size of the hash table in the Omega solver. The default value is 550.

omega-max-keys
The maximal number of keys used by the Omega solver. The default value is 500.

omega-eliminate-redundant-constraints
When set to 1, use expensive methods to eliminate all redundant constraints. The default value is 0.

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. See option ftree-vect-loop-version for more information.

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. See option ftree-vect-loop-version for more information.

max-iterations-to-track
The maximum number of iterations of a loop the brute force algorithm for analysis of # of iterations of the loop tries to evaluate.

hot-bb-count-fraction
Select fraction of the maximal count of repetitions of basic block in program given basic block needs to have to be considered hot.

hot-bb-frequency-fraction
Select fraction of the entry block frequency of executions of basic block in function given basic block needs to have to be considered hot.

max-predicted-iterations
The maximum number of loop iterations we predict statically. This is useful in cases where function contain single loop with known bound and other loop with unknown. We predict the known number of iterations correctly, while the unknown number of iterations average to roughly 10. This means that the loop without bounds would appear artificially cold relative to the other one.

align-threshold
Select fraction of the maximal frequency of executions of basic block in function given basic block will get aligned.

align-loop-iterations
A loop expected to iterate at lest the selected number of iterations will get 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** 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 rather hokey argument, as most of the duplicates will be 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-ratio
tracer-min-branch-ratio-feedback
Stop forward growth if the best edge do have probability lower than this threshold.

Similarly to **tracer-dynamic-coverage** two values are present, one for compilation for profile feedback and one for compilation without. The value for compilation with profile feedback needs to be more conservative (higher) in order to make tracer effective.

max-cse-path-length
Maximum number of basic blocks on path that cse considers. The default is 10.

max-cse-insns
The maximum instructions CSE process before flushing. The default is 1000.

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. The default value is 100.

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. The default value is 500.

reorder-blocks-duplicate
reorder-blocks-duplicate-feedback
Used by basic block reordering pass to decide whether to use unconditional branch or duplicate the code on its destination. Code is duplicated when its estimated size is smaller than this value multiplied by the estimated size of unconditional jump in the hot spots of the program.

The **reorder-block-duplicate-feedback** is used only when profile feedback is available and may be set to higher values than **reorder-block-duplicate** since information about the hot spots is more accurate.

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. The default value is 100.

max-sched-region-blocks
The maximum number of blocks in a region to be considered for interblock scheduling. The default value is 10.

max-pipeline-region-blocks
The maximum number of blocks in a region to be considered for pipelining in the selective scheduler. The default value is 15.

max-sched-region-insns
The maximum number of insns in a region to be considered for interblock scheduling. The default value is 100.

max-pipeline-region-insns
The maximum number of insns in a region to be considered for pipelining in the selective scheduler. The default value is 200.

min-spec-prob
The minimum probability (in percents) of reaching a source block for interblock speculative scheduling. The default value is 40.

max-sched-extend-regions-iters
The maximum number of iterations through CFG to extend regions. 0 − disable region extension, N − do at most N iterations. The default value is 0.

max-sched-insn-conflict-delay
The maximum conflict delay for an insn to be considered for speculative motion. The default value is 3.

sched-spec-prob-cutoff
The minimal probability of speculation success (in percents), so that speculative insn will be scheduled. The default value is 40.

sched-mem-true-dep-cost
Minimal distance (in CPU cycles) between store and load targeting same memory locations. The default value is 1.

selsched-max-lookahead
The maximum size of the lookahead window of selective scheduling. It is a depth of search for available instructions. The default value is 50.

selsched-max-sched-times
The maximum number of times that an instruction will be scheduled during selective scheduling. This is the limit on the number of iterations through which the instruction may be pipelined. The

default value is 2.

selsched-max-insns-to-rename
The maximum number of best instructions in the ready list that are considered for renaming in the selective scheduler. The default value is 2.

sms-min-sc
The minimum value of stage count that swing modulo scheduler will generate. The default value is 2.

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. The default is 10000.

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. The default value is 256.

min-virtual-mappings
Specifies the minimum number of virtual mappings in the incremental SSA updater that should be registered to trigger the virtual mappings heuristic defined by virtual-mappings-ratio. The default value is 100.

virtual-mappings-ratio
If the number of virtual mappings is virtual-mappings-ratio bigger than the number of virtual symbols to be updated, then the incremental SSA updater switches to a full update for those symbols. The default ratio is 3.

ssp-buffer-size
The minimum size of buffers (i.e. arrays) that will receive stack smashing protection when **−fstack−protection** is used.

max-jump-thread-duplication-stmts
Maximum number of statements allowed in a block that needs to be duplicated when threading jumps.

max-fields-for-field-sensitive
Maximum number of fields in a structure we will treat in a field sensitive manner during pointer analysis. The default is zero for −O0, and −O1 and 100 for −Os, −O2, and −O3.

prefetch-latency
Estimate on average number of instructions that are executed before prefetch finishes. The distance we prefetch 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 cache, in bytes.

l1−cache−size
The size of L1 cache, in kilobytes.

l2−cache−size
The size of L2 cache, in kilobytes.

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. By default, this 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 will refuse to create arrays that are bigger than **switch-conversion-max-branch-ratio** times the number of branches in the switch.

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 will allow an unlimited set length.

sccvn-max-scc-size
Maximum size of a strongly connected component (SCC) during SCCVN processing. If this limit is hit, SCCVN processing for the whole function will not be done and optimizations depending on it will be disabled. The default maximum SCC size is 10000.

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. The default value of the parameter is 100.

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. The default value of the parameter is 2000.

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. The default value of the parameter is 2, which is the minimal number of registers needed by typical instructions. This value is the best found from numerous experiments.

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. The default value of the parameter is 1000 for −O1 and 10000 for −O2 and above.

loop-max-datarefs-for-datadeps
Building data dapendencies 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 will not be handled then by the optimizations using loop data dependencies. The default value is 1000.

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. The default is 12.

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-ptr-growth-factor
IPA-SRA will replace 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.

tm-max-aggregate-size
When making copies of thread-local variables in a transaction, this parameter specifies the size in bytes after which variables will be 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. The default value is 10 parameters. A variable whose value is unknown at compilation time and defined outside a SCoP is a parameter of the SCoP.

graphite-max-bbs-per-function
To avoid exponential effects in the detection of SCoPs, the size of the functions analyzed by Graphite is bounded. The default value is 100 basic blocks.

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. The default value is 51 iterations.

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.

lto-partitions
Specify desired number of partitions produced during WHOPR compilation. The number of partitions should exceed the number of CPUs used for compilation. The default value is 32.

lto-minpartition
Size of minimal partition for WHOPR (in estimated instructions). This prevents expenses of splitting very small programs into too many partitions.

cxx-max-namespaces-for-diagnostic-help
The maximum number of namespaces to consult for suggestions when C++ name lookup fails for an identifier. The default is 1000.

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. The default value is 75. 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 stores paires that can be sunk. Set to 0 if either vectorization (**−ftree−vectorize**) or if-conversion (**−ftree−loop−if−convert**) is disabled. The default is 2.

allow-load-data-races
: Allow optimizers to introduce new data races on loads. Set to 1 to allow, otherwise to 0. This option is enabled by default unless implicitly set by the **−fmemory−model=** option.

allow-store-data-races
: Allow optimizers to introduce new data races on stores. Set to 1 to allow, otherwise to 0. This option is enabled by default unless implicitly set by the **−fmemory−model=** option.

allow-packed-load-data-races
: Allow optimizers to introduce new data races on packed data loads. Set to 1 to allow, otherwise to 0. This option is enabled by default unless implicitly set by the **−fmemory−model=** option.

allow-packed-store-data-races
: Allow optimizers to introduce new data races on packed data stores. Set to 1 to allow, otherwise to 0. This option is enabled by default unless implicitly set by the **−fmemory−model=** option.

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. The default is 0.

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.

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.

−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 know how to 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.

−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 will be 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 will need to 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.

−undef
Do not predefine any system-specific or GCC-specific macros. The standard predefined macros remain defined.

−I *dir*
Add the directory *dir* to the list of directories to be searched for header files. Directories named by **−I** are searched before the standard system include directories. If the directory *dir* is a standard system include directory, the option is ignored to ensure that the default search order for system directories and the special treatment of system headers are not defeated . If *dir* begins with =, then the = will be replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−o *file*
Write output to *file*. This is the same as specifying *file* as the second non-option argument to **cpp**. **gcc** has a different interpretation of a second non-option argument, so you must use **−o** to specify the output file.

−Wall
Turns on all optional warnings which are desirable for normal code. At present this is **−Wcomment**, **−Wtrigraphs**, **−Wmultichar** and a warning about integer promotion causing a change of sign in `#if` expressions. Note that many of the preprocessor's warnings are on by default and have no options to control them.

−Wcomment
−Wcomments
Warn whenever a comment-start sequence **/*** appears in a **/*** comment, or whenever a backslash-newline appears in a **//** comment. (Both forms have the same effect.)

−Wtrigraphs
Most trigraphs in comments cannot affect the meaning of the program. However, a trigraph that would form an escaped newline (**??/** at the end of a line) can, by changing where the comment begins or ends. Therefore, only trigraphs that would form escaped newlines produce warnings inside a comment.

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**.

−Wtraditional
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 problematic constructs which should be avoided.

−Wundef
Warn whenever an identifier which is not a macro is encountered in an **#if** directive, outside of **defined**. Such identifiers are replaced with zero.

−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 will also warn 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 CPP will report 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
```

−Wendif−labels
Warn whenever an **#else** or an **#endif** are followed by text. This usually happens in code of the form

```
#if FOO
...
#else FOO
...
#endif FOO
```

The second and third `FOO` should be in comments, but often are not in older programs. This warning is on by default.

−Werror
Make all warnings into hard errors. Source code which triggers warnings will be rejected.

−Wsystem−headers
Issue warnings for code in system headers. These are normally unhelpful in finding bugs in your own code, therefore suppressed. If you are responsible for the system library, you may want to see them.

−w Suppress all warnings, including those which GNU CPP issues by default.

−pedantic
Issue all the mandatory diagnostics listed in the C standard. Some of them are left out by default, since they trigger frequently on harmless code.

−pedantic−errors
Issue all the mandatory diagnostics, and make all mandatory diagnostics into errors. This includes mandatory diagnostics that GCC issues without **−pedantic** but treats as warnings.

−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 will still be 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 will appear in **−MM** dependency output. This is a slight change in semantics from GCC versions 3.0 and earlier.

−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 have sent preprocessed output.

When used with the driver options **−MD** or **−MMD**, **−MF** overrides the default dependency output file.

−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.

−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 will set 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.

−fpch−deps

When using precompiled headers, this flag will cause the dependency-output flags to also list the files from the precompiled header's dependencies. If not specified only the precompiled header would be 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.

−x c
−x c++
−x objective-c
−x assembler-with-cpp

Specify the source language: C, C++, Objective-C, or assembly. This has nothing to do with standards conformance or extensions; it merely selects which base syntax to expect. If you give none of these options, cpp will deduce the language from the extension of the source file: **.c**, **.cc**, **.m**, or **.S**. Some other common extensions for C++ and assembly are also recognized. If cpp does not recognize the extension, it will treat the file as C; this is the most generic mode.

Note: Previous versions of cpp accepted a **−lang** option which selected both the language and the standards conformance level. This option has been removed, because it conflicts with the **−l** option.

−std=*standard*
−ansi

Specify the standard to which the code should conform. Currently CPP knows about C and C++ standards; others may be added in the future.

standard may be one of:

`c90`
`c89`
`iso9899:1990`

The ISO C standard from 1990. **c90** is the customary shorthand for this version of the standard.

The **−ansi** option is equivalent to **−std=c90**.

`iso9899:199409`

The 1990 C standard, as amended in 1994.

`iso9899:1999`
`c99`
`iso9899:199x`
`c9x`

The revised ISO C standard, published in December 1999. Before publication, this was known as C9X.

`iso9899:2011`
`c11`
`c1x`

The revised ISO C standard, published in December 2011. Before publication, this was known as C1X.

`gnu90`
`gnu89`

The 1990 C standard plus GNU extensions. This is the default.

`gnu99`
`gnu9x`
The 1999 C standard plus GNU extensions.

`gnu11`
`gnu1x`
The 2011 C standard plus GNU extensions.

`c++98`
The 1998 ISO C++ standard plus amendments.

`gnu++98`
The same as **−std=c++98** plus GNU extensions. This is the default for C++ code.

−I−
Split the include path. 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`*`"`. This option has been deprecated.

−nostdinc
Do not search the standard system directories for header files. Only the directories you have specified with **−I** 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.)

−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**.

−idirafter *dir*
Search *dir* for header files, but do it *after* all directories specified with **−I** and the standard system directories have been exhausted. *dir* is treated as a system include directory. If *dir* begins with `=`, then the `=` will be replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−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.

−isystem *dir*
Search *dir* for header files, after all directories specified by **−I** but before the standard system directories. Mark it as a system directory, so that it gets the same special treatment as is applied to the standard system directories. If *dir* begins with =, then the = will be replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−iquote *dir*
Search *dir* only for header files requested with `#include "file"`; they are not searched for `#include <file>`, before all directories specified by **−I** and before the standard system directories. If *dir* begins with =, then the = will be replaced by the sysroot prefix; see **−−sysroot** and **−isysroot**.

−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 in identifiers. This option is experimental; in a future version of GCC, it will be enabled by default for C99 and C++.

−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**.

−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.

−fdebug−cpp
This option is only useful for debugging GCC. When used with **−E**, dumps debugging information about location maps. Every token in the output is preceded by the dump of the map its location belongs to. The dump of the map holding the location of a token would be:

```
{"P":F</file/path>;"F":F</includer/path>;"L":<line_num>;"C":<col_num>;"S"
```

When used without **−E**, this option has no effect.

−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 just as if no **−ftrack−macro−expansion** was present on the command line. 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**.

−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 UTF−32 or UTF−16, whichever corresponds to the width of `wchar_t`. 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.

−fworking−directory
Enable generation of linemarkers in the preprocessor output that will let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor will emit, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC will use 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.

−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**.

−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*.

−dCHARS
CHARS is a sequence of one or more of the following characters, and must not be preceded by a space. Other characters are interpreted by the compiler proper, or reserved for future versions of GCC, and so are silently ignored. If you specify characters whose behavior conflicts, the result is undefined.

M 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
```

will show all the predefined macros.

If you use **−dM** without the **−E** option, **−dM** is interpreted as a synonym for **−fdump−rtl−mach**.

D Like **M** except in two respects: it does *not* include the predefined macros, and it outputs *both* the **#define** directives and the result of preprocessing. Both kinds of output go to the standard output file.

N Like **D**, but emit only the macro names, not their expansions.

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

U Like **D** 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.

−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.

−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.

−traditional−cpp
Try to imitate the behavior of old-fashioned C preprocessors, as opposed to ISO C preprocessors.

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

The nine trigraphs and their replacements are

```
Trigraph:       ??(  ??)  ??<  ??>  ??=  ??/  ??'  ??!  ??−
Replacement:      [    ]    {    }    #    \    ^    |    ~
```

−remap
Enable special code to work around file systems which only permit very short file names, such as MS-DOS.

−−help
−−target−help
Print text describing all the command line options instead of preprocessing anything.

−v Verbose mode. Print out GNU CPP's version number at the beginning of execution, and report the final form of the include path.

−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 **...!** .

−version
−−version
Print out GNU CPP's version number. With one dash, proceed to preprocess as normal. With two dashes, exit immediately.

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 know how to 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.

−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.)

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.

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

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

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

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

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

−nodefaultlibs
Do not use the standard system libraries when linking. Only the libraries you specify will be passed to the linker, options specifying linkage of the system libraries, such as `−static−libgcc` or `−shared−libgcc`, will be 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.

−nostdlib
Do not use the standard system startup files or libraries when linking. No startup files and only the libraries you specify will be passed to the linker, options specifying linkage of the system libraries, such as `−static−libgcc` or `−shared−libgcc`, will be 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. (For example, **__main**, used to ensure C++ constructors will be called.)

−pie
Produce a position independent executable on targets that support it. For predictable results, you must also specify the same set of options that were used to generate code (**−fpie**, **−fPIE**, or model suboptions) when you specify this option.

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

−shared
Produce a shared object which can then be linked with other objects to form an executable. Not all systems support this option. For predictable results, you must also specify the same set of options that were used to generate code (**−fpic**, **−fPIC**, or model suboptions) when you specify this option.[1]

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

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

Therefore, the G++ and GCJ drivers automatically add **−shared−libgcc** whenever you build a shared library or a main executable, because C++ and Java 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 will not always be 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 will link the shared version of *libgcc* into shared libraries by default. Otherwise, it will take advantage of the linker and optimize 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++ or GCJ driver, as appropriate for the languages used in the program, or using the option **−shared−libgcc**, such that it is linked with the shared *libgcc*.

−static−libstdc++

When the **g++** program is used to link a C++ program, it will normally automatically link against **libstdc++**. If *libstdc++* is available as a shared library, and the **−static** option is not used, then this will link 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.

Options for Directory Search

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

−I*dir*

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

If a standard system include directory, or a directory specified with **−isystem**, is also specified with **−I**, the **−I** option will be ignored. The directory will still be 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.

−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.

−iquote*dir*

Add the directory *dir* to the head of the list of directories to be searched for header files only for the case of **#include "***file***"**; they are not searched for **#include <***file***>**, otherwise just like **−I**.

−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 each subprogram to be run, the compiler driver first tries the **−B** prefix, if any. If that name is not found, or if **−B** was not specified, the driver tries two standard prefixes, */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 will check to see if the path provided by the **−B** refers to a directory, and if necessary it will add 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 includes files in the preprocessor, because the compiler translates these options into **−isystem** options for the preprocessor. In this case, the compiler appends **include** to the prefix.

The 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 will be replaced by *[dir/]include*. This is to help with boot-strapping the compiler.

−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.

−−sysroot=*dir*
Use *dir* as the logical root directory for headers and libraries. For example, if the compiler would normally search for headers in */usr/include* and libraries in */usr/lib*, it will instead search *dir/usr/include* and *dir/usr/lib*.

If you use both this option and the **−isysroot** option, then the **−−sysroot** option will apply to libraries, but the **−isysroot** option will apply 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** will still work, but the library aspect will not.

−I−
This option has been deprecated. Please use **−iquote** instead for **−I** directories before the **−I−** and remove the **−I−**. Any directories you specify with **−I** options before the **−I−** option are searched only for the case of **#include "***file***"**; they are not searched for **#include <***file***>**.

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

In addition, the **−I−** option inhibits the use of the current directory (where the current input file came from) as the first search directory for **#include "***file***"**. There is no way to override this effect of **−I−**. With **−I.** you can specify searching the directory that was current when the compiler was invoked. That is not exactly the same as what the preprocessor does by default, but it is often satisfactory.

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

Specifying Target Machine and Compiler Version
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 version other than the one that was installed last.

Hardware Models and Configurations
Each target machine types can have its own special options, starting with **−m**, to choose among various hardware models or configurations−−−for example, 68010 vs 68020, floating coprocessor or none. A single installed version of the compiler can compile for any model or configuration, according to the options specified.

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

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
Preferrentially allocate registers that allow short instruction generation. This can result in increasesd instruction count, so if this reduces or increases code size might vary from case to case.

−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 will generally 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**

−mnosplit−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 will be unaffected, if they don't use SIMD vector modes in places where they affect size and/or alignment of relevant types.

−msplit−vecmove−early
Split vector moves into single word moves before reload. In theory this could 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 to use that register as a fixed register, and none, which means that no register is used for this purpose. The default is **−m1reg−none**.

ARM Options

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

−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 will cause the stack frames not to be generated for leaf functions. The default is **−mno−apcs−frame**.

−mapcs
This is a synonym for **−mapcs−frame**.

−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 will start with a recognizable set of instructions (or in fact one of a choice from a small set of different function prologues), and this information can be used to locate the start if functions inside an executable piece of code. The default is **−msched−prolog**.

−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.

−mlittle−endian
Generate code for a processor running in little-endian mode. This is the default for all standard configurations.

−mbig−endian
Generate code for a processor running in big-endian mode; the default is to compile code for a little-endian processor.

−mwords−little−endian
This option only applies when generating code for big-endian processors. Generate code for a little-endian word order but a big-endian byte order. That is, a byte order of the form **32107654**. Note: this option should only be used if you require compatibility with code for big-endian ARM processors generated by versions of the compiler prior to 2.8. This option is now deprecated.

−mcpu=*name*
This specifies the name of the target ARM processor. GCC uses this name to determine what kind of instructions it can emit when generating assembly code. Permissible names are: **arm2**, **arm250**, **arm3**, **arm6**, **arm60**, **arm600**, **arm610**, **arm620**, **arm7**, **arm7m**, **arm7d**, **arm7dm**, **arm7di**, **arm7dmi**, **arm70**, **arm700**, **arm700i**, **arm710**, **arm710c**, **arm7100**, **arm720**, **arm7500**, **arm7500fe**, **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**, **cortex−a5**, **cortex−a7**, **cortex−a8**, **cortex−a9**, **cortex−a15**, **cortex−r4**, **cortex−r4f**, **cortex−r5**, **cortex−m4**, **cortex−m3**, **cortex−m1**, **cortex−m0**, **xscale**, **iwmmxt**, **iwmmxt2**, **ep9312**, **fa526**, **fa626**, **fa606te**, **fa626te**, **fmp626**, **fa726te**.

−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 Linux, and not all architectures are recognized. If the auto-detect is unsuccessful the option has no effect.

−mtune=*name*
This option is very similar to the **−mcpu=** option, except that instead of specifying the actual target processor type, and hence restricting which instructions can be used, it specifies that GCC should tune the performance of the code as if the target were of the type specified in this option, but still choosing the instructions that it will generate based on the CPU specified by a **−mcpu=** option. For some ARM implementations better performance can be obtained by using this option.

−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=native causes the compiler to auto-detect the CPU of the build computer. At present, this feature is only supported on Linux, and not all architectures are recognized. If the auto-detect is unsuccessful the option has no effect.

−march=*name*
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: **armv2**, **armv2a**, **armv3**, **armv3m**, **armv4**,

armv4t, **armv5**, **armv5t**, **armv5e**, **armv5te**, **armv6**, **armv6j**, **armv6t2**, **armv6z**, **armv6zk**, **armv6−m**, **armv7**, **armv7−a**, **armv7−r**, **armv7−m**, **iwmmxt**, **iwmmxt2**, **ep9312**.

−march=native causes the compiler to auto-detect the architecture of the build computer. At present, this feature is only supported on Linux, and not all architectures are recognized. If the auto-detect is unsuccessful the option has no effect.

−mfpu=*name*
−mfpe=*number*
−mfp=*number*

This specifies what floating-point hardware (or hardware emulation) is available on the target. Permissible names are: **fpa**, **fpe2**, **fpe3**, **maverick**, **vfp**, **vfpv3**, **vfpv3−fp16**, **vfpv3−d16**, **vfpv3−d16−fp16**, **vfpv3xd**, **vfpv3xd−fp16**, **neon**, **neon−fp16**, **vfpv4**, **vfpv4−d16**, **fpv4−sp−d16** and **neon−vfpv4**. **−mfp** and **−mfpe** are synonyms for **−mfpu**=**fpe***number*, for compatibility with older versions of GCC.

If **−msoft−float** is specified this specifies the format of floating-point values.

If the selected floating-point hardware includes the NEON extension (e.g. **−mfpu**=**neon**), note that floating-point operations will not be used 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.

−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 size of all structures and unions will be rounded up to a multiple of the number of bits set by this option. Permissible values are 8, 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 the 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.

−mabort−on−noreturn

Generate a call to the function `abort` at the end of a `noreturn` function. It will be executed if the function tries to return.

−mlong−calls
−mno−long−calls

Tells the compiler to perform function calls by first loading the address of the function into a register and then performing a subroutine call on this register. This switch is needed if the target function will lie outside of the 64 megabyte addressing range of the offset based version of subroutine call instruction.

Even if this switch is enabled, not all function calls will be turned into long calls. The heuristic is that static functions, functions 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, will not be turned into long calls. The exception to this rule is that weak function definitions, functions with the **long-call** attribute or the **section** attribute, and functions that are within the scope of a **#pragma long_calls** directive, will always be turned into long calls.

This feature is not enabled by default. Specifying **−mno−long−calls** will restore the default behavior, as will placing the function calls within the scope of a **#pragma long_calls_off** directive. Note these switches have no effect on how the compiler generates code to handle function calls via function pointers.

−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. The default is R10 unless stack-checking is enabled, when R9 is used.

−mcirrus−fix−invalid−insns
Insert NOPs into the instruction stream to in order to work around problems with invalid Maverick instruction combinations. This option is only valid if the **−mcpu=ep9312** option has been used to enable generation of instructions for the Cirrus Maverick floating-point co-processor. This option is not enabled by default, since the problem is only present in older Maverick implementations. The default can be re-enabled by use of the **−mno−cirrus−fix−invalid−insns** switch.

−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.

−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 valid models are **soft**, which generates calls to `__aeabi_read_tp`, **cp15**, which fetches the thread pointer from `cp15` directly (supported in the arm6k architecture), and **auto**, which 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 dialects 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.

−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.

−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 and all ARMv6−M architectures, and enabled for all other architectures. If unaligned access is not enabled then words in packed data structures will be accessed a byte at a time.

The ARM attribute `Tag_CPU_unaligned_access` will be 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` will also be defined.

AVR Options

−mmcu=*mcu*
Specify Atmel AVR instruction set architectures (ISA) or MCU type.

For a complete list of *mcu* values that are supported by **avr-gcc**, see the compiler output when called with the **−−help=target** command line option. The default for this option is@tie{}`avr2`.

GCC supports the following AVR devices and ISAs:

`avr2`
"Classic" devices with up to 8@tie{}KiB of program memory. *mcu*@`tie{}= at90c8534`, `at90s2313`, `at90s2323`, `at90s2333`, `at90s2343`, `at90s4414`, `at90s4433`, `at90s4434`, `at90s8515`, `at90s8535`, `attiny22`, `attiny26`.

`avr25`
"Classic" devices with up to 8@tie{}KiB of program memory and with the `MOVW` instruction. *mcu*@`tie{}= at86rf401`, `ata6289`, `attiny13`, `attiny13a`, `attiny2313`, `attiny2313a`, `attiny24`, `attiny24a`, `attiny25`, `attiny261`, `attiny261a`, `attiny4313`, `attiny43u`, `attiny44`, `attiny44a`, `attiny45`, `attiny461`,

attiny461a, attiny48, attiny84, attiny84a, attiny85, attiny861, attiny861a, attiny87, attiny88.

avr3
: "Classic" devices with 16@tie{}KiB up to 64@tie{}KiB of program memory. *mcu*@tie{}= at43usb355, at76c711.

avr31
: "Classic" devices with 128@tie{}KiB of program memory. *mcu*@tie{}= at43usb320, atmega103.

avr35
: "Classic" devices with 16@tie{}KiB up to 64@tie{}KiB of program memory and with the MOVW instruction. *mcu*@tie{}= at90usb162, at90usb82, atmega16u2, atmega32u2, atmega8u2, attiny167.

avr4
: "Enhanced" devices with up to 8@tie{}KiB of program memory. *mcu*@tie{}= at90pwm1, at90pwm2, at90pwm2b, at90pwm3, at90pwm3b, at90pwm81, atmega48, atmega48a, atmega48p, atmega8, atmega8515, atmega8535, atmega88, atmega88a, atmega88p, atmega88pa, atmega8hva.

avr5
: "Enhanced" devices with 16@tie{}KiB up to 64@tie{}KiB of program memory. *mcu*@tie{}= at90can32, at90can64, at90pwm216, at90pwm316, at90scr100, at90usb646, at90usb647, at94k, atmega16, atmega161, atmega162, atmega163, atmega164a, atmega164p, atmega165, atmega165a, atmega165p, atmega168, atmega168a, atmega168p, atmega169, atmega169a, atmega169p, atmega169pa, atmega16a, atmega16hva, atmega16hva2, atmega16hvb, atmega16m1, atmega16u4, atmega32, atmega323, atmega324a, atmega324p, atmega324pa, atmega325, atmega3250, atmega3250a, atmega3250p, atmega325a, atmega325p, atmega328, atmega328p, atmega329, atmega3290, atmega3290a, atmega3290p, atmega329a, atmega329p, atmega329pa, atmega32c1, atmega32hvb, atmega32m1, atmega32u4, atmega32u6, atmega406, atmega64, atmega640, atmega644, atmega644a, atmega644p, atmega644pa, atmega645, atmega6450, atmega6450a, atmega6450p, atmega645a, atmega645p, atmega649, atmega6490, atmega649a, atmega649p, atmega64c1, atmega64hve, atmega64m1, m3000.

avr51
: "Enhanced" devices with 128@tie{}KiB of program memory. *mcu*@tie{}= at90can128, at90usb1286, at90usb1287, atmega128, atmega1280, atmega1281, atmega1284p, atmega128rfa1.

avr6
: "Enhanced" devices with 3−byte PC, i.e. with more than 128@tie{}KiB of program memory. *mcu*@tie{}= atmega2560, atmega2561.

avrxmega2
: "XMEGA" devices with more than 8@tie{}KiB and up to 64@tie{}KiB of program memory. *mcu*@tie{}= atxmega16a4, atxmega16d4, atxmega16x1, atxmega32a4, atxmega32d4, atxmega32x1.

avrxmega4
: "XMEGA" devices with more than 64@tie{}KiB and up to 128@tie{}KiB of program memory. *mcu*@tie{}= atxmega64a3, atxmega64d3.

avrxmega5
: "XMEGA" devices with more than 64@tie{}KiB and up to 128@tie{}KiB of program memory and more than 64@tie{}KiB of RAM. *mcu*@tie{}= atxmega64a1, atxmega64a1u.

`avrxmega6`
: "XMEGA" devices with more than 128@tie{}KiB of program memory. *mcu*`@tie{}`= `atxmega128a3`, `atxmega128d3`, `atxmega192a3`, `atxmega192d3`, `atxmega256a3`, `atxmega256a3b`, `atxmega256a3bu`, `atxmega256d3`.

`avrxmega7`
: "XMEGA" devices with more than 128@tie{}KiB of program memory and more than 64@tie{}KiB of RAM. *mcu*`@tie{}`= `atxmega128a1`, `atxmega128a1u`.

`avr1`
: This ISA is implemented by the minimal AVR core and supported for assembler only. *mcu*`@tie{}`= `at90s1200`, `attiny11`, `attiny12`, `attiny15`, `attiny28`.

−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.

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 to 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.

−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.

−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 `--relax` option to the linker command line when the linker is called.

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.

−mshort−calls

Use `RCALL`/`RJMP` instructions even on devices with 16@tie{}KiB or more of program memory, i.e. on devices that have the `CALL` and `JMP` instructions. See also the `−mrelax` command line option.

−msp8

Treat the stack pointer register as an 8−bit register, i.e. assume the high byte of the stack pointer is zero. In general, you don't need to set this option by hand.

This option is used internally by the compiler to select and build multilibs for architectures `avr2` and `avr25`. These architectures mix devices with and without `SPH`. For any setting other than `−mmcu=avr2` or `−mmcu=avr25` the compiler driver will add or remove this option from the compiler proper's command line, because the compiler then knows if the device or architecture has an 8−bit stack pointer and thus no `SPH` register or not.

−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
```

−mtiny−stack
Only change the lower 8@tie{}bits of the stack pointer.

EIND and Devices with more than 128 Ki Bytes of Flash

Pointers in the implementation are 16@tie{}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@tie{}Ki words.

In order to facilitate indirect jump on devices with more than 128@tie{}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 implicitely 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 will generate the stubs correctly an all situaltion. See the compiler option −mrelax and the linler 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 (http://nongnu.org/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@tie{}KiB range that can be accessed with 16−bit pointers. To access memory locations outside this 64@tie{}KiB range, the contentent 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@tie{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@tie{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_`*`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.

`__AVR_HAVE_ELPM__`
: The device has the the `ELPM` instruction.

`__AVR_HAVE_ELPMX__`
: The device has the `ELPM R`*`n`*`,Z` and `ELPM R`*`n`*`,Z+` instructions.

`__AVR_HAVE_MOVW__`
: The device has the `MOVW` instruction to perform 16−bit register-register moves.

`__AVR_HAVE_LPMX__`
: The device has the `LPM R`*`n`*`,Z` and `LPM R`*`n`*`,Z+` instructions.

`__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 at least 16@tie{}KiB of program memory and if `−mshort−calls` is not set.

`__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@tie{}KiB of program memory. This also means that the program counter (PC) is 3@tie{}bytes wide.

`__AVR_2_BYTE_PC__`
: The program counter (PC) is 2@tie{}bytes wide. This is the case for devices with up to 128@tie{}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_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@tie{}address.

`__WITH_AVRLIBC__`
The compiler is configured to be used together with AVR-Libc. See the `--with-avrlibc` configure option.

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 will be enabled. If *sirevision* is **none**, no workarounds are enabled. If *sirevision* is **any**, all workarounds for the targeted processor will be 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.

Support for **bf561** is incomplete. For **bf561**, Only the processor macro is defined. Without this option, **bf532** is used as the processor by default. The corresponding predefined processor macros for *cpu* is to be defined. And for **bfin-elf** toolchain, this causes the hardware BSP provided by libgloss to be linked in if **−msim** is not given.

−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. The option **−fomit−frame−pointer** removes the frame pointer for all functions, which might make debugging harder.

−mspecld−anomaly
When enabled, the compiler will ensure 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 will ensure 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.

−mlow−64k
When enabled, the compiler is free to take advantage of the knowledge that the entire program fits into the low 64k of memory.

−mno−low−64k
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 will be generated for jump and call insns.

−mshared−library−id=n
Specified the identification number of the ID based shared library being compiled. Specifying a value of 0 will generate more compact code, specifying other values will force 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** will restore 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 standalone application for multicore Blackfin processor. Proper start files and link scripts will be used to support multicore. This option defines `__BFIN_MULTICORE`. It can only be used with **−mcpu=bf561**[−*sirevision*]. It can be used with **−mcorea** or **−mcoreb**. If it's used without **−mcorea** or **−mcoreb**, single application/dual core programming model is used. In this model, the main function of Core B should be named as coreb_main. If it's used with **−mcorea** or **−mcoreb**, one application per core programming model is used. If this option is not used, single core application programming model is used.

−mcorea
Build standalone application for Core A of BF561 when using one application per core programming model. Proper start files and link scripts will be used to support Core A. This option defines `__BFIN_COREA`. It must be used with **−mmulticore**.

−mcoreb
Build standalone application for Core B of BF561 when using one application per core programming model. Proper start files and link scripts will be used to support Core B. This option defines `__BFIN_COREB`. When this option is used, coreb_main should be used instead of main. It must be used with **−mmulticore**.

−msdram
Build standalone application for SDRAM. Proper start files and link scripts will be used to put the application into SDRAM. Loader should initialize SDRAM before loading the application into SDRAM. This option defines `__BFIN_SDRAM`.

−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=*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 objets, 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** except for cris-axis-linux-gnu, where the default is **v10**.

−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 active by default.

−mpdebug
Enable CRIS-specific verbose debug-related information in the assembly code. This option also has the effect to turn 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) arranges (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 variable needs to be allocated.

−mno−gotplt
−mgotplt
With **−fpic** and **−fPIC**, don't generate (do generate) instruction sequences that load addresses for functions from the PLT part of the GOT rather than (traditional on other architectures) calls to the PLT. The default is **−mgotplt**.

−melf
Legacy no-op option only recognized with the cris-axis-elf and cris-axis-linux-gnu targets.

−mlinux
Legacy no-op option only recognized with the cris-axis-linux-gnu target.

−sim
This option, recognized for the cris-axis-elf 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.

CR16 Options

These options are defined specifically for the CR16 ports.

−mmac
Enable the use of multiply-accumulate instructions. Disabled by default.

−mcr16cplus
−mcr16c
Generate code for CR16C or CR16C+ architecture. CR16C+ architecture is default.

−msim
Links the library libsim.a which is in compatible with simulator. Applicable to elf compiler only.

−mint32
Choose integer type as 32−bit wide.

−mbit−ops
Generates sbit/cbit instructions for bit manipulations.

−mdata−model=*model*
Choose a data model. The choices for *model* are **near**, **far** or **medium**. **medium** is default. However, **far** is not valid when −mcr16c option is chosen as CR16C architecture does not support far data model.

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 will create an object file for the single architecture that it 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 targetting, 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*, will only permit 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*, will fail and print 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*, will quietly give 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 will be 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.

−mmacosx−version−min=*version*

The earliest version of MacOS X that this executable will run on is *version*. Typical values of *version* include `10.1`, `10.2`, and `10.3.9`.

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−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 turn around development. Needed to enable 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 be loading the build output file being linked. See man *ld* (1) for more information.

−dynamiclib

When passed this option, GCC will produce 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.

−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* will be used to perform floating-point operations. Unless they are replaced by routines that emulate the floating-point operations, or compiled in such a way as to call such emulations routines, these routines will issue floating-point operations. If you are compiling for an Alpha without floating-point operations, you must ensure that the library is built so as not to call them.

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

−mfp−reg
−mno−fp−regs

Generate code that uses (does not use) the floating-point register set. **−mno−fp−regs** implies **−msoft−float**. If the floating-point register set is not used, floating-point operands are passed in integer registers as if they were integers and floating-point results are passed in `$0` instead of `$f0`.

This is a non-standard calling sequence, so any function with a floating-point argument or return value called by code compiled with **−mno−fp−regs** must also be compiled with that option.

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

−mieee

The Alpha architecture implements floating-point hardware optimized for maximum performance. It is mostly compliant with the IEEE floating-point standard. However, for full compliance, software assistance is required. This option generates code fully IEEE-compliant code *except* that the *inexact-flag* is not maintained (see below). If this option is turned on, the 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**.

DEBIAN SPECIFIC: (only for alpha architecture) This option is on by default, unless **−ffinite−math−only** (which is part of the **−ffast−math** set) is specified, because the software functions in the GNU libc math libraries generate denormalized numbers, NaNs, and infs (all of which will cause a programs to SIGFPE when it attempts to use the results without **−mieee**).

−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. Under DEC Unix, this has the effect that IEEE-conformant math library routines will be linked in.

−mbuild−constants
Normally GCC examines a 32− or 64−bit integer constant to see if it can construct it from smaller constants in two or three instructions. If it cannot, it will output the constant as a literal and generate code to load it from the data segment at 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 would 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.

−malpha−as
−mgas
Select whether to generate code to be assembled by the vendor-supplied assembler (**−malpha−as**) or by the GNU assembler **−mgas**.

−mbwx
−mno−bwx
−mcix
−mno−cix
−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 was specified.

−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 will choose the default values for the instruction set from the processor you specify. If you do not specify a processor type, GCC will default to the processor on which the compiler was built.

Supported values for *cpu_type* are

ev4
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.

DEC Alpha/VMS Options

These **−m** options are defined for the DEC Alpha/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.

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 will 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.

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*** and ***−*−linux−*android*** targets.

−muclibc
Use uClibc C library. This is the default on ***−*−linux−*uclibc*** 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**.

−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 will run 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.

−mbig−switch
Generate code suitable for big switch tables. Use this option only if the assembler/linker complain about out of range branches within a switch table.

−mjump−in−delay
Fill delay slots of function calls with unconditional jump instructions by modifying the return pointer for the function call to be the target of the conditional jump.

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

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

−mno−space−regs
Generate code that assumes the target has no space registers. This allows GCC to generate faster indirect calls and use unscaled index address modes.

Such code is suitable for level 0 PA systems and kernels.

−mfast−indirect−calls
Generate code that assumes calls never cross space boundaries. This allows GCC to emit code that performs faster indirect calls.

This option will 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 can not use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

−mlong−load−store
Generate 3−instruction load and store sequences as sometimes required by the HP-UX 10 linker. This is equivalent to the **+k** option to the HP compilers.

−mportable−runtime
Use the portable calling conventions proposed by HP for ELF systems.

−mgas
Enable the use of assembler directives only GAS understands.

−mschedule=*cpu-type*
Schedule code according to the constraints for the machine type *cpu-type*. The choices for *cpu-type* are **700 7100**, **7100LC**, **7200**, **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**.

−mlinker−opt
Enable the optimization pass in the HP-UX linker. Note this makes symbolic debugging impossible. It also triggers a bug in the HP-UX 8 and HP-UX 9 linkers in which they give bogus error messages when linking some programs.

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

−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.

−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.

−mgnu−ld
Use GNU ld specific options. 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 have any affect on 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 HP ld specific options. 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 have any affect on 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***.

−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 will degrade 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.

−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.

Intel 386 and AMD x86−64 Options

These **−m** options are defined for the i386 and x86−64 family of computers:

−mtune=*cpu-type*

Tune to *cpu-type* everything applicable about the generated code, except for the ABI and the set of available instructions. The choices for *cpu-type* are:

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** 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, the code generated option will change to reflect the processors that were most common when that version of GCC was 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.

native

This selects the CPU to tune for at compilation time by determining the processor type of the compiling machine. Using **−mtune=native** will produce code optimized for the local machine under the constraints of the selected instruction set. Using **−march=native** will enable all instruction subsets supported by the local machine (hence the result might not run on different machines).

i386

Original Intel's i386 CPU.

i486

Intel's i486 CPU. (No scheduling is implemented for this chip.)

i586, pentium

Intel Pentium CPU with no MMX support.

pentium-mmx

Intel PentiumMMX CPU based on Pentium core with MMX instruction set support.

pentiumpro

Intel PentiumPro CPU.

i686

Same as `generic`, but when used as `march` option, PentiumPro instruction set will be used, so the code will run on all i686 family chips.

pentium2

Intel Pentium2 CPU based on PentiumPro core with MMX instruction set support.

pentium3, pentium3m

Intel Pentium3 CPU based on PentiumPro core with MMX and SSE instruction set support.

pentium-m

Low power version of Intel Pentium3 CPU with MMX, SSE and SSE2 instruction set support. Used by Centrino notebooks.

pentium4, pentium4m
Intel Pentium4 CPU with MMX, SSE and SSE2 instruction set support.

prescott
Improved version of Intel Pentium4 CPU with MMX, SSE, SSE2 and SSE3 instruction set support.

nocona
Improved version of Intel Pentium4 CPU with 64−bit extensions, MMX, SSE, SSE2 and SSE3 instruction set support.

core2
Intel Core2 CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3 and SSSE3 instruction set support.

corei7
Intel Core i7 CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1 and SSE4.2 instruction set support.

corei7−avx
Intel Core i7 CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AES and PCLMUL instruction set support.

core-avx-i
Intel Core CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AES, PCLMUL, FSGSBASE, RDRND and F16C instruction set support.

atom
Intel Atom CPU with 64−bit extensions, MMX, SSE, SSE2, SSE3 and SSSE3 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
AMD K8 core based CPUs with x86−64 instruction set support. (This supersets MMX, SSE, SSE2, 3DNow!, enhanced 3DNow! and 64−bit instruction set extensions.)

k8−sse3, opteron−sse3, athlon64−sse3
Improved versions of k8, opteron and athlon64 with SSE3 instruction set support.

amdfam10, barcelona
AMD Family 10h core based CPUs with x86−64 instruction set support. (This supersets MMX, SSE, SSE2, SSE3, SSE4A, 3DNow!, enhanced 3DNow!, ABM and 64−bit instruction set extensions.)

bdver1
AMD Family 15h core based CPUs with x86−64 instruction set support. (This supersets FMA4, AVX, XOP, LWP, AES, PCL_MUL, 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, AVX, XOP, LWP, AES, PCL_MUL, CX16, MMX, SSE, SSE2, SSE3, SSE4A, SSSE3, SSE4.1, SSE4.2, ABM and 64−bit instruction set extensions.)

btver1
AMD Family 14h core based CPUs with x86−64 instruction set support. (This supersets MMX, SSE, SSE2, SSE3, SSSE3, SSE4A, CX16, ABM 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 Winchip2 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 CPU with MMX and SSE instruction set support. (No scheduling is implemented for this chip.)

geode
Embedded AMD CPU with MMX and 3DNow! instruction set support.

While picking a specific *cpu-type* will schedule things appropriately for that particular chip, the compiler will not generate any code that does not run on the default machine type without the **−march=***cpu-type* option being used. For example, if GCC is configured for i686−pc−linux−gnu then **−mtune=pentium4** will generate code that is tuned for Pentium4 but will still run on i686 machines.

−march=*cpu-type*
Generate instructions for the machine type *cpu-type*. The choices for *cpu-type* are the same as for **−mtune**. Moreover, specifying **−march=***cpu-type* implies **−mtune=***cpu-type*.

−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 i386 compiler.

sse Use scalar floating-point instructions present in the SSE instruction set. This instruction set is supported by Pentium3 and newer chips, in the AMD line by Athlon−4, Athlon-xp and Athlon-mp chips. The earlier version of 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 Pentium4 and the future AMD x86−64 chips, supports double-precision arithmetic too.

For the i386 compiler, you need to 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.

sse,387
sse+387

both

Attempt to utilize both instruction sets at once. This effectively double 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 instable performance.

−masm=*dialect*

Output asm instructions using selected *dialect*. Supported choices are **intel** or **att** (the default one). Darwin does not support **intel**.

−mieee−fp
−mno−ieee−fp

Control whether or not the compiler uses IEEE floating-point comparisons. These handle correctly the case where the result of a comparison is unordered.

−msoft−float

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

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

−mno−fp−ret−in−387

Do not use the FPU registers for return values of functions.

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

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

−mno−fancy−math−387

Some 387 emulators do not support the `sin`, `cos` and `sqrt` instructions for the 387. Specify this option to avoid generating those instructions. This option is the default on FreeBSD, OpenBSD and NetBSD. This option is overridden when **−march** indicates that the target CPU will always have an FPU and so the instruction will not need emulation. As of revision 2.6.1, 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 will be aligned differently than the published application binary interface specifications for the 386 and will not be 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 i386 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 to be 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, the structures and arrays containing `long double` variables will change their size as well as function calling convention for function taking `long double` will be modified. Hence they will not be binary compatible with arrays or structures in code compiled without that switch.

−mlarge−data−threshold=*number*

When **−mcmodel=medium** is specified, the data greater than *threshold* are placed in large data section. This value must be the same across all object linked into the binary and defaults to 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 will be generated for calls to those functions.

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

−mregparm=*num*

Control how many registers are used to pass integer arguments. By default, no registers are used to pass arguments, and at most 3 registers can be used. You can control this behavior for a specific function by using the function attribute **regparm**.

Warning: if you use this switch, and *num* is nonzero, then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

−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 Solaris@tie{}8 and 9 and VxWorks to match the ABI of the Sun Studio compilers until version 12. Later compiler versions (starting with Studio 12 Update@tie{}1) follow the ABI used by other x86 targets, which is the default on Solaris@tie{}10 and later. *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.

−mstackrealign

Realign the stack at entry. On the Intel x86, the **−mstackrealign** option will generate an alternate prologue and epilogue that realigns the run-time stack if necessary. This supports mixing legacy codes that keep a 4−byte aligned stack with modern codes that keep a 16−byte stack 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).

−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** will be used.

On Pentium and PentiumPro, `double` and `long double` values should be aligned to an 8−byte boundary (see **−malign−double**) or suffer significant run time performance penalties. On Pentium III, the Streaming SIMD Extension (SSE) data type `__m128` 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 will most likely misalign the stack. It is recommended that libraries that use callbacks always use the default setting.

This extra alignment does consume extra stack space, 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
−mno−mmx
−msse
−mno−sse
−msse2
−mno−sse2
−msse3
−mno−sse3
−mssse3
−mno−ssse3
−msse4.1
−mno−sse4.1
−msse4.2
−mno−sse4.2
−msse4
−mno−sse4
−mavx
−mno−avx
−mavx2
−mno−avx2

−maes
−mno−aes
−mpclmul
−mno−pclmul
−mfsgsbase
−mno−fsgsbase
−mrdrnd
−mno−rdrnd
−mf16c
−mno−f16c
−mfma
−mno−fma
−msse4a
−mno−sse4a
−mfma4
−mno−fma4
−mxop
−mno−xop
−mlwp
−mno−lwp
−m3dnow
−mno−3dnow
−mpopcnt
−mno−popcnt
−mabm
−mno−abm
−mbmi
−mbmi2
−mno−bmi
−mno−bmi2
−mlzcnt
−mno−lzcnt
−mtbm
−mno−tbm

These switches enable or disable the use of instructions in the MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, AVX, AVX2, AES, PCLMUL, FSGSBASE, RDRND, F16C, FMA, SSE4A, FMA4, XOP, LWP, ABM, BMI, BMI2, LZCNT or 3DNow!
extended instruction sets. 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.

To have SSE/SSE2 instructions generated 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 will 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.

−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 AVX to SSE transition penalty as well as remove unnecessary zeroupper intrinsics.

−mcx16

This option will enable GCC to use CMPXCHG16B instruction in generated code. CMPXCHG16B allows for atomic operations on 128−bit double quadword (or oword) data types. This is useful for high resolution counters that could be updated by multiple processors (or cores). This instruction is generated as part of atomic built-in functions: see **__sync Builtins** or **__atomic Builtins** for details.

−msahf

This option will enable GCC to use SAHF instruction in generated 64−bit code. Early Intel CPUs with Intel 64 lacked LAHF and SAHF instructions supported by AMD64 until introduction of Pentium 4 G1 step in December 2005. LAHF and SAHF are load and store instructions, respectively, for certain status flags. In 64−bit mode, SAHF instruction is used to optimize `fmod`, `drem` or `remainder` built-in functions: see **Other Builtins** for details.

−mmovbe

This option will enable GCC to use movbe instruction to implement `__builtin_bswap32` and `__builtin_bswap64`.

−mcrc32

This option will enable built-in functions, `__builtin_ia32_crc32qi`, `__builtin_ia32_crc32hi`. `__builtin_ia32_crc32si` and `__builtin_ia32_crc32di` to generate the crc32 machine instruction.

−mrecip

This option will enable GCC to use 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 **−finite−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 allows to control 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** would enable 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 types are `svml` for the Intel short vector math library and `acml` for the AMD math core library style of interfacing. GCC will currently emit calls to `vmldExp2`, `vmldLn2`, `vmldLog102`, `vmldLog102`, `vmldPow2`, `vmldTanh2`, `vmldTan2`, `vmldAtan2`, `vmldAtanh2`, `vmldCbrt2`, `vmldSinh2`, `vmldSin2`, `vmldAsinh2`, `vmldAsin2`, `vmldCosh2`, `vmldCos2`, `vmldAcosh2`, `vmldAcos2`, `vmlsExp4`, `vmlsLn4`, `vmlsLog104`, `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 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 corresponding function type when **−mveclibabi=acml** is used. Both **−ftree−vectorize** and **−funsafe−math−optimizations** have to be enabled. A SVML or ACML ABI compatible library will have to be specified at link time.

−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 Windows. On all other systems, the default is the SYSV ABI. You can control this behavior for a specific function by using the function attribute **ms_abi**/**sysv_abi**.

−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 will be computed in the function prologue. This is faster on most modern CPUs because of reduced dependencies, improved scheduling and reduced stack usage when preferred stack boundary is not equal to 2. The drawback is a notable increase in code size. This switch implies **−mno−push−args**.

−mthreads
Support thread-safe exception handling on **Mingw32**. Code that relies on thread-safe exception handling must compile and link all code with the **−mthreads** option. When compiling, **−mthreads** defines **−D_MT**; when linking, it links in a special thread helper library **−lmingwthrd** which cleans up per thread exception handling data.

−mno−align−stringops
Do not align destination of inlined string operations. This switch reduces code size and improves performance in case the destination is already aligned, but GCC 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, increase code size, but may improve performance of code that depends on fast memcpy, strlen and memset for short lengths.

−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*
Overwrite internal decision heuristic about particular algorithm to inline string operation with. The allowed values are `rep_byte`, `rep_4byte`, `rep_8byte` for expanding using i386 `rep` prefix of specified size, `byte_loop`, `loop`, `unrolled_loop` for expanding inline loop, `libcall` for always expanding library call.

−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−frame−pointer** removes the frame pointer for all 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 legal depends on the operating system, and whether it maps the segment to cover the entire TLS area.

For systems that use GNU libc, 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 prologue. Note: On x86 architectures the attribute `ms_hook_prologue` isn't possible at the moment for **−mfentry** and **−pg**.

−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.

These **−m** switches are supported in addition to the above on AMD x86−64 processors in 64−bit environments.

−m32
−m64
−mx32
Generate code for a 32−bit or 64−bit environment. The **−m32** option sets int, long and pointer to 32 bits and generates code that runs on any i386 system. The **−m64** option sets int to 32 bits and long and pointer to 64 bits and generates code for AMD's x86−64 architecture. The **−mx32** option sets int, long and pointer to 32 bits and generates code for AMD's x86−64 architecture. For darwin only the **−m64** option turns off the **−fno−pic** and **−mdynamic−no−pic** options.

−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 will not be 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.

i386 and x86−64 Windows Options

These additional options are available for Windows targets:

−mconsole
This option is available for Cygwin and MinGW targets. It 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 is the default behavior for Cygwin and 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.

−mthread
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 specifies that the UNICODE macro is getting predefined and that the unicode capable runtime startup code is chosen.

−mwin32
This option is available for Cygwin and MinGW targets. It specifies that the typical 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 stack used by nested functions isn't set. This is necessary for binaries running in kernel mode of Windows, as there the user32 API, which is used to set executable privileges, isn't available.

−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 will be enabled by default if GCC detects that the target assembler found during configuration supports the feature.

See also under **i386 and x86−64 Options** for standard options.

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 DWARF2 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 can not use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

−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 will result in generation of the ld.a instructions and the corresponding check instructions (ld.c / chk.a). The default is 'disable'.

−msched−ar−data−spec
−mno−sched−ar−data−spec
(En/Dis)able data speculative scheduling after reload. This will result in generation of the ld.a instructions and the corresponding check instructions (ld.c / chk.a). The default is 'enable'.

−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 will result in generation of the ld.s instructions and the corresponding check instructions chk.s . The default is 'disable'.

−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 is 'enable'.

−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 is 'enable'.

−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 is 'enable'.

−mno−sched−prefer−non−data−spec−insns
−msched−prefer−non−data−spec−insns
If enabled, data speculative instructions will be chosen for schedule only if there are no other choices at the moment. This will make the use of the data speculation much more conservative. The default is 'disable'.

−mno−sched−prefer−non−control−spec−insns
−msched−prefer−non−control−spec−insns
If enabled, control speculative instructions will be chosen for schedule only if there are no other choices at the moment. This will make the use of the control speculation much more conservative. The default is 'disable'.

−mno−sched−count−spec−in−critical−path
−msched−count−spec−in−critical−path
If enabled, speculative dependencies will be considered during computation of the instructions priorities. This will make the use of the speculation a bit more conservative. The default is 'disable'.

−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
Disallow more than 'msched−max−memory−insns' in instruction group. Otherwise, limit is 'soft' meaning that we would prefer non-memory operations when limit is reached but may still schedule memory operations.

IA−64/VMS Options

These **−m** options are defined for the IA−64/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.

LM32 Options

These **−m** options are defined for the Lattice Mico32 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.

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 will use during code generation. These pseudo-registers will be 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 the default runtime libraries gcc builds.

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 will generate `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 will generate `seth/add3` instructions to load their addresses), and assume subroutines may not be reachable with the `bl` instruction (the compiler will generate the much slower `seth/add3/jl` instruction sequence).

−msdata=none
Disable use of the small data area. Variables will be put into one of **.data**, **bss**, or **.rodata** (unless the `section` attribute has been specified). This is the default.

The small data area consists of sections **.sdata** and **.sbss**. Objects may be explicitly put in the small data area with the `section` attribute using one of these sections.

−msdata=sdata
Put small global and static data in the small data area, but do not generate special code to reference them.

−msdata=use
Put small global and static data in the small data area, and generate special instructions to reference them.

−G *num*
Put global and static objects less than or equal to *num* bytes into the small data or bss sections instead of the normal data or bss sections. The default value of *num* is 8. The **−msdata** option must be set to one of **sdata** or **use** for this option to have any effect.

All modules should be compiled with the same **−G** *num* value. Compiling with different values of *num* may or may not work; if it doesn't the linker will give an error message−−−incorrect code will not be generated.

−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 will be preferred over conditional code, if it is 2, then the opposite will apply.

−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 will only be 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 51cn 51em 51qe**
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 **__m***uarch***__** 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 will be generated for calls to those functions.

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

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

−mno−rtd
Do not use the calling conventions selected by **−mrtd**. This is the default.

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

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

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

−mno−strict−align
−mstrict−align
Do not (do) assume that unaligned memory references will be handled by the system.

−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
Specified the identification number of the ID based shared library being compiled. Specifying a value of 0 will generate more compact code, specifying other values will force 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.

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.

MeP Options

−mabsdiff
Enables the `abs` instruction, which is the absolute difference between two registers.

−mall−opts
Enables all the optional instructions − average, multiply, divide, bit operations, leading zero, absolute difference, min/max, clip, and saturation.

−maverage
Enables the `ave` instruction, which computes the average of two registers.

−mbased=*n*
Variables of size *n* bytes or smaller will be placed in the `.based` section by default. Based variables use the `$tp` register as a base register, and there is a 128−byte limit to the `.based` section.

−mbitops
Enables the bit operation instructions − bit test (`btstm`), set (`bsetm`), clear (`bclrm`), invert (`bnotm`), and test-and-set (`tas`).

−mc=*name*
Selects which section constant data will be placed in. *name* may be `tiny`, `near`, or `far`.

−mclip
Enables the `clip` instruction. Note that `−mclip` is not useful unless you also provide `−mminmax`.

−mconfig=*name*
Selects one of the build-in core configurations. Each MeP chip has one or more modules in it; each module has a core CPU and a variety of coprocessors, optional instructions, and peripherals. The `MeP−Integrator` tool, not part of GCC, provides these configurations through this option; using this option is the same as using all the corresponding command-line options. The default configuration is `default`.

−mcop
Enables the coprocessor instructions. By default, this is a 32−bit coprocessor. Note that the coprocessor is normally enabled via the `−mconfig=` option.

−mcop32
Enables the 32−bit coprocessor's instructions.

−mcop64
Enables the 64−bit coprocessor's instructions.

−mivc2
Enables IVC2 scheduling. IVC2 is a 64−bit VLIW coprocessor.

−mdc
Causes constant variables to be placed in the `.near` section.

−mdiv
Enables the `div` and `divu` instructions.

−meb
Generate big-endian code.

−mel
Generate little-endian code.

−mio−volatile
Tells the compiler that any variable marked with the `io` attribute is to be considered volatile.

−ml
Causes variables to be assigned to the `.far` section by default.

−mleadz
Enables the `leadz` (leading zero) instruction.

−mm
Causes variables to be assigned to the `.near` section by default.

−mminmax
Enables the `min` and `max` instructions.

−mmult
Enables the multiplication and multiply-accumulate instructions.

−mno−opts
Disables all the optional instructions enabled by `−mall−opts`.

−mrepeat
Enables the `repeat` and `erepeat` instructions, used for low-overhead looping.

−ms
Causes all variables to default to the `.tiny` section. Note that there is a 65536−byte limit to this section. Accesses to these variables use the `%gp` base register.

−msatur
Enables the saturation instructions. Note that the compiler does not currently generate these itself, but this option is included for compatibility with other tools, like `as`.

−msdram
Link the SDRAM-based runtime instead of the default ROM-based runtime.

−msim
Link the simulator runtime libraries.

−msimnovec
Link the simulator runtime libraries, excluding built-in support for reset and exception vectors and tables.

−mtf
Causes all functions to default to the `.far` section. Without this option, functions default to the `.near` section.

−mtiny=*n*
Variables that are *n* bytes or smaller will be allocated to the `.tiny` section. These variables use the `$gp` base register. The default for this option is 4, but note that there's a 65536−byte limit to the `.tiny` section.

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 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**, **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.

−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 be 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*.

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 will run 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**, **mips64** and **mips64r2**. 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**, **74kc**, **74kf2_1**, **74kf1_1**, **74kf3_2**, **1004kc**, **1004kf2_1**, **1004kf1_1**, **loongson2e**, **loongson2f**, **loongson3a**, **m4k**, **octeon**, **octeon+**, **octeon2**, **orion**, **r2000**, **r3000**, **r3900**, **r4000**, **r4400**, **r4600**, **r4650**, **r6000**, **r8000**, **rm7000**, **rm9000**, **r10000**, **r12000**, **r14000**, **r16000**, **sb1**, **sr71000**, **vr4100**, **vr4111**, **vr4120**, **vr4130**, **vr4300**, **vr5000**, **vr5400**, **vr5500** and **xlr**. 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).

Native Linux/GNU and IRIX toolchains also support 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** will set **_MIPS_ARCH** to **"r2000"** and define the macro **_MIPS_ARCH_R2000**.

Note that the **_MIPS_ARCH** macro uses the processor names given above. In other words, it will have the full prefix and will 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 will optimize for the processor specified by **−march**. By using **−march** and **−mtune** together, it is possible to generate code that will run 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**.

−mips32r2
Equivalent to **−march=mips32r2**.

−mips64
Equivalent to **−march=mips64**.

−mips64r2
Equivalent to **−march=mips64r2**.

−mips16
−mno−mips16
Generate (do not generate) MIPS16 code. If GCC is targetting a MIPS32 or MIPS64 architecture, it will make use of the MIPS16e ASE.

MIPS16 code generation can also be controlled on a per-function basis by means of `mips16` and `nomips16` attributes.

−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−mips16
−mno−interlink−mips16
Require (do not require) that non−MIPS16 code be link-compatible with MIPS16 code.

For example, non−MIPS16 code cannot jump directly to MIPS16 code; it must either use a call or an indirect jump. **−minterlink−mips16** therefore disables direct jumps unless GCC knows that the target of the jump is not MIPS16.

−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 <**http://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 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, but all 64 bits are saved.

−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** will generally make 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 will only work if the GOT is smaller than about 64k. Anything larger will cause 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**. It should then work with very large GOTs, although it will also be less efficient, since it will take 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.

−mhard−float

Use floating-point coprocessor instructions.

−msoft−float

Do not use floating-point coprocessor instructions. Implement floating-point calculations using library calls instead.

−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.

−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 will use 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**.

−mmt
−mno−mt
Use (do not use) MT Multithreading 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 `int`s, `long`s and pointers depends on the ABI. All the supported ABIs use 32−bit `int`s. The n64 ABI uses 64−bit `long`s, as does the 64−bit EABI; the others use 32−bit `long`s. Pointers are the same size as `long`s, 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 access the data more efficiently; 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 will be in a small data section if 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 `$gp` register might not hold the value of `_gp`. For example, if the code is part of a library that might be used in a boot monitor, programs that call boot monitor routines will pass an unknown value in `$gp`. (In such situations, the boot monitor itself would usually be 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
−mno−explicit−relocs
Use (do not use) assembler relocation operators when dealing with symbolic addresses. The alternative, selected by **−mno−explicit−relocs**, is to use assembler macros instead.

−mexplicit−relocs is the default if GCC was configured to use an assembler that supports relocation operators.

−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**.

−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.

−mfused−madd
−mno−fused−madd
Enable (disable) use of the floating-point multiply-accumulate instructions, when they are available. The default is **−mfused−madd**.

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.

−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−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 will avoid 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 was 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 will overwrite 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 will 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 will not be generated by default because the MIPS32 and MIPS64 architectures specifically deprecate their use.

−mfp−exceptions
−mno−fp−exceptions

Specifies whether FP exceptions are enabled. This affects how we schedule FP instructions 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 will align 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) will be generated when `__builtin___clear_cache()` is compiled.

This option defaults to `−mno−synci`, but the default can be overridden by configuring 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 will 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 supports 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**.

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 would 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__**.

−mnoliw
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__**.

−mnosetlb
Do not allow the compiler to generate *SETLB* or *Lcc* instructions. This option defines the preprocessor macro **__NO_SETLB__**.

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.)

−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.

−m45
Generate code for a PDP−11/45. This is the default.

−m10
Generate code for a PDP−11/10.

−mbcopy−builtin
Use inline `movmemhi` patterns for copying memory. This is the default.

−mbcopy
Do not use inline `movmemhi` patterns for copying memory.

−mint16
−mno−int32
Use 16−bit `int`. This is the default.

−mint32
−mno−int16
Use 32−bit `int`.

−mfloat64
−mno−float32
Use 64−bit `float`. This is the default.

−mfloat32
−mno−float64
Use 32−bit `float`.

−mabshi
Use `abshi2` pattern. This is the default.

−mno−abshi
Do not use `abshi2` pattern.

−mbranch−expensive
Pretend that branches are expensive. This is for experimenting with code generation only.

−mbranch−cheap
Do not pretend that branches are expensive. This is the default.

−munix−asm
Use Unix assembler syntax. This is the default when configured for **pdp11−*−bsd**.

−mdec−asm
Use DEC assembler syntax. This is the default when configured for any PDP−11 target other than **pdp11−*−bsd**.

picoChip Options

These **−m** options are defined for picoChip implementations:

−mae=*ae_type*
Set the instruction set, register set, and instruction scheduling parameters for array element type *ae_type*. Supported values for *ae_type* are **ANY**, **MUL**, and **MAC**.

−mae=ANY selects a completely generic AE type. Code generated with this option will run on any of the other AE types. The code will not be as efficient as it would be if compiled for a specific AE type, and some types of operation (e.g., multiplication) will not work properly on all types of AE.

−mae=MUL selects a MUL AE type. This is the most useful AE type for compiled code, and is the default.

−mae=MAC selects a DSP-style MAC AE. Code compiled with this option may suffer from poor performance of byte (char) manipulation, since the DSP AE does not provide hardware support for byte load/stores.

−msymbol−as−address
Enable the compiler to directly use a symbol name as an address in a load/store instruction, without first loading it into a register. Typically, the use of this option will generate larger programs, which run faster than when the option isn't used. However, the results vary from program to program, so it is left as a user option, rather than being permanently enabled.

−mno−inefficient−warnings
Disables warnings about the generation of inefficient code. These warnings can be generated, for example, when compiling code that performs byte-level memory operations on the MAC AE type. The MAC AE has no hardware support for byte-level memory operations, so all byte load/stores must be synthesized from word load/store operations. This is inefficient and a warning will be generated indicating to the programmer that they should rewrite the code to avoid byte operations, or to target an AE type that has the necessary hardware support. This option enables the warning to be turned off.

PowerPC Options

These are listed under

RL78 Options

−msim
Links in additional target libraries to support operation within a simulator.

−mmul=none
−mmul=g13
−mmul=rl78
Specifies the type of hardware multiplication support to be used. The default is `none`, which uses software multiplication functions. The `g13` option is for the hardware multiply/divide peripheral only on the RL78/G13 targets. The `rl78` option is for the standard hardware multiplication defined in the RL78 software manual.

IBM RS/6000 and PowerPC Options

These **−m** options are defined for the IBM RS/6000 and PowerPC:

−mpower
−mno−power
−mpower2
−mno−power2
−mpowerpc
−mno−powerpc
−mpowerpc−gpopt
−mno−powerpc−gpopt
−mpowerpc−gfxopt
−mno−powerpc−gfxopt
−mpowerpc64
−mno−powerpc64
−mmfcrf
−mno−mfcrf
−mpopcntb
−mno−popcntb
−mpopcntd
−mno−popcntd
−mfprnd
−mno−fprnd
−mcmpb
−mno−cmpb
−mmfpgpr

−mno−mfpgpr
−mhard−dfp
−mno−hard−dfp

GCC supports two related instruction set architectures for the RS/6000 and PowerPC. The *POWER* instruction set are those instructions supported by the **rios** chip set used in the original RS/6000 systems and the *PowerPC* instruction set is the architecture of the Freescale MPC5xx, MPC6xx, MPC8xx microprocessors, and the IBM 4xx, 6xx, and follow-on microprocessors.

Neither architecture is a subset of the other. However there is a large common subset of instructions supported by both. An MQ register is included in processors supporting the POWER architecture.

You use these options to specify which instructions are available on the processor you are using. The default value of these options is determined when configuring GCC. Specifying the **−mcpu=***cpu_type* overrides the specification of these options. We recommend you use the **−mcpu=***cpu_type* option rather than the options listed above.

The **−mpower** option allows GCC to generate instructions that are found only in the POWER architecture and to use the MQ register. Specifying **−mpower2** implies **−power** and also allows GCC to generate instructions that are present in the POWER2 architecture but not the original POWER architecture.

The **−mpowerpc** option allows GCC to generate instructions that are found only in the 32−bit subset of the PowerPC architecture. Specifying **−mpowerpc−gpopt** implies **−mpowerpc** and also allows GCC to use the optional PowerPC architecture instructions in the General Purpose group, including floating-point square root. Specifying **−mpowerpc−gfxopt** implies **−mpowerpc** and also allows GCC to use the optional PowerPC architecture instructions in the Graphics group, including floating-point select.

The **−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 **−mmfpgpr** option allows GCC to generate the FP move to/from general-purpose register instructions implemented on the POWER6X processor and other processors that support the extended 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**.

If you specify both **−mno−power** and **−mno−powerpc**, GCC will use only the instructions in the common subset of both architectures plus some special AIX common-mode calls, and will not use the MQ register. Specifying both **−mpower** and **−mpowerpc** permits GCC to use any instruction from either architecture and to allow use of the MQ register; specify this for the Motorola MPC601.

−mnew−mnemonics
−mold−mnemonics

Select which mnemonics to use in the generated assembler code. With **−mnew−mnemonics**, GCC uses the assembler mnemonics defined for the PowerPC architecture. With **−mold−mnemonics** it uses the assembler mnemonics defined for the POWER architecture. Instructions defined in only one architecture have only one mnemonic; GCC uses that mnemonic irrespective of which of these options is specified.

GCC defaults to the mnemonics appropriate for the architecture in use. Specifying **−mcpu=***cpu_type* sometimes overrides the value of these option. Unless you are building a cross-compiler, you should normally not specify either **−mnew−mnemonics** or **−mold−mnemonics**, but should instead accept the default.

−mcpu=*cpu_type*
Set architecture type, register usage, choice of mnemonics, and instruction scheduling parameters for machine type *cpu_type*. Supported values for *cpu_type* are **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**, **ec603e**, **G3**, **G4**, **G5**, **titan**, **power**, **power2**, **power3**, **power4**, **power5**, **power5+**, **power6**, **power6x**, **power7**, **common**, **powerpc**, **powerpc64**, **rios**, **rios1**, **rios2**, **rsc**, and **rs64**.

−mcpu=common selects a completely generic processor. Code generated under this option will run on any POWER or PowerPC processor. GCC will use only the instructions in the common subset of both architectures, and will not use the MQ register. GCC assumes a generic processor model for scheduling purposes.

−mcpu=power, **−mcpu=power2**, **−mcpu=powerpc**, and **−mcpu=powerpc64** specify generic POWER, POWER2, pure 32−bit PowerPC (i.e., not MPC601), and 64−bit PowerPC architecture machine types, with an appropriate, generic processor model assumed for scheduling purposes.

The other options specify a specific processor. Code generated under those options will run 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 −mnew−mnemonics −mpopcntb −mpopcntd −mpower −mpower2 −mpowerpc64 −mpowerpc−gpopt −mpowerpc−gfxopt −msingle−float −mdouble−float −msimple−fpu −mstring −mmulhw −mdlmzb −mmfpgpr −mvsx

The particular options set for any particular CPU will vary 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, register usage, or choice of mnemonics, as **−mcpu=***cpu_type* would. The same values for *cpu_type* are used for **−mtune** as for **−mcpu**. If both are specified, the code generated will use the architecture, registers, and mnemonics 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.

−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.

−mvrsave
−mno−vrsave
Generate VRSAVE instructions when generating AltiVec code.

−mgen−cell−microcode
Generate Cell microcode instructions

−mwarn−cell−microcode
Warning when a Cell microcode instruction is going to emitted. An example of a Cell microcode instruction is a variable shift.

−msecure−plt
Generate code that allows ld and ld.so to build executables and shared libraries with non-exec .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.

−misel
−mno−isel
This switch enables or disables the generation of ISEL instructions.

−misel=*yes/no*
This switch has been deprecated. Use **−misel** and **−mno−isel** instead.

−mspe
−mno−spe
This switch enables or disables the generation of SPE simd instructions.

−mpaired
−mno−paired
This switch enables or disables the generation of PAIRED simd instructions.

−mspe=*yes/no*
This option has been deprecated. Use **−mspe** and **−mno−spe** instead.

−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.

−mfloat−gprs=*yes/single/double/no*
−mfloat−gprs
This switch enables or disables the generation of floating-point operations on the general-purpose registers for architectures that support it.

The argument *yes* or *single* enables the use of single-precision floating-point operations.

The argument *double* enables the use of single and double-precision floating-point operations.

The argument *no* disables floating-point operations on the general-purpose registers.

This option is currently only available on the MPC854x.

−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 will allocate at least one TOC entry for each unique non-automatic variable reference in your program. GCC will also place floating-point constants in the TOC. However, only 16,384 entries are available in the TOC.

If you receive a linker error message that saying you have overflowed the available TOC space, you can reduce the amount of TOC space used with the **−mno−fp−in−toc** and **−mno−sum−in−toc** options. **−mno−fp−in−toc** prevents GCC from putting floating-point constants in the TOC and **−mno−sum−in−toc** forces GCC to generate code to calculate the sum of an address and a constant at run time instead of putting that sum into the TOC. You may specify one or both of these options. Each causes GCC to produce very slightly slower and larger code at the expense of conserving TOC space.

If you still run out of space in the TOC even when you specify both of these options, specify **−mminimal−toc** instead. This option causes GCC to make only one TOC entry for every file. When you specify this option, GCC will produce code that is slower and larger but which uses extremely little TOC space. You may wish to use this option only on files that contain less frequently executed code.

−maix64
−maix32

Enable 64−bit AIX ABI and calling convention: 64−bit pointers, 64−bit `long` type, and the infrastructure needed to support them. Specifying **−maix64** implies **−mpowerpc64** and **−mpowerpc**, while **−maix32** disables the 64−bit ABI and implies **−mno−powerpc64**. GCC defaults to **−maix32**.

−mxl−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.

−msingle−float
−mdouble−float
Generate code for single− or double-precision floating-point operations. **−mdouble−float** implies **−msingle−float**.

−msimple−fpu
Do not generate sqrt and div instructions for hardware floating-point unit.

−mfpu
Specify type of floating-point unit. Valid values are *sp_lite* (equivalent to −msingle−float −msimple−fpu), *dp_lite* (equivalent to −mdouble−float −msimple−fpu), *sp_full* (equivalent to −msingle−float), and *dp_full* (equivalent to −mdouble−float).

−mxilinx−fpu
Perform optimizations for the floating-point unit on Xilinx PPC 405/440.

−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.

−mstring
−mno−string
Generate code that uses (does not use) the load string instructions and the store string word instructions to save multiple registers and do small block moves. These instructions are generated by default on POWER systems, and not generated on PowerPC systems. Do not use **−mstring** on little-endian PowerPC systems, since those instructions do not work when the processor is in little-endian mode. The exceptions are PPC740 and PPC750 which permit 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 targetting 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 targetting 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 targetting 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 will be 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 and Mac OS X 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 *0/1/2* to assign *no/highest/second−highest* 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 latency >= *number* is costly.

−minsert−sched−nops=*scheme*
This option controls which nop insertion scheme will be 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 adheres to the March 1995 draft of the System V Application Binary Interface, PowerPC processor supplement. This is the default unless you configured GCC using **powerpc−*−eabiaix**.

−mcall−sysv−eabi
−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.

−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*, *spe*, *no-spe*, *ibmlongdouble*, *ieeelongdouble*.

−mabi=spe
Extend the current ABI with SPE ABI extensions. This does not change the default ABI, instead it adds the SPE ABI extensions to the current ABI.

−mabi=no−spe
Disable Booke SPE ABI extensions for the current ABI.

−mabi=ibmlongdouble
Change the current ABI to use IBM extended-precision long double. This is a PowerPC 32−bit SYSV ABI option.

−mabi=ieeelongdouble
Change the current ABI to use IEEE extended-precision long double. This is a PowerPC 32−bit Linux ABI option.

−mprototype
−mno−prototype
On System V.4 and embedded PowerPC systems assume that all calls to variable argument functions are properly prototyped. Otherwise, the compiler must insert an instruction before every non prototyped call to set or clear bit 6 of the condition code register (*CR*) to indicate whether floating-point values were passed in the floating-point registers in case the function takes variable arguments. With **−mprototype**, only calls to prototyped variable argument functions will set or clear the bit.

−msim
On embedded PowerPC systems, assume that the startup module is called *sim−crt0.o* and that the standard C libraries are *libsim.a* and *libc.a*. This is the default for **powerpc−*−eabisim** configurations.

−mmvme
On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libmvme.a* and *libc.a*.

−mads
On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libads.a* and *libc.a*.

−myellowknife
On embedded PowerPC systems, assume that the startup module is called *crt0.o* and the standard C libraries are *libyk.a* and *libc.a*.

−mvxworks
On System V.4 and embedded PowerPC systems, specify that you are compiling for a VxWorks system.

−memb
On embedded PowerPC systems, set the *PPC_EMB* bit in the ELF flags header to indicate that **eabi** extended relocations are used.

−meabi
−mno−eabi
On System V.4 and embedded PowerPC systems do (do not) adhere to the Embedded Applications Binary Interface (eabi) which is a set of modifications to the System V.4 specifications. Selecting **−meabi** means that the stack is aligned to an 8−byte boundary, a function `__eabi` is called to from `main` to set up the eabi environment, and the **−msdata** option can use both `r2` and `r13` to point to two separate small data areas. Selecting **−mno−eabi** means that the stack is aligned to a 16−byte boundary, do not call an initialization function from `main`, and the **−msdata** option will only use `r13` to point to a single small data area. The **−meabi** option is on by default if you configured GCC using one of the **powerpc*−*−eabi*** options.

−msdata=eabi
On System V.4 and embedded PowerPC systems, put small initialized `const` global and static data in the **.sdata2** section, which is pointed to by register `r2`. Put small initialized non−`const` global and

static data in the **.sdata** section, which is pointed to by register `r13`. Put small uninitialized global and static data in the **.sbss** section, which is adjacent to the **.sdata** section. The **−msdata=eabi** option is incompatible with the **−mrelocatable** option. The **−msdata=eabi** option also sets the **−memb** option.

−msdata=sysv

On System V.4 and embedded PowerPC systems, put small global and static data in the **.sdata** section, which is pointed to by register `r13`. Put small uninitialized global and static data in the **.sbss** section, which is adjacent to the **.sdata** section. The **−msdata=sysv** option is incompatible with the **−mrelocatable** option.

−msdata=default
−msdata

On System V.4 and embedded PowerPC systems, if **−meabi** is used, compile code the same as **−msdata=eabi**, otherwise compile code the same as **−msdata=sysv**.

−msdata=data

On System V.4 and embedded PowerPC systems, put small global 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.

−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.

−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 more expensive calling sequence is required. This is required for calls further than 32 megabytes (33,554,432 bytes) from the current location. A short call will be 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 Darwin/PPC systems, `#pragma longcall` will generate ‘‘jbsr callee, L42’’, plus a ‘‘branch island’’ (glue code). The two target addresses represent the callee and the ‘‘branch island’’. The Darwin/PPC linker will prefer the first address and generate a ‘‘bl callee’’ if the PPC ‘‘bl’’ instruction will reach the callee directly; otherwise, the linker will generate ‘‘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, we may cause GCC to ignore all longcall specifications when the linker is known to generate glue.

−mtls−markers
−mno−tls−markers
Mark (do not mark) calls to `__tls_get_addr` with a relocation specifying the function argument. The relocation allows ld to reliably associate function call with argument setup instructions for TLS optimization, which in turn allows gcc to better schedule the sequence.

−pthread
Adds support for multithreading with the *pthreads* library. This option sets flags for both the preprocessor and linker.

−mrecip
−mno−recip
This option will enable GCC to use 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**, **−finite−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 allows to control 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** would enable the 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** or **−mcpu=power7** 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 will currently emit 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** have to be enabled. The

MASS libraries will have to 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 will not be able to call through pointers to nested functions or pointers to functions compiled in other languages that use the static chain if you use the **−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.

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 will only be generated for 32−bit floating-point values however, so if the **−m64bit−doubles** option is in use then the FPU hardware will not be used for doubles.

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 which once could have been held in the reserved register are now pushed onto the stack.

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 generating it is not the default option.

−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 will attempt 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` will be 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 will 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.

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 will will ensure 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* will be 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* will be 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−2 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 **−mmo−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**.

−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 will run on *cpu-type*, which is the name of a system representing a certain processor type. Possible values for *cpu-type* are **g5**, **g6**, **z900**, **z990**, **z9−109**, **z9−ec** and **z10**. When generating code using the instructions available on z/Architecture, the default is **−march=z900**. Otherwise, the default is **−march=g5**.

−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.

−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 s390 back end emits additional instructions in the function prologue which trigger a trap if the stack size is *stack-guard* bytes above the *stack-size* (remember that the stack on s390 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*.

Score Options

These options are defined for Score implementations:

−meb
Compile code for big-endian mode. This is the default.

−mel
Compile code for little-endian mode.

−mnhwloop
Disable generate bcnz instruction.

−muls
Enable generate unaligned load and store instruction.

−mmac
Enable the use of multiply-accumulate instructions. Disabled by default.

−mscore5
Specify the SCORE5 as the target architecture.

−mscore5u
Specify the SCORE5U of the target architecture.

−mscore7
Specify the SCORE7 as the target architecture. This is the default.

−mscore7d
Specify the SCORE7D as the target architecture.

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.

−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 will not work unless you recompile it first with **−mdalign**.

−mrelax
Shorten some address references at link time, when possible; uses the linker option **−relax**.

−mbigtable
Use 32−bit offsets in `switch` tables. The default is to use 16−bit offsets.

−mbitops
Enable the use of bit manipulation instructions on SH2A.

−mfmovd
Enable the use of the instruction `fmovd`. Check **−mdalign** for alignment constraints.

−mhitachi
Comply with the calling conventions defined by Renesas.

−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 **−mhitachi** 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 implcit 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 will manipulate the instruction cache address array directly with an associative write. This not only requires privileged mode, but it will also fail 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.

−msoft−atomic
Generate GNU/Linux compatible gUSA software atomic sequences for the atomic built-in functions. The generated atomic sequences require support from the interrupt / exception handling code of the system and are only suitable for single-core systems. They will not perform correctly on multi-core systems. This option is enabled by default when the target is `sh−*−linux*`. For details on the atomic built-in functions see **__atomic Builtins**.

−mspace
Optimize for space instead of speed. Implied by **−Os**.

−mprefergot
When generating position-independent code, emit function calls using the Global Offset Table instead of the Procedure Linkage Table.

−musermode
Don't generate privileged mode only code; implies −mno−inline−ic_invalidate if the inlined code would not work in user mode. This is the default when the target is `sh−*−linux*`.

−multcost=*number*
Set the cost to assume for a multiply insn.

−mdiv=*strategy*
Set the division strategy to use for SHmedia code. *strategy* must be one of: call, call2, fp, inv, inv:minlat, inv20u, inv20l, inv:call, inv:call2, inv:fp . ''fp'' performs the operation in floating point. This has a very high latency, but needs only a few instructions, so it might be a good choice if your code has enough easily-exploitable ILP to allow the compiler to schedule the floating-point instructions together with other instructions. Division by zero causes a floating-point exception. ''inv'' uses integer operations to calculate the inverse of the divisor, and then multiplies the dividend with the inverse. This strategy allows cse and hoisting of the inverse calculation. Division by zero calculates an unspecified result, but does not trap. ''inv:minlat'' is a variant of ''inv'' where if no cse / hoisting opportunities have been found, or if the entire operation has been hoisted to the same place, the last stages of the inverse calculation are intertwined with the final multiply to reduce the overall latency, at the expense of using a few more instructions, and thus offering fewer scheduling opportunities with other code. ''call'' calls a library function that usually implements the inv:minlat strategy. This gives high code density for m5−*media−nofpu compilations. ''call2'' uses a different entry point of the same library function, where it assumes that a pointer to a lookup table has already been set up, which exposes the pointer load to cse / code hoisting optimizations. ''inv:call'', ''inv:call2'' and ''inv:fp'' all use the ''inv'' algorithm for initial code generation, but if the code stays unoptimized, revert to the ''call'', ''call2'', or ''fp'' strategies, respectively. Note that the potentially-trapping side effect of division by zero is carried by a separate instruction, so it is possible that all the integer instructions are hoisted out, but the marker for the side effect stays where it is. A recombination to fp operations or a call is not possible in that case. ''inv20u'' and ''inv20l'' are variants of the ''inv:minlat'' strategy. In the case that the inverse calculation was nor separated from the multiply, they speed up division where the dividend fits into 20 bits (plus sign where applicable), by inserting a test to skip a number of operations in this case; this test slows down the case of larger dividends. inv20u assumes the case of a such a small dividend to be unlikely, and inv20l assumes it to be likely.

−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 affect the name used in the call and inv:call division strategies, and the compiler will still expect the same sets of input/output/clobbered registers as if this option was 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 can not use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

−madjust−unroll
Throttle unrolling to avoid thrashing target registers. This option only has an effect if the gcc code base supports the TARGET_ADJUST_UNROLL_MAX target hook.

−mindexed−addressing
Enable the use of the indexed addressing mode for SHmedia32/SHcompact. This is only safe if the hardware and/or OS implement 32−bit wrap-around semantics for the indexed addressing mode. The architecture allows the implementation of processors with 64−bit MMU, which the OS could use to get 32−bit addressing, but since no current hardware implementation supports this or any other way to make the indexed addressing mode safe to use in the 32−bit ABI, the default is **−mno−indexed−addressing**.

−mgettrcost=*number*
Set the cost assumed for the gettr instruction to *number*. The default is 2 if **−mpt−fixed** is in effect, 100 otherwise.

−mpt−fixed
Assume pt* instructions won't trap. This will generally generate better scheduled code, but is unsafe on current hardware. The current architecture definition says that ptabs and ptrel trap when the target anded with 3 is 3. This has the unintentional effect of making it unsafe to schedule ptabs / ptrel before a branch, or hoist it out of a loop. For example, __do_global_ctors, a part of libgcc that runs constructors at program startup, calls functions in a list which is delimited by −1. With the −mpt−fixed option, the ptabs will be done before testing against −1. That means that all the constructors will be run a bit quicker, but when the loop comes to the end of the list, the program crashes because ptabs loads −1 into a target register. Since this option is unsafe for any hardware implementing the current architecture specification, the default is −mno−pt−fixed. Unless the user specifies a specific cost with **−mgettrcost**, −mno−pt−fixed also implies **−mgettrcost=100**; this deters register allocation using target registers for storing ordinary integers.

−minvalid−symbols
Assume symbols might be invalid. Ordinary function symbols generated by the compiler will always be valid to load with movi/shori/ptabs or movi/shori/ptrel, but with assembler and/or linker tricks it is possible to generate symbols that will cause ptabs / ptrel to trap. This option is only meaningful when **−mno−pt−fixed** is in effect. It will then prevent cross-basic-block cse, hoisting and most scheduling of symbol loads. The default is **−mno−invalid−symbols**.

−mbranch−cost=*num*
Assume *num* to be the cost for a branch instruction. Higher numbers will 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.

−mcbranchdi
Enable the `cbranchdi4` instruction pattern.

−mcmpeqdi
Emit the `cmpeqdi_t` instruction pattern even when **−mcbranchdi** is in effect.

−mfused−madd
Allow the usage of the `fmac` instruction (floating-point multiply-accumulate) if the processor type supports it. Enabling this option might generate code that produces different numeric floating-point results compared to strict IEEE 754 arithmetic.

−mpretend−cmove
Prefer zero-displacement conditional branches for conditional move instruction patterns. This can result in faster code on the SH4 processor.

Solaris 2 Options

These **−m** options are supported on Solaris 2:

−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 will 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

Add support for multithreading using the POSIX threads library. This option sets flags for both the preprocessor and linker. This option does not affect the thread safety of object code produced by the compiler or that of libraries supplied with it.

−pthread

This is a synonym for **−pthreads**.

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. 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 will be 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.

−mno−faster−structs
−mfaster−structs

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

−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**, **sparclite**, **f930**, **f934**, **sparclite86x**, **sparclet**, **tsc701**, **v9**, **ultrasparc**, **ultrasparc3**, **niagara**, **niagara2**, **niagara3**, and **niagara4**.

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
v8	supersparc, hypersparc, leon
sparclite	f930, f934, sparclite86x
sparclet	tsc701
v9	ultrasparc, ultrasparc3, niagara, niagara2, niagara3, niagara4

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.

−mtune=*cpu_type*

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

The same values for **−mcpu=***cpu_type* 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**, **f930**, **f934**, **sparclite86x**, **tsc701**, **ultrasparc**, **ultrasparc3**, **niagara**, **niagara2**, **niagara3** and **niagara4**. 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 targetting 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 targetting a cpu that supports such instructions, such as niagara−3 and later. Setting **−mvis3** also sets **−mvis2** and **−mvis**.

−mpopc
−mno−popc

With **−mpopc**, GCC generates code that takes advantage of the UltraSPARC population count instruction. The default is **−mpopc** when targetting a cpu that supports such instructions, such as Niagara−2 and later.

−mfmaf
−mno−fmaf

With **−mfmaf**, GCC generates code that takes advantage of the UltraSPARC Fused Multiply-Add Floating-point extensions. The default is **−mfmaf** when targetting a cpu that supports such instructions, such as Niagara−3 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).

These **−m** options are supported in addition to the above on SPARC−V9 processors in 64−bit environments:

−mlittle−endian
Generate code for a processor running in little-endian mode. It is only available for a few configurations and most notably not on Solaris and Linux.

−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.

SPU Options

These **−m** options are supported on the SPU:

−mwarn−reloc
−merror−reloc
The loader for SPU does not handle dynamic relocations. By default, GCC will give an error when it generates code that requires a dynamic relocation. **−mno−error−reloc** disables the error, **−mwarn−reloc** will generate a warning instead.

−msafe−dma
−munsafe−dma
Instructions that initiate or test completion of DMA must not be reordered with respect to loads and stores of the memory that is being accessed. Users typically address this problem using the volatile keyword, but that can lead to inefficient code in places where the memory is known to not change. Rather than mark the memory as volatile we treat the DMA instructions as potentially effecting all memory. With **−munsafe−dma** users must use the volatile keyword to protect memory accesses.

−mbranch−hints
By default, GCC will generate a branch hint instruction to avoid pipeline stalls for always taken or probably taken branches. A hint will not be generated closer than 8 instructions away from its branch. There is little reason to disable them, except for debugging purposes, or to make an object a little bit smaller.

−msmall−mem
−mlarge−mem
By default, GCC generates code assuming that addresses are never larger than 18 bits. With **−mlarge−mem** code is generated that assumes a full 32−bit address.

−mstdmain
By default, GCC links against startup code that assumes the SPU-style main function interface (which has an unconventional parameter list). With **−mstdmain**, GCC will link your program against startup code that assumes a C99−style interface to `main`, including a local copy of `argv` strings.

−mfixed−range=*register-range*
Generate code treating the given register range as fixed registers. A fixed register is one that the register allocator can not use. This is useful when compiling kernel code. A register range is specified as two registers separated by a dash. Multiple register ranges can be specified separated by a comma.

−mea32
−mea64
Compile code assuming that pointers to the PPU address space accessed via the `__ea` named address space qualifier are either 32 or 64 bits wide. The default is 32 bits. As this is an ABI changing option, all object code in an executable must be compiled with the same setting.

−maddress−space−conversion
−mno−address−space−conversion
Allow/disallow treating the `__ea` address space as superset of the generic address space. This enables explicit type casts between `__ea` and generic pointer as well as implicit conversions of generic pointers to `__ea` pointers. The default is to allow address space pointer conversions.

−mcache−size=*cache-size*
This option controls the version of libgcc that the compiler links to an executable and selects a software-managed cache for accessing variables in the `__ea` address space with a particular cache size. Possible options for *cache-size* are **8**, **16**, **32**, **64** and **128**. The default cache size is 64KB.

−matomic−updates
−mno−atomic−updates
This option controls the version of libgcc that the compiler links to an executable and selects whether atomic updates to the software-managed cache of PPU-side variables are used. If you use atomic updates, changes to a PPU variable from SPU code using the `__ea` named address space qualifier will not interfere with changes to other PPU variables residing in the same cache line from PPU code. If you do not use atomic updates, such interference may occur; however, writing back cache lines will be more efficient. The default behavior is to use atomic updates.

−mdual−nops
−mdual−nops=*n*
By default, GCC will insert nops to increase dual issue when it expects it to increase performance. *n* can be a value from 0 to 10. A smaller *n* will insert fewer nops. 10 is the default, 0 is the same as **−mno−dual−nops**. Disabled with **−Os**.

−mhint−max−nops=*n*
Maximum number of nops to insert for a branch hint. A branch hint must be at least 8 instructions away from the branch it is effecting. GCC will insert up to *n* nops to enforce this, otherwise it will not generate the branch hint.

−mhint−max−distance=*n*
The encoding of the branch hint instruction limits the hint to be within 256 instructions of the branch it is effecting. By default, GCC makes sure it is within 125.

−msafe−hints
Work around a hardware bug that causes the SPU to stall indefinitely. By default, GCC will insert the `hbrp` instruction to make sure this stall won't happen.

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.

TILE-Gx Options

These **−m** options are supported on the TILE-Gx:

−mcpu=*name*
Selects the type of CPU to be targeted. Currently the only supported type is **tilegx**.

−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.

TILEPro Options

These **−m** options are supported on the TILEPro:

−mcpu=*name*
Selects the type of CPU to be targeted. Currently the only supported type is **tilepro**.

−m32
Generate code for a 32−bit environment, which sets int, long, and pointer to 32 bits. This is the only supported behavior so the flag is essentially ignored.

V850 Options

These **−m** options are defined for V850 implementations:

−mlong−calls
−mno−long−calls
Treat all calls as being far away (near). If calls are assumed to be far away, the compiler will always load the functions address up into a register, and call indirect through the pointer.

−mno−ep
−mep
Do not optimize (do optimize) basic blocks that use the same index pointer 4 or more times to copy pointer into the `ep` register, and use the shorter `sld` and `sst` instructions. The **−mep** option is on by default if you optimize.

−mno−prolog−function
−mprolog−function
Do not use (do use) external functions to save and restore registers at the 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.

−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 will cause r2 and r5 to be used in the code generated by the compiler. This setting is the default.

−mno−app−regs
This option will cause r2 and r5 to be treated as fixed registers.

−mv850e2v3
Specify that the target processor is the V850E2V3. The preprocessor constants **__v850e2v3__** will be defined if this option is used.

−mv850e2
Specify that the target processor is the V850E2. The preprocessor constants **__v850e2__** will be defined if this option is used.

−mv850e1
Specify that the target processor is the V850E1. The preprocessor constants **__v850e1__** and **__v850e__** will be 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__** will be defined if this option is used.

If neither **−mv850** nor **−mv850e** nor **−mv850e1** nor **−mv850e2** nor **−mv850e2v3** are defined then a default target processor will be chosen and the relevant **__v850*__** preprocessor constant will be defined.

The preprocessor constants **__v850** and **__v851__** are always defined, regardless of which processor variant is the target.

−mdisable−callt
This option will suppress generation of the CALLT instruction for the v850e, v850e1, v850e2 and v850e2v3 flavors of the v850 architecture. The default is **−mno−disable−callt** which allows the CALLT instruction to be used.

VAX Options

These **−m** options are defined for the VAX:

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

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

−mg
Output code for G−format floating-point numbers instead of D−format.

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__`.

−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.

x86−64 Options

These are listed under

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
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.

−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 will be performed. The default is **−mtarget−align**. These options do not affect the treatment of auto-aligned instructions like `LOOP`, which the assembler will always align, either by widening density instructions or by inserting no-op 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 will still show direct call instructions−−−look at the disassembled object code to see the actual instructions. Note that the assembler will use an indirect call for every cross-file call, not

just those that really will be out of range.

zSeries Options

These are listed under

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** would be **−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.

−fbounds−check
For front ends that support it, generate additional code to check that indices used to access arrays are within the declared range. This is currently only supported by the Java and Fortran front ends, where this option defaults to true and false respectively.

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

−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. This option is enabled by default for the Java front end, as required by the Java language specification.

−fexceptions
Enable exception handling. Generates extra code needed to propagate exceptions. For some targets, this implies GCC will generate frame unwind information for all functions, which can produce significant data size overhead, although it does not affect execution. If you do not specify this option, GCC will enable it by default for languages like C++ that normally require exception handling, and disable it for languages like C that do not normally require it. However, you may need to enable this option when compiling C code that needs to interoperate properly with exception handlers written in C++. You may also wish to disable this option if you are compiling older C++ programs that don't use exception handling.

−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`.

−funwind−tables
Similar to **−fexceptions**, except that it will just generate any needed static data, but will not affect the generated code in any other way. You will normally not enable this option; instead, a language processor that needs this handling would enable it on your behalf.

−fasynchronous−unwind−tables
Generate unwind table in dwarf2 format, if supported by target machine. The table is exact at each instruction boundary, so it can be used for stack unwinding from asynchronous events (such as debugger or garbage collector).

−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 will be equivalent to the smallest integer type that has enough room.

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−double

Use the same size for `double` as for `float`.

Warning: the **−fshort−double** 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.

−fno−common

In C code, controls the placement of uninitialized global variables. Unix C compilers have traditionally permitted multiple definitions of such variables in different compilation units by placing the variables in a common block. This is the behavior specified by **−fcommon**, and is the default for GCC on most targets. On the other hand, this behavior is not required by ISO C, and on some targets may carry a speed or code size penalty on variable references. The **−fno−common** option specifies that the compiler should place uninitialized global variables in the data section of the object file, rather than generating them as common blocks. This has the effect that if the same variable is declared (without `extern`) in two different compilations, you will get a multiple-definition error when you link them. In this case, you must compile with **−fcommon** instead. Compiling with **−fno−common** is useful on targets for which it provides better performance, or if you wish to verify that the program will work on other systems that always treat uninitialized variable declarations this way.

−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.

−frecord−gcc−switches

This switch causes the command line that was 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 and 32k on the m68k and RS/6000. The 386 has no such limit.)

Position-independent code requires special support, and therefore works only on certain machines. For the 386, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM RS/6000 is always position-independent.

When this flag is set, the macros `__pic__` and `__PIC__` are defined to 1.

−fPIC

If supported for the target machine, emit position-independent code, suitable for dynamic linking and avoiding any limit on the size of the global offset table. This option makes a difference on the m68k, PowerPC and SPARC.

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

When this flag is set, the macros `__pic__` and `__PIC__` are defined to 2.

−fpie
−fPIE

These options are similar to **−fpic** and **−fPIC**, but generated position independent code can be only linked into executables. Usually these options are used when **−pie** GCC option will be used during linking.

−fpie and **−fPIE** both define the macros `__pie__` and `__PIE__`. The macros have the value 1 for **−fpie** and 2 for **−fPIE**.

−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.

−ffixed−*reg*

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

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

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

−fcall−used−*reg*

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

It is an error to used this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model will produce disastrous results.

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

−fcall−saved−*reg*

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

It is an error to used this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model will produce disastrous results.

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

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

−fpack−struct[=*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 will be 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.

−finstrument−functions

Generate instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions will be called with the address of the current function and its call site. (On some platforms, `__builtin_return_address` does not work beyond the current function, so the call site information may not be available to the profiling functions otherwise.)

```
void __cyg_profile_func_enter (void *this_fn,
                               void *call_site);
void __cyg_profile_func_exit  (void *this_fn,
                               void *call_site);
```

The first argument is the address of the start of the current function, which may be looked up exactly in the symbol table.

This instrumentation is also done for functions expanded inline in other functions. The profiling calls will indicate where, conceptually, the inline function is entered and exited. This means that addressable versions of such functions must be available. If all your uses of a function are expanded inline, this may mean an additional expansion of code size. If you use **extern inline** in your C code, an addressable version of such functions must be provided. (This is normally the case anyways, but if you get lucky and the optimizer always expands the functions inline, you might have gotten away without providing static copies.)

A function may be given the attribute `no_instrument_function`, in which case this instrumentation will not be done. This can be used, for example, for the profiling functions listed above, high-priority interrupt routines, and any functions from which the profiling functions cannot safely be called (perhaps signal handlers, if the profiling routines generate output or allocate memory).

−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
```

will exclude 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.

−fstack−check

Generate code to verify that you do not go beyond the boundary of the stack. You should specify this flag if you are running in an environment with multiple threads, but only rarely need to specify it in a single-threaded environment since stack overflow is automatically detected on nearly all systems if there is only one stack.

Note that this switch does not actually cause checking to be done; the operating system 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 will always be allocated dynamically if their size exceeds a fixed threshold.
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, the performances of the code are 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−limit−register=*reg*
−fstack−limit−symbol=*sym*
−fno−stack−limit

Generate code to ensure that the stack does not grow beyond a certain value, either the value of a register or the address of a symbol. If the stack would grow beyond the value, a signal is raised. For most targets, the signal is raised before the stack overruns the boundary, so it is possible to catch the signal without taking special precautions.

For instance, if the stack starts at 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.

−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 i386 and x86_64 back ends 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.

−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`.

The default without **−fpic** is `initial−exec`; with **−fpic** the default is `global−dynamic`.

−fvisibility=*default|internal|hidden|protected*

Set the default ELF image symbol visibility to the specified option−−−all symbols will be 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 will be `hidden`. The default if **−fvisibility** isn't specified is `default`, i.e., make every symbol public−−−this causes the same behavior as previous versions of GCC.

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 <**http://people.redhat.com/~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 DLL's 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 will 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 will be thrown between DSOs must be explicitly marked with default visibility so that the **type_info** nodes will be unified between the DSOs.

An overview of these techniques, their benefits and how to use them is at <**http://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 the user could 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 will use the most efficient instruction. In the previous example, that might be a 32−bit load instruction, even though that will access bytes that do not contain any portion of the bit-field, or memory-mapped registers unrelated to the one being updated.

If the target requires strict alignment, and honoring the field type would require violating this alignment, a warning is issued. If the field has `packed` attribute, the access is done without honoring the field type. If the field doesn't have `packed` attribute, the access is done honoring the field type. In both cases, GCC assumes that the user knows something about the target hardware that it is unaware of.

The default value of this option is determined by the application binary interface for the target processor.

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 would otherwise be interpreted as a string end or escape.

The **LC_MESSAGES** environment variable specifies the language to use in diagnostic messages.

If the **LC_ALL** environment variable is set, it overrides the value of **LC_CTYPE** and **LC_MESSAGES**;

otherwise, **LC_CTYPE** and **LC_MESSAGES** default to the value of the **LANG** environment variable. If none of these variables are set, GCC defaults to traditional C English behavior.

TMPDIR

If **TMPDIR** is set, it specifies the directory to use for temporary files. GCC uses temporary files to hold the output of one stage of compilation which is to be used as input to the next stage: for example, the output of the preprocessor, which is the input to the compiler proper.

GCC_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 will attempt to figure out an appropriate prefix to use based on the pathname it was invoked with.

If GCC cannot find the subprogram using the specified prefix, it tries looking in the usual places for the subprogram.

The default value of **GCC_EXEC_PREFIX** is *prefix/lib/gcc/* 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 will search *foo/bar* where it would normally search */usr/local/lib/bar*. These alternate directories are searched first; the standard directories come next. 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 can't find the subprograms using **GCC_EXEC_PREFIX**.

LIBRARY_PATH

The value of **LIBRARY_PATH** is a colon-separated list of directories, much like **PATH**. When configured as a native compiler, GCC tries the directories thus specified when searching for special linker files, if it can't find them using **GCC_EXEC_PREFIX**. Linking using GCC also uses these directories when searching for ordinary libraries for the **−l** option (but directories specified with **−L** come first).

LANG

This variable is used to pass locale information to the compiler. One way in which this information is used is to determine the character set to be used when character literals, string literals and comments are parsed in C and C++. When the compiler is configured to allow multibyte characters, the following values for **LANG** are recognized:

C−JIS

Recognize JIS characters.

C−SJIS

Recognize SJIS characters.

C−EUCJP
Recognize EUCJP characters.

If **LANG** is not defined, or if it has some other value, then the compiler will use mblen and mbtowc as defined by the default locale to recognize and translate multibyte characters.

Some additional environments 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.

BUGS

For instructions on reporting bugs, see http://bugs.debian.org/.

FOOTNOTES

1. On some systems, **gcc −shared** needs to build supplementary stub code for constructors to work. On multi-libbed systems, **gcc −shared** must select the correct support libraries to link against. Failing to supply the correct flags may lead to subtle defects. Supplying them in cases where they are not necessary is innocuous.

SEE ALSO

gpl (7), *gfdl* (7), *fsf−funding* (7), *cpp* (1), *gcov* (1), *as* (1), *ld* (1), *gdb* (1), *adb* (1), *dbx* (1), *sdb* (1) and the Info entries for *gcc*, *cpp*, *as*, *ld*, *binutils* and *gdb*.

AUTHOR

See the Info entry for **gcc**, or <**http://gcc.gnu.org/onlinedocs/gcc/Contributors.html**>, for contributors to GCC.

COPYRIGHT