Using the GNU Compiler Collection

For GCC version 4.7.2

(GCC)

Richard M. Stallman and the GCC Developer Community

Published by:

GNU Press
a division of the
Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301 USA

Website: www.gnupress.org
General: press@gnu.org
Orders: sales@gnu.org
Tel 617-542-5942
Fax 617-542-2652

Last printed October 2003 for GCC 3.3.1.
Printed copies are available for $45 each.

Copyright © 1988, 1989, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with the Invariant Sections being "Funding Free Software", the Front-Cover Texts being (a) (see below), and with the Back-Cover Texts being (b) (see below). A copy of the license is included in the section entitled "GNU Free Documentation License".

(a) The FSF's Front-Cover Text is:

A GNU Manual

(b) The FSF's Back-Cover Text is:

You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.

Short Contents

Introduction 1
1 Programming Languages Supported by GCC 3
2 Language Standards Supported by GCC 5
3 GCC Command Options 9
4 C Implementation-defined behavior 299
5 C++ Implementation-defined behavior 307
6 Extensions to the C Language Family 309
7 Extensions to the C++ Language 623
8 GNU Objective-C features 635
9 Binary Compatibility 651
10 `gcov`—a Test Coverage Program 655
11 Known Causes of Trouble with GCC 663
12 Reporting Bugs 679
13 How To Get Help with GCC 681
14 Contributing to GCC Development 683
Funding Free Software 685
The GNU Project and GNU/Linux 687
GNU General Public License 689
GNU Free Documentation License 701
Contributors to GCC 709
Option Index 725
Keyword Index 743

Table of Contents

Introduction 1

1 Programming Languages Supported by GCC 3

2 Language Standards Supported by GCC 5

2.1 C language 5
2.2 C++ language 6
2.3 Objective-C and Objective-C++ languages 7
2.4 Go language 8
2.5 References for other languages 8

3 GCC Command Options 9

3.1 Option Summary 9
3.2 Options Controlling the Kind of Output 23
3.3 Compiling C++ Programs 29
3.4 Options Controlling C Dialect 29
3.5 Options Controlling C++ Dialect 35
3.6 Options Controlling Objective-C and Objective-C++ Dialects 45
3.7 Options to Control Diagnostic Messages Formatting 48
3.8 Options to Request or Suppress Warnings 49
3.9 Options for Debugging Your Program or GCC 72
3.10 Options That Control Optimization 92
3.11 Options Controlling the Preprocessor 143
3.12 Passing Options to the Assembler 154
3.13 Options for Linking 154
3.14 Options for Directory Search 158
3.15 Specifying subprocesses and the switches to pass to them 160
3.16 Specifying Target Machine and Compiler Version 167
3.17 Hardware Models and Configurations 167
3.17.1 Adapteva Epiphany Options 167
3.17.2 ARM Options 169
3.17.3 AVR Options 175
3.17.3.1 `EIND` and Devices with more than 128 Ki Bytes of Flash 178
3.17.3.2 Handling of the `RAMPD`, `RAMPX`, `RAMPY` and `RAMPZ` Special Function Registers 180
3.17.3.3 AVR Built-in Macros 180
3.17.4 Blackfin Options 182
3.17.5 C6X Options 185
3.17.6 CRIS Options 185
3.17.7 CR16 Options 187

3.17.8 Darwin Options ... 188
3.17.9 DEC Alpha Options ... 191
3.17.10 DEC Alpha/VMS Options ... 196
3.17.11 FR30 Options ... 196
3.17.12 FRV Options ... 196
3.17.13 GNU/Linux Options ... 200
3.17.14 H8/300 Options ... 201
3.17.15 HPPA Options ... 201
3.17.16 Intel 386 and AMD x86-64 Options ... 204
3.17.17 i386 and x86-64 Windows Options ... 218
3.17.18 IA-64 Options ... 218
3.17.19 IA-64/VMS Options ... 222
3.17.20 LM32 Options ... 222
3.17.21 M32C Options ... 223
3.17.22 M32R/D Options ... 223
3.17.23 M680x0 Options ... 225
3.17.24 MCore Options ... 230
3.17.25 MeP Options ... 231
3.17.26 MicroBlaze Options ... 232
3.17.27 MIPS Options ... 233
3.17.28 MMIX Options ... 245
3.17.29 MN10300 Options ... 246
3.17.30 PDP-11 Options ... 247
3.17.31 picoChip Options ... 248
3.17.32 PowerPC Options ... 248
3.17.33 RL78 Options ... 249
3.17.34 IBM RS/6000 and PowerPC Options ... 249
3.17.35 RX Options ... 263
3.17.36 S/390 and zSeries Options ... 265
3.17.37 Score Options ... 268
3.17.38 SH Options ... 269
3.17.39 Solaris 2 Options ... 273
3.17.40 SPARC Options ... 274
3.17.41 SPU Options ... 279
3.17.42 Options for System V ... 281
3.17.43 TILE-Gx Options ... 281
3.17.44 TILEPro Options ... 281
3.17.45 V850 Options ... 281
3.17.46 VAX Options ... 283
3.17.47 VxWorks Options ... 283
3.17.48 x86-64 Options ... 284
3.17.49 Xstormy16 Options ... 284
3.17.50 Xtensa Options ... 284
3.17.51 zSeries Options ... 285
3.18 Options for Code Generation Conventions ... 285
3.19 Environment Variables Affecting GCC ... 294
3.20 Using Precompiled Headers ... 297

4 C Implementation-defined behavior 299

4.1 Translation 299
4.2 Environment 299
4.3 Identifiers 299
4.4 Characters 300
4.5 Integers 300
4.6 Floating point 301
4.7 Arrays and pointers 302
4.8 Hints 303
4.9 Structures, unions, enumerations, and bit-fields 303
4.10 Qualifiers 304
4.11 Declarators 304
4.12 Statements 304
4.13 Preprocessing directives 304
4.14 Library functions 305
4.15 Architecture 305
4.16 Locale-specific behavior 305

5 C++ Implementation-defined behavior 307

5.1 Conditionally-supported behavior 307
5.2 Exception handling 307

6 Extensions to the C Language Family 309

6.1 Statements and Declarations in Expressions 309
6.2 Locally Declared Labels 310
6.3 Labels as Values 311
6.4 Nested Functions 312
6.5 Constructing Function Calls 314
6.6 Referring to a Type with `typeof` 316
6.7 Conditionals with Omitted Operands 317
6.8 128-bits integers 318
6.9 Double-Word Integers 318
6.10 Complex Numbers 318
6.11 Additional Floating Types 319
6.12 Half-Precision Floating Point 319
6.13 Decimal Floating Types 320
6.14 Hex Floats 320
6.15 Fixed-Point Types 321
6.16 Named Address Spaces 322
6.16.1 AVR Named Address Spaces 322
6.16.2 M32C Named Address Spaces 324
6.16.3 RL78 Named Address Spaces 324
6.16.4 SPU Named Address Spaces 324
6.17 Arrays of Length Zero 324
6.18 Structures With No Members 326
6.19 Arrays of Variable Length 326
6.20 Macros with a Variable Number of Arguments. 327

6.21 Slightly Looser Rules for Escaped Newlines ... 328
6.22 Non-Lvalue Arrays May Have Subscripts ... 328
6.23 Arithmetic on `void`- and Function-Pointers ... 328
6.24 Non-Constant Initializers ... 328
6.25 Compound Literals ... 328
6.26 Designated Initializers ... 330
6.27 Case Ranges ... 331
6.28 Cast to a Union Type ... 331
6.29 Mixed Declarations and Code ... 332
6.30 Declaring Attributes of Functions ... 332
6.31 Attribute Syntax ... 361
6.32 Prototypes and Old-Style Function Definitions ... 364
6.33 C++ Style Comments ... 365
6.34 Dollar Signs in Identifier Names ... 365
6.35 The Character ESC in Constants ... 365
6.36 Specifying Attributes of Variables ... 365
6.36.1 AVR Variable Attributes ... 370
6.36.2 Blackfin Variable Attributes ... 370
6.36.3 M32R/D Variable Attributes ... 371
6.36.4 MeP Variable Attributes ... 371
6.36.5 i386 Variable Attributes ... 372
6.36.6 PowerPC Variable Attributes ... 374
6.36.7 SPU Variable Attributes ... 374
6.36.8 Xstormy16 Variable Attributes ... 374
6.37 Specifying Attributes of Types ... 374
6.37.1 ARM Type Attributes ... 378
6.37.2 MeP Type Attributes ... 379
6.37.3 i386 Type Attributes ... 379
6.37.4 PowerPC Type Attributes ... 379
6.37.5 SPU Type Attributes ... 380
6.38 Inquiring on Alignment of Types or Variables ... 380
6.39 An Inline Function is As Fast As a Macro ... 380
6.40 When is a Volatile Object Accessed? ... 382
6.41 Assembler Instructions with C Expression Operands ... 383
6.41.1 Size of an `asm` ... 389
6.41.2 i386 floating point asm operands ... 389
6.42 Constraints for `asm` Operands ... 390
6.42.1 Simple Constraints ... 390
6.42.2 Multiple Alternative Constraints ... 393
6.42.3 Constraint Modifier Characters ... 393
6.42.4 Constraints for Particular Machines ... 394
6.43 Controlling Names Used in Assembler Code ... 417
6.44 Variables in Specified Registers ... 418
6.44.1 Defining Global Register Variables ... 418
6.44.2 Specifying Registers for Local Variables ... 420
6.45 Alternate Keywords ... 420
6.46 Incomplete `enum` Types ... 421
6.47 Function Names as Strings ... 421

6.48 Getting the Return or Frame Address of a Function 422
6.49 Using vector instructions through built-in functions 423
6.50 Offsetof 425
6.51 Legacy __sync built-in functions for atomic memory access ... 425
6.52 Built-in functions for memory model aware atomic operations 427
6.53 Object Size Checking Builtins 431
6.54 Other built-in functions provided by GCC 433
6.55 Built-in Functions Specific to Particular Target Machines 442
6.55.1 Alpha Built-in Functions 442
6.55.2 ARM iWMMXt Built-in Functions 443
6.55.3 ARM NEON Intrinsics 446
6.55.3.1 Addition 446
6.55.3.2 Multiplication 450
6.55.3.3 Multiply-accumulate 451
6.55.3.4 Multiply-subtract 452
6.55.3.5 Subtraction 453
6.55.3.6 Comparison (equal-to) 457
6.55.3.7 Comparison (greater-than-or-equal-to) 457
6.55.3.8 Comparison (less-than-or-equal-to) 458
6.55.3.9 Comparison (greater-than) 459
6.55.3.10 Comparison (less-than) 460
6.55.3.11 Comparison (absolute greater-than-or-equal-to) ... 460
6.55.3.12 Comparison (absolute less-than-or-equal-to) 460
6.55.3.13 Comparison (absolute greater-than) 461
6.55.3.14 Comparison (absolute less-than) 461
6.55.3.15 Test bits 461
6.55.3.16 Absolute difference 462
6.55.3.17 Absolute difference and accumulate 463
6.55.3.18 Maximum 463
6.55.3.19 Minimum 464
6.55.3.20 Pairwise add 465
6.55.3.21 Pairwise add, single_opcode widen and accumulate 466
6.55.3.22 Folding maximum 466
6.55.3.23 Folding minimum 467
6.55.3.24 Reciprocal step 467
6.55.3.25 Vector shift left 467
6.55.3.26 Vector shift left by constant 470
6.55.3.27 Vector shift right by constant 473
6.55.3.28 Vector shift right by constant and accumulate 476
6.55.3.29 Vector shift right and insert 477
6.55.3.30 Vector shift left and insert 478
6.55.3.31 Absolute value 479
6.55.3.32 Negation 480
6.55.3.33 Bitwise not 481
6.55.3.34 Count leading sign bits 481
6.55.3.35 Count leading zeros 482

6.55.3.36 Count number of set bits ... 482
6.55.3.37 Reciprocal estimate ... 483
6.55.3.38 Reciprocal square-root estimate ... 483
6.55.3.39 Get lanes from a vector ... 483
6.55.3.40 Set lanes in a vector ... 484
6.55.3.41 Create vector from literal bit pattern ... 485
6.55.3.42 Set all lanes to the same value ... 486
6.55.3.43 Combining vectors ... 488
6.55.3.44 Splitting vectors ... 489
6.55.3.45 Conversions ... 490
6.55.3.46 Move, single_opcode narrowing ... 490
6.55.3.47 Move, single_opcode long ... 491
6.55.3.48 Table lookup ... 491
6.55.3.49 Extended table lookup ... 492
6.55.3.50 Multiply, lane ... 493
6.55.3.51 Long multiply, lane ... 493
6.55.3.52 Saturating doubling long multiply, lane ... 493
6.55.3.53 Saturating doubling multiply high, lane ... 494
6.55.3.54 Multiply-accumulate, lane ... 494
6.55.3.55 Multiply-subtract, lane ... 495
6.55.3.56 Vector multiply by scalar ... 496
6.55.3.57 Vector long multiply by scalar ... 496
6.55.3.58 Vector saturating doubling long multiply by scalar ... 496
6.55.3.59 Vector saturating doubling multiply high by scalar ... 496
6.55.3.60 Vector multiply-accumulate by scalar ... 497
6.55.3.61 Vector multiply-subtract by scalar ... 498
6.55.3.62 Vector extract ... 498
6.55.3.63 Reverse elements ... 500
6.55.3.64 Bit selection ... 501
6.55.3.65 Transpose elements ... 503
6.55.3.66 Zip elements ... 504
6.55.3.67 Unzip elements ... 505
6.55.3.68 Element/structure loads, VLD1 variants ... 505
6.55.3.69 Element/structure stores, VST1 variants ... 509
6.55.3.70 Element/structure loads, VLD2 variants ... 511
6.55.3.71 Element/structure stores, VST2 variants ... 513
6.55.3.72 Element/structure loads, VLD3 variants ... 515
6.55.3.73 Element/structure stores, VST3 variants ... 517
6.55.3.74 Element/structure loads, VLD4 variants ... 519
6.55.3.75 Element/structure stores, VST4 variants ... 521
6.55.3.76 Logical operations (AND) ... 523
6.55.3.77 Logical operations (OR) ... 523
6.55.3.78 Logical operations (exclusive OR) ... 524
6.55.3.79 Logical operations (AND-NOT) ... 525
6.55.3.80 Logical operations (OR-NOT) ... 526
6.55.3.81 Reinterpret casts ... 526

6.55.4 AVR Built-in Functions ... 532
6.55.5 Blackfin Built-in Functions ... 533
6.55.6 FR-V Built-in Functions ... 533
6.55.6.1 Argument Types ... 534
6.55.6.2 Directly-mapped Integer Functions ... 534
6.55.6.3 Directly-mapped Media Functions ... 534
6.55.6.4 Raw read/write Functions ... 536
6.55.6.5 Other Built-in Functions ... 537
6.55.7 X86 Built-in Functions ... 537
6.55.8 MIPS DSP Built-in Functions ... 557
6.55.9 MIPS Paired-Single Support ... 561
6.55.10 MIPS Loongson Built-in Functions ... 562
6.55.10.1 Paired-Single Arithmetic ... 564
6.55.10.2 Paired-Single Built-in Functions ... 564
6.55.10.3 MIPS-3D Built-in Functions ... 565
6.55.11 picoChip Built-in Functions ... 568
6.55.12 Other MIPS Built-in Functions ... 568
6.55.13 PowerPC AltiVec Built-in Functions ... 568
6.55.14 RX Built-in Functions ... 603
6.55.15 SPARC VIS Built-in Functions ... 605
6.55.16 SPU Built-in Functions ... 607
6.55.17 TI C6X Built-in Functions ... 608
6.55.18 TILE-Gx Built-in Functions ... 608
6.55.19 TILEPro Built-in Functions ... 609
6.56 Format Checks Specific to Particular Target Machines ... 609
6.56.1 Solaris Format Checks ... 609
6.56.2 Darwin Format Checks ... 609
6.57 Pragmas Accepted by GCC ... 610
6.57.1 ARM Pragmas ... 610
6.57.2 M32C Pragmas ... 610
6.57.3 MeP Pragmas ... 610
6.57.4 RS/6000 and PowerPC Pragmas ... 611
6.57.5 Darwin Pragmas ... 611
6.57.6 Solaris Pragmas ... 612
6.57.7 Symbol-Renaming Pragmas ... 612
6.57.8 Structure-Packing Pragmas ... 613
6.57.9 Weak Pragmas ... 614
6.57.10 Diagnostic Pragmas ... 614
6.57.11 Visibility Pragmas ... 615
6.57.12 Push/Pop Macro Pragmas ... 615
6.57.13 Function Specific Option Pragmas ... 616
6.58 Unnamed struct/union fields within structs/unions ... 617
6.59 Thread-Local Storage ... 618
6.59.1 ISO/IEC 9899:1999 Edits for Thread-Local Storage ... 618
6.59.2 ISO/IEC 14882:1998 Edits for Thread-Local Storage ... 619
6.60 Binary constants using the '0b' prefix ... 620

7 Extensions to the C++ Language 623

7.1 When is a Volatile C++ Object Accessed? 623
7.2 Restricting Pointer Aliasing 623
7.3 Vague Linkage 624
7.4 #pragma interface and implementation 625
7.5 Where's the Template? 626
7.6 Extracting the function pointer from a bound pointer to member function 629
7.7 C++-Specific Variable, Function, and Type Attributes 629
7.8 Namespace Association 630
7.9 Type Traits 630
7.10 Java Exceptions 632
7.11 Deprecated Features 633
7.12 Backwards Compatibility 634

8 GNU Objective-C features 635

8.1 GNU Objective-C runtime API 635
8.1.1 Modern GNU Objective-C runtime API 635
8.1.2 Traditional GNU Objective-C runtime API 636
8.2 `+load`: Executing code before main 636
8.2.1 What you can and what you cannot do in `+load` 637
8.3 Type encoding 638
8.3.1 Legacy type encoding 640
8.3.2 `@`encode 640
8.3.3 Method signatures 641
8.4 Garbage Collection 641
8.5 Constant string objects 642
8.6 compatibility_alias 643
8.7 Exceptions 643
8.8 Synchronization 645
8.9 Fast enumeration 645
8.9.1 Using fast enumeration 645
8.9.2 c99-like fast enumeration syntax 645
8.9.3 Fast enumeration details 646
8.9.4 Fast enumeration protocol 647
8.10 Messaging with the GNU Objective-C runtime 648
8.10.1 Dynamically registering methods 648
8.10.2 Forwarding hook 648

9 Binary Compatibility 651

10 `gcov`—a Test Coverage Program 655

10.1 Introduction to `gcov` 655
10.2 Invoking `gcov` 655
10.3 Using `gcov` with GCC Optimization 661
10.4 Brief description of `gcov` data files 662
10.5 Data file relocation to support cross-profiling 662

11 Known Causes of Trouble with GCC 663
11.1 Actual Bugs We Haven't Fixed Yet 663
11.2 Cross-Compiler Problems 663
11.3 Interoperation 663
11.4 Incompatibilities of GCC 665
11.5 Fixed Header Files 668
11.6 Standard Libraries 668
11.7 Disappointments and Misunderstandings 669
11.8 Common Misunderstandings with GNU C++ 670
11.8.1 Declare *and* Define Static Members 670
11.8.2 Name lookup, templates, and accessing members of base classes 671
11.8.3 Temporaries May Vanish Before You Expect 672
11.8.4 Implicit Copy-Assignment for Virtual Bases 673
11.9 Certain Changes We Don't Want to Make 674
11.10 Warning Messages and Error Messages 677

12 Reporting Bugs 679
12.1 Have You Found a Bug? 679
12.2 How and where to Report Bugs 679

13 How To Get Help with GCC 681

14 Contributing to GCC Development 683

Funding Free Software 685

The GNU Project and GNU/Linux 687

GNU General Public License 689

GNU Free Documentation License 701
ADDENDUM: How to use this License for your documents 708

Contributors to GCC 709

Option Index 725

Keyword Index 743

Introduction

This manual documents how to use the GNU compilers, as well as their features and incompatibilities, and how to report bugs. It corresponds to the compilers (GCC) version 4.7.2. The internals of the GNU compilers, including how to port them to new targets and some information about how to write front ends for new languages, are documented in a separate manual. See Section "Introduction" in *GNU Compiler Collection (GCC) Internals*.

1 Programming Languages Supported by GCC

GCC stands for "GNU Compiler Collection". GCC is an integrated distribution of compilers for several major programming languages. These languages currently include C, C++, Objective-C, Objective-C++, Java, Fortran, Ada, and Go.

The abbreviation *GCC* has multiple meanings in common use. The current official meaning is "GNU Compiler Collection", which refers generically to the complete suite of tools. The name historically stood for "GNU C Compiler", and this usage is still common when the emphasis is on compiling C programs. Finally, the name is also used when speaking of the *language-independent* component of GCC: code shared among the compilers for all supported languages.

The language-independent component of GCC includes the majority of the optimizers, as well as the "back ends" that generate machine code for various processors.

The part of a compiler that is specific to a particular language is called the "front end". In addition to the front ends that are integrated components of GCC, there are several other front ends that are maintained separately. These support languages such as Pascal, Mercury, and COBOL. To use these, they must be built together with GCC proper.

Most of the compilers for languages other than C have their own names. The C++ compiler is G++, the Ada compiler is GNAT, and so on. When we talk about compiling one of those languages, we might refer to that compiler by its own name, or as GCC. Either is correct.

Historically, compilers for many languages, including C++ and Fortran, have been implemented as "preprocessors" which emit another high level language such as C. None of the compilers included in GCC are implemented this way; they all generate machine code directly. This sort of preprocessor should not be confused with the *C preprocessor*, which is an integral feature of the C, C++, Objective-C and Objective-C++ languages.

2 Language Standards Supported by GCC

For each language compiled by GCC for which there is a standard, GCC attempts to follow one or more versions of that standard, possibly with some exceptions, and possibly with some extensions.

2.1 C language

GCC supports three versions of the C standard, although support for the most recent version is not yet complete.

The original ANSI C standard (X3.159-1989) was ratified in 1989 and published in 1990. This standard was ratified as an ISO standard (ISO/IEC 9899:1990) later in 1990. There were no technical differences between these publications, although the sections of the ANSI standard were renumbered and became clauses in the ISO standard. This standard, in both its forms, is commonly known as *C89*, or occasionally as *C90*, from the dates of ratification. The ANSI standard, but not the ISO standard, also came with a Rationale document. To select this standard in GCC, use one of the options '`-ansi`', '`-std=c90`' or '`-std=iso9899:1990`'; to obtain all the diagnostics required by the standard, you should also specify '`-pedantic`' (or '`-pedantic-errors`' if you want them to be errors rather than warnings). See Section 3.4 [Options Controlling C Dialect], page 29.

Errors in the 1990 ISO C standard were corrected in two Technical Corrigenda published in 1994 and 1996. GCC does not support the uncorrected version.

An amendment to the 1990 standard was published in 1995. This amendment added digraphs and `__STDC_VERSION__` to the language, but otherwise concerned the library. This amendment is commonly known as *AMD1*; the amended standard is sometimes known as *C94* or *C95*. To select this standard in GCC, use the option '`-std=iso9899:199409`' (with, as for other standard versions, '`-pedantic`' to receive all required diagnostics).

A new edition of the ISO C standard was published in 1999 as ISO/IEC 9899:1999, and is commonly known as *C99*. GCC has incomplete support for this standard version; see `http://gcc.gnu.org/gcc-4.7/c99status.html` for details. To select this standard, use '`-std=c99`' or '`-std=iso9899:1999`'. (While in development, drafts of this standard version were referred to as *C9X*.)

Errors in the 1999 ISO C standard were corrected in three Technical Corrigenda published in 2001, 2004 and 2007. GCC does not support the uncorrected version.

A fourth version of the C standard, known as *C11*, was published in 2011 as ISO/IEC 9899:2011. GCC has limited incomplete support for parts of this standard, enabled with '`-std=c11`' or '`-std=iso9899:2011`'. (While in development, drafts of this standard version were referred to as *C1X*.)

By default, GCC provides some extensions to the C language that on rare occasions conflict with the C standard. See Chapter 6 [Extensions to the C Language Family], page 309. Use of the '`-std`' options listed above will disable these extensions where they conflict with the C standard version selected. You may also select an extended version of the C language explicitly with '`-std=gnu90`' (for C90 with GNU extensions), '`-std=gnu99`' (for C99 with GNU extensions) or '`-std=gnu11`' (for C11 with GNU extensions). The default, if no C language dialect options are given, is '`-std=gnu90`'; this will change to '`-std=gnu99`' or '`-std=gnu11`' in some future release when the C99 or C11 support is complete. Some

features that are part of the C99 standard are accepted as extensions in C90 mode, and some features that are part of the C11 standard are accepted as extensions in C90 and C99 modes.

The ISO C standard defines (in clause 4) two classes of conforming implementation. A *conforming hosted implementation* supports the whole standard including all the library facilities; a *conforming freestanding implementation* is only required to provide certain library facilities: those in `<float.h>`, `<limits.h>`, `<stdarg.h>`, and `<stddef.h>`; since AMD1, also those in `<iso646.h>`; since C99, also those in `<stdbool.h>` and `<stdint.h>`; and since C11, also those in `<stdalign.h>` and `<stdnoreturn.h>`. In addition, complex types, added in C99, are not required for freestanding implementations. The standard also defines two environments for programs, a *freestanding environment*, required of all implementations and which may not have library facilities beyond those required of freestanding implementations, where the handling of program startup and termination are implementation-defined, and a *hosted environment*, which is not required, in which all the library facilities are provided and startup is through a function `int main (void)` or `int main (int, char *[])`. An OS kernel would be a freestanding environment; a program using the facilities of an operating system would normally be in a hosted implementation.

GCC aims towards being usable as a conforming freestanding implementation, or as the compiler for a conforming hosted implementation. By default, it will act as the compiler for a hosted implementation, defining `__STDC_HOSTED__` as `1` and presuming that when the names of ISO C functions are used, they have the semantics defined in the standard. To make it act as a conforming freestanding implementation for a freestanding environment, use the option '`-ffreestanding`'; it will then define `__STDC_HOSTED__` to `0` and not make assumptions about the meanings of function names from the standard library, with exceptions noted below. To build an OS kernel, you may well still need to make your own arrangements for linking and startup. See Section 3.4 [Options Controlling C Dialect], page 29.

GCC does not provide the library facilities required only of hosted implementations, nor yet all the facilities required by C99 of freestanding implementations; to use the facilities of a hosted environment, you will need to find them elsewhere (for example, in the GNU C library). See Section 11.6 [Standard Libraries], page 668.

Most of the compiler support routines used by GCC are present in '`libgcc`', but there are a few exceptions. GCC requires the freestanding environment provide `memcpy`, `memmove`, `memset` and `memcmp`. Finally, if `__builtin_trap` is used, and the target does not implement the `trap` pattern, then GCC will emit a call to `abort`.

For references to Technical Corrigenda, Rationale documents and information concerning the history of C that is available online, see `http://gcc.gnu.org/readings.html`

2.2 C++ language

GCC supports the original ISO C++ standard (1998) and contains experimental support for the second ISO C++ standard (2011).

The original ISO C++ standard was published as the ISO standard (ISO/IEC 14882:1998) and amended by a Technical Corrigenda published in 2003 (ISO/IEC 14882:2003). These standards are referred to as C++98 and C++03, respectively. GCC implements the majority of C++98 (`export` is a notable exception) and most of the changes in C++03. To select this standard in GCC, use one of the options '`-ansi`', '`-std=c++98`', or '`-std=c++03`'; to

obtain all the diagnostics required by the standard, you should also specify '`-pedantic`' (or '`-pedantic-errors`' if you want them to be errors rather than warnings).

A revised ISO C++ standard was published in 2011 as ISO/IEC 14882:2011, and is referred to as C++11; before its publication it was commonly referred to as C++0x. C++11 contains several changes to the C++ language, most of which have been implemented in an experimental C++11 mode in GCC. For information regarding the C++11 features available in the experimental C++11 mode, see `http://gcc.gnu.org/projects/cxx0x.html`. To select this standard in GCC, use the option '`-std=c++11`'; to obtain all the diagnostics required by the standard, you should also specify '`-pedantic`' (or '`-pedantic-errors`' if you want them to be errors rather than warnings).

More information about the C++ standards is available on the ISO C++ committee's web site at `http://www.open-std.org/jtc1/sc22/wg21/`.

By default, GCC provides some extensions to the C++ language; See Section 3.5 [C++ Dialect Options], page 35. Use of the '`-std`' option listed above will disable these extensions. You may also select an extended version of the C++ language explicitly with '`-std=gnu++98`' (for C++98 with GNU extensions) or '`-std=gnu++11`' (for C++11 with GNU extensions). The default, if no C++ language dialect options are given, is '`-std=gnu++98`'.

2.3 Objective-C and Objective-C++ languages

GCC supports "traditional" Objective-C (also known as "Objective-C 1.0") and contains support for the Objective-C exception and synchronization syntax. It has also support for a number of "Objective-C 2.0" language extensions, including properties, fast enumeration (only for Objective-C), method attributes and the `@`optional and `@`required keywords in protocols. GCC supports Objective-C++ and features available in Objective-C are also available in Objective-C++.

GCC by default uses the GNU Objective-C runtime library, which is part of GCC and is not the same as the Apple/NeXT Objective-C runtime library used on Apple systems. There are a number of differences documented in this manual. The options '`-fgnu-runtime`' and '`-fnext-runtime`' allow you to switch between producing output that works with the GNU Objective-C runtime library and output that works with the Apple/NeXT Objective-C runtime library.

There is no formal written standard for Objective-C or Objective-C++. The authoritative manual on traditional Objective-C (1.0) is "Object-Oriented Programming and the Objective-C Language", available at a number of web sites:

- `http://www.gnustep.org/resources/documentation/ObjectivCBook.pdf` is the original NeXTstep document;
- `http://objc.toodarkpark.net` is the same document in another format;
- `http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/ObjectiveC/` has an updated version but make sure you search for "Object Oriented Programming and the Objective-C Programming Language 1.0", not documentation on the newer "Objective-C 2.0" language

The Objective-C exception and synchronization syntax (that is, the keywords `@`try, `@`throw, `@`catch, `@`finally and `@`synchronized) is supported by GCC and is enabled with the

option '`-fobjc-exceptions`'. The syntax is briefly documented in this manual and in the Objective-C 2.0 manuals from Apple.

The Objective-C 2.0 language extensions and features are automatically enabled; they include properties (via the `@`property, `@`synthesize and `@`dynamic keywords), fast enumeration (not available in Objective-C++), attributes for methods (such as deprecated, noreturn, sentinel, format), the unused attribute for method arguments, the `@`package keyword for instance variables and the `@`optional and `@`required keywords in protocols. You can disable all these Objective-C 2.0 language extensions with the option '`-fobjc-std=objc1`', which causes the compiler to recognize the same Objective-C language syntax recognized by GCC 4.0, and to produce an error if one of the new features is used.

GCC has currently no support for non-fragile instance variables.

The authoritative manual on Objective-C 2.0 is available from Apple:

- `http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/ObjectiveC/`

For more information concerning the history of Objective-C that is available online, see `http://gcc.gnu.org/readings.html`

2.4 Go language

The Go language continues to evolve as of this writing; see the current language specifications (`http://golang.org/doc/go_spec.html`). At present there are no specific versions of Go, and there is no way to describe the language supported by GCC in terms of a specific version. In general GCC tracks the evolving specification closely, and any given release will support the language as of the date that the release was frozen.

2.5 References for other languages

See Section "About This Guide" in *GNAT Reference Manual*, for information on standard conformance and compatibility of the Ada compiler.

See Section "Standards" in *The GNU Fortran Compiler*, for details of standards supported by GNU Fortran.

See Section "Compatibility with the Java Platform" in *GNU gcj*, for details of compatibility between `gcj` and the Java Platform.

3 GCC Command Options

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

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

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

See Section 3.3 [Compiling C++ Programs], page 29, for a summary of special options for compiling C++ programs.

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.

See [Option Index], page 725, for an index to GCC's options.

3.1 Option Summary

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

Overall Options

See Section 3.2 [Options Controlling the Kind of Output], page 23.

```
-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

See Section 3.4 [Options Controlling C Dialect], page 29.

```
-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

See Section 3.5 [Options Controlling C++ Dialect], page 35.

```
-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

See Section 3.6 [Options Controlling Objective-C and Objective-C++ Dialects], page 45.

```
-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

See Section 3.7 [Options to Control Diagnostic Messages Formatting], page 48.

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

Warning Options

See Section 3.8 [Options to Request or Suppress Warnings], page 49.

```
-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

See Section 3.9 [Options for Debugging Your Program or GCC], page 72.

```
-dletters -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 -glevel -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

See Section 3.10 [Options that Control Optimization], page 92.

-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

See Section 3.11 [Options Controlling the Preprocessor], page 143.

```
-Aquestion=answer
-A-question[=answer]
-C -dD -dI -dM -dN
-Dmacro[=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 -Umacro
-Wp,option -Xpreprocessor option
```

Assembler Option

See Section 3.12 [Passing Options to the Assembler], page 154.

```
-Wa,option -Xassembler option
```

Linker Options

See Section 3.13 [Options for Linking], page 154.

```
object-file-name -llibrary
-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

See Section 3.14 [Options for Directory Search], page 158.

```
-Bprefix -Idir -iplugindir=dir
-iquotedir -Ldir -specs=file -I-
--sysroot=dir
```

Machine Dependent Options

See Section 3.17 [Hardware Models and Configurations], page 167.

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
-Gnum -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

See Section 3.18 [Options for Code Generation Conventions], page 285.

`-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`

3.2 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*`|`[`^`]*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 '`^`' 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,^joined,^undocumented
```

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. See Section "Generating Ada Bindings for C and C++ headers" in *GNAT User's Guide*, which provides detailed documentation on this feature.

`-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.

3.3 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. See Section 3.4 [Options Controlling C Dialect], page 29, for explanations of options for languages related to C. See Section 3.5 [Options Controlling C++ Dialect], page 35, for explanations of options that are meaningful only for C++ programs.

3.4 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`'. See Section 3.8 [Warning Options], page 49.

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. See Section 6.54 [Other built-in functions provided by GCC], page 433, for details of the functions affected.

`-std=` Determine the language standard. See Chapter 2 [Language Standards Supported by GCC], page 5, for details of these standard versions. 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. See Section 6.39 [An Inline Function is As Fast As a Macro], page 380. 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 (see Section 6.30 [Function Attributes], page 332).

  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. See Section "Common Predefined Macros" in *The C Preprocessor*.

`-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. See Section 6.54 [Other built-in functions provided by GCC], page 433, for details of the functions affected, including those which are not built-in functions when '`-ansi`' or '`-std`' options for strict ISO C conformance are used because they do not have an ISO standard meaning.

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`'.

See Chapter 2 [Language Standards Supported by GCC], page 5, for details of freestanding and hosted environments.

`-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, See Section "The GNU Transactional Memory Library" in *GNU Transactional Memory Library*.

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. See Section 6.58 [Unnamed struct/union fields within structs/unions], page 617, for details.

`-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. See Section 6.58 [Unnamed struct/union fields within structs/unions], page 617, for details. 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.

3.5 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. See Section 7.5 [Template Instantiation], page 626, for more information.

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

`-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`'. See Section 7.5 [Template Instantiation], page 626, for more information.

`-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. See Section 7.5 [Template Instantiation], page 626.

`-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. See Section 3.10 [Options That Control Optimization], page 92. 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 =`'.

3.6 Options Controlling Objective-C and Objective-C++ Dialects

(NOTE: This manual does not describe the Objective-C and Objective-C++ languages themselves. See Chapter 2 [Language Standards Supported by GCC], page 5, for references.)

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.

3.7 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.

3.8 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 Section 3.5 [C++ Dialect Options], page 35 and Section 3.6 [Objective-C and Objective-C++ Dialect Options], page 45.

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. See Section 6.45 [Alternate Keywords], page 420.

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 Section 3.5 [C++ Dialect Options], page 35 and Section 3.6 [Objective-C and Objective-C++ Dialect Options], page 45.

'`-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 (see Section 6.30 [Function Attributes], page 332), 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). See Section 3.4 [Options Controlling C Dialect], page 29.

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 (see Section 6.36 [Variable Attributes], page 365).

  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 (see Section 6.36 [Variable Attributes], page 365).

  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 (see Section 6.36 [Variable Attributes], page 365).

`-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 (see Section 6.36 [Variable Attributes], page 365).

`-Wno-unused-result`
: Do not warn if a caller of a function marked with attribute `warn_unused_result` (see Section 6.30 [Function Attributes], page 332) 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 (see Section 6.36 [Variable Attributes], page 365).

`-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`. See Section 6.30 [Function Attributes], page 332.

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. See Section 6.29 [Mixed Declarations], page 332.

`-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. See Section 7.11 [Deprecated Features], page 633.

`-Wno-deprecated-declarations`
: Do not warn about uses of functions (see Section 6.30 [Function Attributes], page 332), variables (see Section 6.36 [Variable Attributes], page 365), and types (see Section 6.37 [Type Attributes], page 374) 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 (see Section 6.26 [Designated Initializers], page 330).

  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 misaligned 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 (see Section 3.20 [Precompiled Headers], page 297) 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.

3.9 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`'). See Section 10.5 [Cross-profiling], page 662.

`--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`' (see Section 3.10 [Options that Control Optimization], page 92).
- 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 (see Chapter 10 [`gcov`—a Test Coverage Program], page 655) 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 ‘`lib`*suffix*’ 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-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 '`/`'. See Section 3.19 [Environment Variables], page 294.

`-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.) See Section 3.15 [Spec Files], page 160.

`-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.

3.10 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. See Section 3.2 [Overall Options], page 23, for examples.

`-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 '`-ffflag`' 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. See Section "Register Usage" in *GNU Compiler Collection (GCC) Internals*.

  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 (See Section 6.30 [Declaring Attributes of Functions], page 332.)

`-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. See Section 4.9 [Structures unions enumerations and bit-fields implementation], page 303. 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 fol-

lowing 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`' (see Section 3.9 [Options for Debugging Your Program or `gcc`], page 72), 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 $O(N^2)$ in the number of edges incoming to each block. Increasing values mean more aggressive optimization, making the compilation time increase with probably small improvement in executable size.

  `min-crossjump-insns`
  : The minimum number of instructions that must be matched at the end of two blocks before 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 $O(N^2)$ behavior in a number of passes, GCC factors computed gotos early in the compilation process, and unfactors them as late as possible. Only computed jumps at the end of a basic blocks with no more than max-goto-duplication-insns are unfactored. 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 dependencies is expensive for very large loops. This parameter limits the number of data references in loops that are considered for data dependence analysis. These large loops 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.

3.11 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` (see Section 3.19 [Environment Variables], page 294). 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 (see [-MF], page 146), 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 (see Section 3.20 [Precompiled Headers], page 297), 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 (see Section 3.20 [Precompiled Headers], page 297) 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’:‘/file/path’;‘F’:‘/includer/path’;‘L’:line_num;‘C’:col_num;‘S’:system_header_p;‘M’:map_
```

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`'. See Section 3.9 [Debugging Options], page 72.

'`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.

3.12 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.

3.13 Options for Linking

These options come into play when the compiler links object files into an executable output file. They are meaningless if the compiler is not doing a link step.

`object-file-name`
: A file name that does not end in a special recognized suffix is considered to name an object file or library. (Object files are distinguished from libraries by the linker according to the file contents.) If linking is done, these object files are used as input to the linker.

`-c`
`-S`
`-E`
: If any of these options is used, then the linker is not run, and object file names should not be used as arguments. See Section 3.2 [Overall Options], page 23.

`-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 '`lib`*`library`*`.a`'. The linker then uses this file as if it had been specified precisely by name.

  The directories searched include several standard system directories plus any that you specify with '`-L`'.

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

`-lobjc` You need this special case of the '`-l`' option in order to link an Objective-C 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. (See Section "Interfacing to GCC Output" in *GNU Compiler Collection (GCC) Internals*, for more discussion of '`libgcc.a`'.) In most cases, you need '`libgcc.a`' even when you want to avoid other standard libraries. In other words, when you specify '`-nostdlib`' or '`-nodefaultlibs`' you should usually specify '`-lgcc`' as well. This ensures that you have no unresolved references to internal GCC library subroutines. (For example, '`__main`', used to ensure C++ constructors will be called; see Section "`collect2`" in *GNU Compiler Collection (GCC) Internals*.)

`-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`'.

[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.

`-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.

3.14 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*`/`' (see Section 3.16 [Target Options], page 167).

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`. See Section 3.19 [Environment Variables], page 294.

As a special kludge, if the path provided by '`-B`' is '`[dir/]stage`*N*`/`', 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.

3.15 Specifying subprocesses and the switches to pass to them

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

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

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

`%include <`*file*`>`
: Search for *file* and insert its text at the current point in the specs file.

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

`%rename` *old_name new_name*
: Rename the spec string *old_name* to *new_name*.

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

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

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

This says that any input file whose name ends in '`.ZZ`' should be passed to the program '`z-compile`', which should be invoked with the command-line switch '`-input`' and with the result of performing the '`%i`' substitution. (See below.)

As an alternative to providing a spec string, the text that follows a suffix directive can be one of the following:

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

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

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

`#name`
: This causes an error messages saying:

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

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

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

```
asm          Options to pass to the assembler
asm_final    Options to pass to the assembler post-processor
cpp          Options to pass to the C preprocessor
cc1          Options to pass to the C compiler
cc1plus      Options to pass to the C++ compiler
endfile      Object files to include at the end of the link
link         Options to pass to the linker
lib          Libraries to include on the command line to the linker
libgcc       Decides which GCC support library to pass to the linker
linker       Sets the name of the linker
predefines   Defines to be passed to the C preprocessor
signed_char  Defines to pass to CPP to say whether char is signed
             by default
startfile    Object files to include at the start of the link
```

Here is a small example of a spec file:

```
%rename lib                 old_lib

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

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

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

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

`%%`
: Substitute one '`%`' into the program name or argument.

`%i`
: Substitute the name of the input file being processed.

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

`%B` This is the same as '`%b`', but include the file suffix (text after the last period).

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

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

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

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

`%j`*suffix* Substitutes the name of the `HOST_BIT_BUCKET`, if any, and if it is writable, and if save-temps is off; otherwise, substitute the name of a temporary file, just like '`%u`'. This temporary file is not meant for communication between processes, but rather as a junk disposal mechanism.

`%|`*suffix*

`%m`*suffix* Like '`%g`', except if '`-pipe`' is in effect. In that case '`%|`' substitutes a single dash and '`%m`' substitutes nothing at all. These are the two most common ways to instruct a program that it should read from standard input or write to standard output. If you need something more elaborate you can use an '`%{pipe:X}`' construct: see for example '`f/lang-specs.h`'.

`%.`*SUFFIX* Substitutes *.SUFFIX* for the suffixes of a matched switch's args when it is subsequently output with '`%*`'. *SUFFIX* is terminated by the next space or %.

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

`%o` Substitutes the names of all the output files, with spaces automatically placed around them. You should write spaces around the '`%o`' as well or the results are

undefined. '%o' is for use in the specs for running the linker. Input files whose names have no recognized suffix are not compiled at all, but they are included among the output files, so they will be linked.

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

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

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

%I Substitute any of '-iprefix' (made from GCC_EXEC_PREFIX), '-isysroot' (made from TARGET_SYSTEM_ROOT), '-isystem' (made from COMPILER_PATH and '-B' options) and '-imultilib' as necessary.

%s Current argument is the name of a library or startup file of some sort. Search for that file in a standard list of directories and substitute the full name found. The current working directory is included in the list of directories scanned.

%T Current argument is the name of a linker script. Search for that file in the current list of directories to scan for libraries. If the file is located insert a '--script' option into the command line followed by the full path name found. If the file is not found then generate an error message. Note: the current working directory is not searched.

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

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

%x{*option*}
Accumulate an option for '%X'.

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

%Y Output the accumulated assembler options specified by '-Wa'.

%Z Output the accumulated preprocessor options specified by '-Wp'.

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

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

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

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

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

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

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

`%E` Process the `endfile` spec. This is a spec string that specifies the last object files that will be passed to the linker.

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

`%1` Process the `cc1` spec. This is used to construct the options to be passed to the actual C compiler ('`cc1`').

`%2` Process the `cc1plus` spec. This is used to construct the options to be passed to the actual C++ compiler ('`cc1plus`').

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

`%<S` Remove all occurrences of `-S` from the command line. Note—this command is position dependent. '`%`' commands in the spec string before this one will see `-S`, '`%`' commands in the spec string after this one will not.

`%:`*function*`(`*args*`)`
Call the named function *function*, passing it *args*. *args* is first processed as a nested spec string, then split into an argument vector in the usual fashion. The function returns a string which is processed as if it had appeared literally as part of the current spec.

The following built-in spec functions are provided:

`getenv` The `getenv` spec function takes two arguments: an environment variable name and a string. If the environment variable is not defined, a fatal error is issued. Otherwise, the return value is the value of the environment variable concatenated with the string. For example, if `TOPDIR` is defined as '`/path/to/top`', then:

```
%:getenv(TOPDIR /include)
```

expands to '`/path/to/top/include`'.

`if-exists`
The `if-exists` spec function takes one argument, an absolute pathname to a file. If the file exists, `if-exists` returns the pathname. Here is a small example of its usage:

```
*startfile:
crt0%O%s %:if-exists(crti%O%s) crtbegin%O%s
```

`if-exists-else`
: The `if-exists-else` spec function is similar to the `if-exists` spec function, except that it takes two arguments. The first argument is an absolute pathname to a file. If the file exists, `if-exists-else` returns the pathname. If it does not exist, it returns the second argument. This way, `if-exists-else` can be used to select one file or another, based on the existence of the first. Here is a small example of its usage:

```
*startfile:
crt0%O%s %:if-exists(crti%O%s) \
%:if-exists-else(crtbeginT%O%s crtbegin%O%s)
```

`replace-outfile`
: The `replace-outfile` spec function takes two arguments. It looks for the first argument in the outfiles array and replaces it with the second argument. Here is a small example of its usage:

```
%{fgnu-runtime:%:replace-outfile(-lobjc -lobjc-gnu)}
```

`remove-outfile`
: The `remove-outfile` spec function takes one argument. It looks for the first argument in the outfiles array and removes it. Here is a small example its usage:

```
%:remove-outfile(-lm)
```

`pass-through-libs`
: The `pass-through-libs` spec function takes any number of arguments. It finds any '`-l`' options and any non-options ending in `".a"` (which it assumes are the names of linker input library archive files) and returns a result containing all the found arguments each prepended by '`-plugin-opt=-pass-through=`' and joined by spaces. This list is intended to be passed to the LTO linker plugin.

```
%:pass-through-libs(%G %L %G)
```

`print-asm-header`
: The `print-asm-header` function takes no arguments and simply prints a banner like:

```
Assembler options
=================

Use "-Wa,OPTION" to pass "OPTION" to the assembler.
```

  It is used to separate compiler options from assembler options in the '`--target-help`' output.

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

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

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

`%{S*&T*}` Like %{S*}, but preserve order of `S` and `T` options (the order of `S` and `T` in the spec is not significant). There can be any number of ampersand-separated variables; for each the wild card is optional. Useful for CPP as '`%{D*&U*&A*}`'.

`%{S:X}` Substitutes `X`, if the '`-S`' switch was given to GCC.

`%{!S:X}` Substitutes `X`, if the '`-S`' switch was *not* given to GCC.

`%{S*:X}` Substitutes `X` if one or more switches whose names start with `-S` are specified to GCC. Normally `X` is substituted only once, no matter how many such switches appeared. However, if `%*` appears somewhere in `X`, then `X` will be substituted once for each matching switch, with the `%*` replaced by the part of that switch that matched the `*`.

`%{.S:X}` Substitutes `X`, if processing a file with suffix `S`.

`%{!.S:X}` Substitutes `X`, if *not* processing a file with suffix `S`.

`%{,S:X}` Substitutes `X`, if processing a file for language `S`.

`%{!,S:X}` Substitutes `X`, if not processing a file for language `S`.

`%{S|P:X}` Substitutes `X` if either `-S` or `-P` was given to GCC. This may be combined with '`!`', '`.`', '`,`', and `*` sequences as well, although they have a stronger binding than the '`|`'. If `%*` appears in `X`, all of the alternatives must be starred, and only the first matching alternative is substituted.

For example, a spec string like this:

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

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

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

`%{S:X; T:Y; :D}`

If `S` was given to GCC, substitutes `X`; else if `T` was given to GCC, substitutes `Y`; else substitutes `D`. There can be as many clauses as you need. This may be combined with `.`, `,`, `!`, `|`, and `*` as needed.

The conditional text `X` in a %{S:X} or similar construct may contain other nested '`%`' constructs or spaces, or even newlines. They are processed as usual, as described above. Trailing white space in `X` is ignored. White space may also appear anywhere on the left side of the colon in these constructs, except between `.` or `*` and the corresponding word.

The '`-O`', '`-f`', '`-m`', and '`-W`' switches are handled specifically in these constructs. If another value of '`-O`' or the negated form of a '`-f`', '`-m`', or '`-W`' switch is found later in

the command line, the earlier switch value is ignored, except with `{S*}` where `S` is just one letter, which passes all matching options.

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

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

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

3.16 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.

3.17 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.

3.17.1 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 incor-

rect 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`’.

3.17.2 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. See Section 6.12 [Half-Precision], page 319, for more information.

-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.

3.17.3 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 `avr2`.

  GCC supports the following AVR devices and ISAs:

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

  `avr25`
  : "Classic" devices with up to 8 KiB of program memory and with the `MOVW` instruction.
  mcu = `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 KiB up to 64 KiB of program memory.
  mcu = `at43usb355`, `at76c711`.

  `avr31`
  : "Classic" devices with 128 KiB of program memory.
  mcu = `at43usb320`, `atmega103`.

`avr35` "Classic" devices with 16 KiB up to 64 KiB of program memory and with the `MOVW` instruction.
mcu = `at90usb162`, `at90usb82`, `atmega16u2`, `atmega32u2`, `atmega8u2`, `attiny167`.

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

`avr5` "Enhanced" devices with 16 KiB up to 64 KiB of program memory.
mcu = `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 KiB of program memory.
mcu = `at90can128`, `at90usb1286`, `at90usb1287`, `atmega128`, `atmega1280`, `atmega1281`, `atmega1284p`, `atmega128rfa1`.

`avr6` "Enhanced" devices with 3-byte PC, i.e. with more than 128 KiB of program memory.
mcu = `atmega2560`, `atmega2561`.

`avrxmega2`
"XMEGA" devices with more than 8 KiB and up to 64 KiB of program memory.
mcu = `atxmega16a4`, `atxmega16d4`, `atxmega16x1`, `atxmega32a4`, `atxmega32d4`, `atxmega32x1`.

`avrxmega4`
"XMEGA" devices with more than 64 KiB and up to 128 KiB of program memory.
mcu = `atxmega64a3`, `atxmega64d3`.

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

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

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

`avr1`
: This ISA is implemented by the minimal AVR core and supported for assembler only.
mcu = `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 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 bits of the stack pointer.

3.17.3.1 `EIND` and Devices with more than 128 Ki Bytes of Flash

Pointers in the implementation are 16 bits wide. The address of a function or label is represented as word address so that indirect jumps and calls can target any code address in the range of 64 Ki words.

In order to facilitate indirect jump on devices with more than 128 Ki bytes of program memory space, there is a special function register called `EIND` that serves as most significant part of the target address when `EICALL` or `EIJMP` instructions are used.

Indirect jumps and calls on these devices are handled as follows by the compiler and are subject to some limitations:

- The compiler never sets `EIND`.
- The compiler uses `EIND` 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.

3.17.3.2 Handling of the `RAMPD`, `RAMPX`, `RAMPY` and `RAMPZ` Special Function Registers

Some AVR devices support memories larger than the 64 KiB range that can be accessed with 16-bit pointers. To access memory locations outside this 64 KiB range, the 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], page 322 other than generic or `__flash` is used, then `RAMPZ` is set as needed before the operation.
- If the device supports RAM larger than 64 KiB and the compiler needs to change `RAMPZ` to accomplish an operation, `RAMPZ` is reset to zero after the operation.
- If the device comes with a specific `RAMP` register, the ISR prologue/epilogue saves/restores that SFR and initializes it with zero in case the ISR code might (implicitly) use it.
- RAM larger than 64 KiB is not supported by GCC for AVR targets. If you use inline assembler to read from locations outside the 16-bit address range and change one of the `RAMP` registers, you must reset it to zero after the access.

3.17.3.3 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], page 322 and Section 6.55.4 [AVR Built-in Functions], page 532.

`__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 Rn,Z` and `ELPM Rn,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 Rn,Z` and `LPM Rn,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 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 KiB of program memory. This also means that the program counter (PC) is 3 bytes wide.

`__AVR_2_BYTE_PC__`
: The program counter (PC) is 2 bytes wide. This is the case for devices with up to 128 KiB of program memory.

`__AVR_HAVE_8BIT_SP__`
`__AVR_HAVE_16BIT_SP__`
: The stack pointer (SP) register is treated as 8-bit respectively 16-bit register by the compiler. The definition of these macros is affected by `-mtiny-stack`.

`__AVR_HAVE_SPH__`
`__AVR_SP8__`
: The device has the SPH (high part of stack pointer) special function register or has an 8-bit stack pointer, respectively. The definition of these macros is affected by `-mmcu=` and in the cases of `-mmcu=avr2` and `-mmcu=avr25` also by `-msp8`.

`__AVR_HAVE_RAMPD__`
`__AVR_HAVE_RAMPX__`
`__AVR_HAVE_RAMPY__`
`__AVR_HAVE_RAMPZ__`
: The device has the `RAMPD`, `RAMPX`, `RAMPY`, `RAMPZ` special function register, respectively.

`__NO_INTERRUPTS__`
: This macro reflects the `-mno-interrupts` command line option.

`__AVR_ERRATA_SKIP__`
`__AVR_ERRATA_SKIP_JMP_CALL__`
: Some AVR devices (AT90S8515, ATmega103) must not skip 32-bit instructions because of a hardware erratum. Skip instructions are `SBRS`, `SBRC`, `SBIS`, `SBIC` and `CPSE`. The second macro is only defined if `__AVR_HAVE_JMP_CALL__` is also set.

`__AVR_SFR_OFFSET__=`*offset*
: Instructions that can address I/O special function registers directly like `IN`, `OUT`, `SBI`, etc. may use a different address as if addressed by an instruction to access RAM like `LD` or `STS`. This offset depends on the device architecture and has to be subtracted from the RAM address in order to get the respective I/O address.

`__WITH_AVRLIBC__`
: The compiler is configured to be used together with AVR-Libc. See the `--with-avrlibc` configure option.

3.17.4 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.

3.17.5 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.

3.17.6 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.

3.17.7 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.

3.17.8 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.

3.17.9 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`'.

`-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.

3.17.10 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.

3.17.11 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.

3.17.12 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`'.

3.17.13 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.

3.17.14 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`'. See Section "`ld` and the H8/300" in *Using ld*, for a fuller description.

`-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.

3.17.15 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.

3.17.16 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`'. See Section 6.30 [Function Attributes], page 332.

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`'. See Section 6.30 [Function Attributes], page 332.

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`'. See Section 6.30 [Function Attributes], page 332.

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 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 1) follow the ABI used by other x86 targets, which is the default on Solaris 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 Section 6.55.7 [X86 Built-in Functions], page 537, 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 Section 6.51 [__sync Builtins], page 425 or Section 6.52 [__atomic Builtins], page 427 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 Section 6.54 [Other Builtins], page 433 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`'. See Section 6.30 [Function Attributes], page 332.

`-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.

3.17.17 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 pre-defined 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 pre-defined 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 Section 3.17.16 [i386 and x86-64 Options], page 204 for standard options.

3.17.18 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.

3.17.19 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.

3.17.20 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.

3.17.21 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.

3.17.22 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.

3.17.23 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 '`__mcarch`' and '`__mcarch__`' when tuning for 680x0 architecture *arch*. It also defines '`mcarch`' 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 '`__muarch__`' 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.

3.17.24 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.

3.17.25 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.

3.17.26 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 '`vX.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`*’.

3.17.27 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. See Section 6.30 [Function Attributes], page 332, for more information.

`-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. See Section 6.55.8 [MIPS DSP Built-in Functions], page 557. 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. See Section 6.55.8 [MIPS DSP Built-in Functions], page 557. 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. See Section 6.55.9 [MIPS Paired-Single Support], page 561. 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. See Section 6.55.10.3 [MIPS-3D Built-in Functions], page 565. 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`'.

3.17.28 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.

3.17.29 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__`'.

3.17.30 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`'.

3.17.31 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.

3.17.32 PowerPC Options

These are listed under See Section 3.17.34 [RS/6000 and PowerPC Options], page 249.

3.17.33 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.

3.17.34 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.

3.17.35 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.

3.17.36 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*.

3.17.37 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.

3.17.38 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 Section 6.52 [__atomic Builtins], page 427.

`-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.

3.17.39 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`'.

3.17.40 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.

3.17.41 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.

3.17.42 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.

3.17.43 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.

3.17.44 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.

3.17.45 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.

3.17.46 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.

3.17.47 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 (see Section 3.13 [Link Options], page 154); '`-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.

3.17.48 x86-64 Options

These are listed under See Section 3.17.16 [i386 and x86-64 Options], page 204.

3.17.49 Xstormy16 Options

These options are defined for Xstormy16:

`-msim` Choose startup files and linker script suitable for the simulator.

3.17.50 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.

3.17.51 zSeries Options

These are listed under See Section 3.17.36 [S/390 and zSeries Options], page 265.

3.18 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 (see Section 6.59 [Thread-Local], page 618). 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.

3.19 Environment Variables Affecting GCC

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`' (see Section 3.14 [Directory Options], page 158). These take precedence over places specified using environment variables, which in turn take precedence over those specified by the configuration of GCC. See Section "Controlling the Compilation Driver '`gcc`'" in *GNU Compiler Collection (GCC) Internals*.

`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`' (see Section 3.11 [Preprocessor Options], page 143), 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. See Section 3.11 [Preprocessor Options], page 143.

3.20 Using Precompiled Headers

Often large projects have many header files that are included in every source file. The time the compiler takes to process these header files over and over again can account for nearly all of the time required to build the project. To make builds faster, GCC allows users to 'precompile' a header file; then, if builds can use the precompiled header file they will be much faster.

To create a precompiled header file, simply compile it as you would any other file, if necessary using the '`-x`' option to make the driver treat it as a C or C++ header file. You will probably want to use a tool like `make` to keep the precompiled header up-to-date when the headers it contains change.

A precompiled header file will be searched for when `#include` is seen in the compilation. As it searches for the included file (see Section "Search Path" in *The C Preprocessor*) the compiler looks for a precompiled header in each directory just before it looks for the include file in that directory. The name searched for is the name specified in the `#include` with '`.gch`' appended. If the precompiled header file can't be used, it is ignored.

For instance, if you have `#include "all.h"`, and you have '`all.h.gch`' in the same directory as '`all.h`', then the precompiled header file will be used if possible, and the original header will be used otherwise.

Alternatively, you might decide to put the precompiled header file in a directory and use '`-I`' to ensure that directory is searched before (or instead of) the directory containing the original header. Then, if you want to check that the precompiled header file is always used, you can put a file of the same name as the original header in this directory containing an `#error` command.

This also works with '`-include`'. So yet another way to use precompiled headers, good for projects not designed with precompiled header files in mind, is to simply take most of the header files used by a project, include them from another header file, precompile that header file, and '`-include`' the precompiled header. If the header files have guards against multiple inclusion, they will be skipped because they've already been included (in the precompiled header).

If you need to precompile the same header file for different languages, targets, or compiler options, you can instead make a *directory* named like '`all.h.gch`', and put each precompiled header in the directory, perhaps using '`-o`'. It doesn't matter what you call the files in the directory, every precompiled header in the directory will be considered. The first precompiled header encountered in the directory that is valid for this compilation will be used; they're searched in no particular order.

There are many other possibilities, limited only by your imagination, good sense, and the constraints of your build system.

A precompiled header file can be used only when these conditions apply:

- Only one precompiled header can be used in a particular compilation.
- A precompiled header can't be used once the first C token is seen. You can have preprocessor directives before a precompiled header; you can even include a precompiled header from inside another header, so long as there are no C tokens before the `#include`.
- The precompiled header file must be produced for the same language as the current compilation. You can't use a C precompiled header for a C++ compilation.

- The precompiled header file must have been produced by the same compiler binary as the current compilation is using.
- Any macros defined before the precompiled header is included must either be defined in the same way as when the precompiled header was generated, or must not affect the precompiled header, which usually means that they don't appear in the precompiled header at all.

  The '`-D`' option is one way to define a macro before a precompiled header is included; using a `#define` can also do it. There are also some options that define macros implicitly, like '`-O`' and '`-Wdeprecated`'; the same rule applies to macros defined this way.
- If debugging information is output when using the precompiled header, using '`-g`' or similar, the same kind of debugging information must have been output when building the precompiled header. However, a precompiled header built using '`-g`' can be used in a compilation when no debugging information is being output.
- The same '`-m`' options must generally be used when building and using the precompiled header. See Section 3.17 [Submodel Options], page 167, for any cases where this rule is relaxed.
- Each of the following options must be the same when building and using the precompiled header:

  ```
  -fexceptions
  ```
- Some other command-line options starting with '`-f`', '`-p`', or '`-O`' must be defined in the same way as when the precompiled header was generated. At present, it's not clear which options are safe to change and which are not; the safest choice is to use exactly the same options when generating and using the precompiled header. The following are known to be safe:

  ```
  -fmessage-length= -fpreprocessed -fsched-interblock
  -fsched-spec -fsched-spec-load -fsched-spec-load-dangerous
  -fsched-verbose=number -fschedule-insns -fvisibility=
  -pedantic-errors
  ```

For all of these except the last, the compiler will automatically ignore the precompiled header if the conditions aren't met. If you find an option combination that doesn't work and doesn't cause the precompiled header to be ignored, please consider filing a bug report, see Chapter 12 [Bugs], page 679.

If you do use differing options when generating and using the precompiled header, the actual behavior will be a mixture of the behavior for the options. For instance, if you use '`-g`' to generate the precompiled header but not when using it, you may or may not get debugging information for routines in the precompiled header.

4 C Implementation-defined behavior

A conforming implementation of ISO C is required to document its choice of behavior in each of the areas that are designated "implementation defined". The following lists all such areas, along with the section numbers from the ISO/IEC 9899:1990 and ISO/IEC 9899:1999 standards. Some areas are only implementation-defined in one version of the standard.

Some choices depend on the externally determined ABI for the platform (including standard character encodings) which GCC follows; these are listed as "determined by ABI" below. See Chapter 9 [Binary Compatibility], page 651, and `http://gcc.gnu.org/readings.html`. Some choices are documented in the preprocessor manual. See Section "Implementation-defined behavior" in *The C Preprocessor*. Some choices are made by the library and operating system (or other environment when compiling for a freestanding environment); refer to their documentation for details.

4.1 Translation

- *How a diagnostic is identified (C90 3.7, C99 3.10, C90 and C99 5.1.1.3).*

  Diagnostics consist of all the output sent to stderr by GCC.

- *Whether each nonempty sequence of white-space characters other than new-line is retained or replaced by one space character in translation phase 3 (C90 and C99 5.1.1.2).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

4.2 Environment

The behavior of most of these points are dependent on the implementation of the C library, and are not defined by GCC itself.

- *The mapping between physical source file multibyte characters and the source character set in translation phase 1 (C90 and C99 5.1.1.2).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

4.3 Identifiers

- *Which additional multibyte characters may appear in identifiers and their correspondence to universal character names (C99 6.4.2).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

- *The number of significant initial characters in an identifier (C90 6.1.2, C90 and C99 5.2.4.1, C99 6.4.2).*

  For internal names, all characters are significant. For external names, the number of significant characters are defined by the linker; for almost all targets, all characters are significant.

- *Whether case distinctions are significant in an identifier with external linkage (C90 6.1.2).*

  This is a property of the linker. C99 requires that case distinctions are always significant in identifiers with external linkage and systems without this property are not supported by GCC.

4.4 Characters

- *The number of bits in a byte (C90 3.4, C99 3.6).*

  Determined by ABI.

- *The values of the members of the execution character set (C90 and C99 5.2.1).*

  Determined by ABI.

- *The unique value of the member of the execution character set produced for each of the standard alphabetic escape sequences (C90 and C99 5.2.2).*

  Determined by ABI.

- *The value of a* `char` *object into which has been stored any character other than a member of the basic execution character set (C90 6.1.2.5, C99 6.2.5).*

  Determined by ABI.

- *Which of* `signed char` *or* `unsigned char` *has the same range, representation, and behavior as "plain"* `char` *(C90 6.1.2.5, C90 6.2.1.1, C99 6.2.5, C99 6.3.1.1).*

  Determined by ABI. The options '`-funsigned-char`' and '`-fsigned-char`' change the default. See Section 3.4 [Options Controlling C Dialect], page 29.

- *The mapping of members of the source character set (in character constants and string literals) to members of the execution character set (C90 6.1.3.4, C99 6.4.4.4, C90 and C99 5.1.1.2).*

  Determined by ABI.

- *The value of an integer character constant containing more than one character or containing a character or escape sequence that does not map to a single-byte execution character (C90 6.1.3.4, C99 6.4.4.4).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

- *The value of a wide character constant containing more than one multibyte character, or containing a multibyte character or escape sequence not represented in the extended execution character set (C90 6.1.3.4, C99 6.4.4.4).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

- *The current locale used to convert a wide character constant consisting of a single multibyte character that maps to a member of the extended execution character set into a corresponding wide character code (C90 6.1.3.4, C99 6.4.4.4).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

- *The current locale used to convert a wide string literal into corresponding wide character codes (C90 6.1.4, C99 6.4.5).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

- *The value of a string literal containing a multibyte character or escape sequence not represented in the execution character set (C90 6.1.4, C99 6.4.5).*

  See Section "Implementation-defined behavior" in *The C Preprocessor*.

4.5 Integers

- *Any extended integer types that exist in the implementation (C99 6.2.5).*

  GCC does not support any extended integer types.

- *Whether signed integer types are represented using sign and magnitude, two's complement, or one's complement, and whether the extraordinary value is a trap representation or an ordinary value (C99 6.2.6.2).*

  GCC supports only two's complement integer types, and all bit patterns are ordinary values.

- *The rank of any extended integer type relative to another extended integer type with the same precision (C99 6.3.1.1).*

  GCC does not support any extended integer types.

- *The result of, or the signal raised by, converting an integer to a signed integer type when the value cannot be represented in an object of that type (C90 6.2.1.2, C99 6.3.1.3).*

  For conversion to a type of width N, the value is reduced modulo 2^N to be within range of the type; no signal is raised.

- *The results of some bitwise operations on signed integers (C90 6.3, C99 6.5).*

  Bitwise operators act on the representation of the value including both the sign and value bits, where the sign bit is considered immediately above the highest-value value bit. Signed '`>>`' acts on negative numbers by sign extension.

  GCC does not use the latitude given in C99 only to treat certain aspects of signed '`<<`' as undefined, but this is subject to change.

- *The sign of the remainder on integer division (C90 6.3.5).*

  GCC always follows the C99 requirement that the result of division is truncated towards zero.

4.6 Floating point

- *The accuracy of the floating-point operations and of the library functions in* `<math.h>` *and* `<complex.h>` *that return floating-point results (C90 and C99 5.2.4.2.2).*

  The accuracy is unknown.

- *The rounding behaviors characterized by non-standard values of* `FLT_ROUNDS` *(C90 and C99 5.2.4.2.2).*

  GCC does not use such values.

- *The evaluation methods characterized by non-standard negative values of* `FLT_EVAL_METHOD` *(C99 5.2.4.2.2).*

  GCC does not use such values.

- *The direction of rounding when an integer is converted to a floating-point number that cannot exactly represent the original value (C90 6.2.1.3, C99 6.3.1.4).*

  C99 Annex F is followed.

- *The direction of rounding when a floating-point number is converted to a narrower floating-point number (C90 6.2.1.4, C99 6.3.1.5).*

  C99 Annex F is followed.

- *How the nearest representable value or the larger or smaller representable value immediately adjacent to the nearest representable value is chosen for certain floating constants (C90 6.1.3.1, C99 6.4.4.2).*

  C99 Annex F is followed.

- *Whether and how floating expressions are contracted when not disallowed by the* `FP_CONTRACT` *pragma (C99 6.5).*

  Expressions are currently only contracted if '`-funsafe-math-optimizations`' or '`-ffast-math`' are used. This is subject to change.

- *The default state for the* `FENV_ACCESS` *pragma (C99 7.6.1).*

  This pragma is not implemented, but the default is to "off" unless '`-frounding-math`' is used in which case it is "on".

- *Additional floating-point exceptions, rounding modes, environments, and classifications, and their macro names (C99 7.6, C99 7.12).*

  This is dependent on the implementation of the C library, and is not defined by GCC itself.

- *The default state for the* `FP_CONTRACT` *pragma (C99 7.12.2).*

  This pragma is not implemented. Expressions are currently only contracted if '`-funsafe-math-optimizations`' or '`-ffast-math`' are used. This is subject to change.

- *Whether the "inexact" floating-point exception can be raised when the rounded result actually does equal the mathematical result in an IEC 60559 conformant implementation (C99 F.9).*

  This is dependent on the implementation of the C library, and is not defined by GCC itself.

- *Whether the "underflow" (and "inexact") floating-point exception can be raised when a result is tiny but not inexact in an IEC 60559 conformant implementation (C99 F.9).*

  This is dependent on the implementation of the C library, and is not defined by GCC itself.

4.7 Arrays and pointers

- *The result of converting a pointer to an integer or vice versa (C90 6.3.4, C99 6.3.2.3).*

  A cast from pointer to integer discards most-significant bits if the pointer representation is larger than the integer type, sign-extends[1] if the pointer representation is smaller than the integer type, otherwise the bits are unchanged.

  A cast from integer to pointer discards most-significant bits if the pointer representation is smaller than the integer type, extends according to the signedness of the integer type if the pointer representation is larger than the integer type, otherwise the bits are unchanged.

  When casting from pointer to integer and back again, the resulting pointer must reference the same object as the original pointer, otherwise the behavior is undefined. That is, one may not use integer arithmetic to avoid the undefined behavior of pointer arithmetic as proscribed in C99 6.5.6/8.

- *The size of the result of subtracting two pointers to elements of the same array (C90 6.3.6, C99 6.5.6).*

  The value is as specified in the standard and the type is determined by the ABI.

[1] Future versions of GCC may zero-extend, or use a target-defined `ptr_extend` pattern. Do not rely on sign extension.

4.8 Hints

- *The extent to which suggestions made by using the* `register` *storage-class specifier are effective (C90 6.5.1, C99 6.7.1).*

  The `register` specifier affects code generation only in these ways:

  - When used as part of the register variable extension, see Section 6.44 [Explicit Reg Vars], page 418.
  - When '`-O0`' is in use, the compiler allocates distinct stack memory for all variables that do not have the `register` storage-class specifier; if `register` is specified, the variable may have a shorter lifespan than the code would indicate and may never be placed in memory.
  - On some rare x86 targets, `setjmp` doesn't save the registers in all circumstances. In those cases, GCC doesn't allocate any variables in registers unless they are marked `register`.

- *The extent to which suggestions made by using the inline function specifier are effective (C99 6.7.4).*

  GCC will not inline any functions if the '`-fno-inline`' option is used or if '`-O0`' is used. Otherwise, GCC may still be unable to inline a function for many reasons; the '`-Winline`' option may be used to determine if a function has not been inlined and why not.

4.9 Structures, unions, enumerations, and bit-fields

- *A member of a union object is accessed using a member of a different type (C90 6.3.2.3).*

  The relevant bytes of the representation of the object are treated as an object of the type used for the access. See [Type-punning], page 114. This may be a trap representation.

- *Whether a "plain"* `int` *bit-field is treated as a* `signed int` *bit-field or as an* `unsigned int` *bit-field (C90 6.5.2, C90 6.5.2.1, C99 6.7.2, C99 6.7.2.1).*

  By default it is treated as `signed int` but this may be changed by the '`-funsigned-bitfields`' option.

- *Allowable bit-field types other than* `_Bool`, `signed int`, *and* `unsigned int` *(C99 6.7.2.1).*

  No other types are permitted in strictly conforming mode.

- *Whether a bit-field can straddle a storage-unit boundary (C90 6.5.2.1, C99 6.7.2.1).*

  Determined by ABI.

- *The order of allocation of bit-fields within a unit (C90 6.5.2.1, C99 6.7.2.1).*

  Determined by ABI.

- *The alignment of non-bit-field members of structures (C90 6.5.2.1, C99 6.7.2.1).*

  Determined by ABI.

- *The integer type compatible with each enumerated type (C90 6.5.2.2, C99 6.7.2.2).*

  Normally, the type is `unsigned int` if there are no negative values in the enumeration, otherwise `int`. If '`-fshort-enums`' is specified, then if there are negative values it is the first of `signed char`, `short` and `int` that can represent all the values, otherwise it

is the first of `unsigned char`, `unsigned short` and `unsigned int` that can represent all the values.

On some targets, '`-fshort-enums`' is the default; this is determined by the ABI.

4.10 Qualifiers

- *What constitutes an access to an object that has volatile-qualified type (C90 6.5.3, C99 6.7.3).*

  Such an object is normally accessed by pointers and used for accessing hardware. In most expressions, it is intuitively obvious what is a read and what is a write. For example

  ```
  volatile int *dst = somevalue;
  volatile int *src = someothervalue;
  *dst = *src;
  ```

  will cause a read of the volatile object pointed to by *src* and store the value into the volatile object pointed to by *dst*. There is no guarantee that these reads and writes are atomic, especially for objects larger than `int`.

  However, if the volatile storage is not being modified, and the value of the volatile storage is not used, then the situation is less obvious. For example

  ```
  volatile int *src = somevalue;
  *src;
  ```

  According to the C standard, such an expression is an rvalue whose type is the unqualified version of its original type, i.e. `int`. Whether GCC interprets this as a read of the volatile object being pointed to or only as a request to evaluate the expression for its side-effects depends on this type.

  If it is a scalar type, or on most targets an aggregate type whose only member object is of a scalar type, or a union type whose member objects are of scalar types, the expression is interpreted by GCC as a read of the volatile object; in the other cases, the expression is only evaluated for its side-effects.

4.11 Declarators

- *The maximum number of declarators that may modify an arithmetic, structure or union type (C90 6.5.4).*

  GCC is only limited by available memory.

4.12 Statements

- *The maximum number of* `case` *values in a* `switch` *statement (C90 6.6.4.2).*

  GCC is only limited by available memory.

4.13 Preprocessing directives

See Section "Implementation-defined behavior" in *The C Preprocessor*, for details of these aspects of implementation-defined behavior.

- *How sequences in both forms of header names are mapped to headers or external source file names (C90 6.1.7, C99 6.4.7).*

- *Whether the value of a character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set (C90 6.8.1, C99 6.10.1).*
- *Whether the value of a single-character character constant in a constant expression that controls conditional inclusion may have a negative value (C90 6.8.1, C99 6.10.1).*
- *The places that are searched for an included '<>' delimited header, and how the places are specified or the header is identified (C90 6.8.2, C99 6.10.2).*
- *How the named source file is searched for in an included '""' delimited header (C90 6.8.2, C99 6.10.2).*
- *The method by which preprocessing tokens (possibly resulting from macro expansion) in a* `#include` *directive are combined into a header name (C90 6.8.2, C99 6.10.2).*
- *The nesting limit for* `#include` *processing (C90 6.8.2, C99 6.10.2).*
- *Whether the '#' operator inserts a '\' character before the '\' character that begins a universal character name in a character constant or string literal (C99 6.10.3.2).*
- *The behavior on each recognized non-*`STDC` `#pragma` *directive (C90 6.8.6, C99 6.10.6).*

  See Section "Pragmas" in *The C Preprocessor*, for details of pragmas accepted by GCC on all targets. See Section 6.57 [Pragmas Accepted by GCC], page 610, for details of target-specific pragmas.
- *The definitions for* `__DATE__` *and* `__TIME__` *when respectively, the date and time of translation are not available (C90 6.8.8, C99 6.10.8).*

4.14 Library functions

The behavior of most of these points are dependent on the implementation of the C library, and are not defined by GCC itself.

- *The null pointer constant to which the macro* `NULL` *expands (C90 7.1.6, C99 7.17).*

  In `<stddef.h>`, `NULL` expands to `((void *)0)`. GCC does not provide the other headers which define `NULL` and some library implementations may use other definitions in those headers.

4.15 Architecture

- *The values or expressions assigned to the macros specified in the headers* `<float.h>`, `<limits.h>`, *and* `<stdint.h>` *(C90 and C99 5.2.4.2, C99 7.18.2, C99 7.18.3).*

  Determined by ABI.
- *The number, order, and encoding of bytes in any object (when not explicitly specified in this International Standard) (C99 6.2.6.1).*

  Determined by ABI.
- *The value of the result of the* `sizeof` *operator (C90 6.3.3.4, C99 6.5.3.4).*

  Determined by ABI.

4.16 Locale-specific behavior

The behavior of these points are dependent on the implementation of the C library, and are not defined by GCC itself.

5 C++ Implementation-defined behavior

A conforming implementation of ISO C++ is required to document its choice of behavior in each of the areas that are designated "implementation defined". The following lists all such areas, along with the section numbers from the ISO/IEC 14822:1998 and ISO/IEC 14822:2003 standards. Some areas are only implementation-defined in one version of the standard.

Some choices depend on the externally determined ABI for the platform (including standard character encodings) which GCC follows; these are listed as "determined by ABI" below. See Chapter 9 [Binary Compatibility], page 651, and `http://gcc.gnu.org/readings.html`. Some choices are documented in the preprocessor manual. See Section "Implementation-defined behavior" in *The C Preprocessor*. Some choices are documented in the corresponding document for the C language. See Chapter 4 [C Implementation], page 299. Some choices are made by the library and operating system (or other environment when compiling for a freestanding environment); refer to their documentation for details.

5.1 Conditionally-supported behavior

Each implementation shall include documentation that identifies all conditionally-supported constructs that it does not support (C++0x 1.4).

- *Whether an argument of class type with a non-trivial copy constructor or destructor can be passed to ... (C++0x 5.2.2).*

  Such argument passing is not supported.

5.2 Exception handling

- *In the situation where no matching handler is found, it is implementation-defined whether or not the stack is unwound before std::terminate() is called (C++98 15.5.1).*

  The stack is not unwound before std::terminate is called.

6 Extensions to the C Language Family

GNU C provides several language features not found in ISO standard C. (The '`-pedantic`' option directs GCC to print a warning message if any of these features is used.) To test for the availability of these features in conditional compilation, check for a predefined macro `__GNUC__`, which is always defined under GCC.

These extensions are available in C and Objective-C. Most of them are also available in C++. See Chapter 7 [Extensions to the C++ Language], page 623, for extensions that apply *only* to C++.

Some features that are in ISO C99 but not C90 or C++ are also, as extensions, accepted by GCC in C90 mode and in C++.

6.1 Statements and Declarations in Expressions

A compound statement enclosed in parentheses may appear as an expression in GNU C. This allows you to use loops, switches, and local variables within an expression.

Recall that a compound statement is a sequence of statements surrounded by braces; in this construct, parentheses go around the braces. For example:

```
({ int y = foo (); int z;
   if (y > 0) z = y;
   else z = - y;
   z; })
```

is a valid (though slightly more complex than necessary) expression for the absolute value of `foo ()`.

The last thing in the compound statement should be an expression followed by a semicolon; the value of this subexpression serves as the value of the entire construct. (If you use some other kind of statement last within the braces, the construct has type `void`, and thus effectively no value.)

This feature is especially useful in making macro definitions "safe" (so that they evaluate each operand exactly once). For example, the "maximum" function is commonly defined as a macro in standard C as follows:

```
#define max(a,b) ((a) > (b) ? (a) : (b))
```

But this definition computes either *a* or *b* twice, with bad results if the operand has side effects. In GNU C, if you know the type of the operands (here taken as `int`), you can define the macro safely as follows:

```
#define maxint(a,b) \
  ({int _a = (a), _b = (b); _a > _b ? _a : _b; })
```

Embedded statements are not allowed in constant expressions, such as the value of an enumeration constant, the width of a bit-field, or the initial value of a static variable.

If you don't know the type of the operand, you can still do this, but you must use `typeof` (see Section 6.6 [Typeof], page 316).

In G++, the result value of a statement expression undergoes array and function pointer decay, and is returned by value to the enclosing expression. For instance, if `A` is a class, then

```
A a;

({a;}).Foo ()
```

will construct a temporary `A` object to hold the result of the statement expression, and that will be used to invoke `Foo`. Therefore the `this` pointer observed by `Foo` will not be the address of `a`.

Any temporaries created within a statement within a statement expression will be destroyed at the statement's end. This makes statement expressions inside macros slightly different from function calls. In the latter case temporaries introduced during argument evaluation will be destroyed at the end of the statement that includes the function call. In the statement expression case they will be destroyed during the statement expression. For instance,

```
#define macro(a)  ({__typeof__(a) b = (a); b + 3; })
template<typename T> T function(T a) { T b = a; return b + 3; }

void foo ()
{
  macro (X ());
  function (X ());
}
```

will have different places where temporaries are destroyed. For the `macro` case, the temporary `X` will be destroyed just after the initialization of `b`. In the `function` case that temporary will be destroyed when the function returns.

These considerations mean that it is probably a bad idea to use statement-expressions of this form in header files that are designed to work with C++. (Note that some versions of the GNU C Library contained header files using statement-expression that lead to precisely this bug.)

Jumping into a statement expression with `goto` or using a `switch` statement outside the statement expression with a `case` or `default` label inside the statement expression is not permitted. Jumping into a statement expression with a computed `goto` (see Section 6.3 [Labels as Values], page 311) yields undefined behavior. Jumping out of a statement expression is permitted, but if the statement expression is part of a larger expression then it is unspecified which other subexpressions of that expression have been evaluated except where the language definition requires certain subexpressions to be evaluated before or after the statement expression. In any case, as with a function call the evaluation of a statement expression is not interleaved with the evaluation of other parts of the containing expression. For example,

```
foo (), (({ bar1 (); goto a; 0; }) + bar2 ()), baz();
```

will call `foo` and `bar1` and will not call `baz` but may or may not call `bar2`. If `bar2` is called, it will be called after `foo` and before `bar1`

6.2 Locally Declared Labels

GCC allows you to declare *local labels* in any nested block scope. A local label is just like an ordinary label, but you can only reference it (with a `goto` statement, or by taking its address) within the block in which it was declared.

A local label declaration looks like this:

```
__label__ label;
```

or

```
__label__ label1, label2, /* ... */;
```

Local label declarations must come at the beginning of the block, before any ordinary declarations or statements.

The label declaration defines the label *name*, but does not define the label itself. You must do this in the usual way, with *label*`:`, within the statements of the statement expression.

The local label feature is useful for complex macros. If a macro contains nested loops, a `goto` can be useful for breaking out of them. However, an ordinary label whose scope is the whole function cannot be used: if the macro can be expanded several times in one function, the label will be multiply defined in that function. A local label avoids this problem. For example:

```
#define SEARCH(value, array, target)              \
do {                                              \
  __label__ found;                                \
  typeof (target) _SEARCH_target = (target);      \
  typeof (*(array)) *_SEARCH_array = (array);     \
  int i, j;                                       \
  int value;                                      \
  for (i = 0; i < max; i++)                       \
    for (j = 0; j < max; j++)                     \
      if (_SEARCH_array[i][j] == _SEARCH_target)  \
        { (value) = i; goto found; }              \
  (value) = -1;                                   \
 found:;                                          \
} while (0)
```

This could also be written using a statement-expression:

```
#define SEARCH(array, target)                     \
({                                                \
  __label__ found;                                \
  typeof (target) _SEARCH_target = (target);      \
  typeof (*(array)) *_SEARCH_array = (array);     \
  int i, j;                                       \
  int value;                                      \
  for (i = 0; i < max; i++)                       \
    for (j = 0; j < max; j++)                     \
      if (_SEARCH_array[i][j] == _SEARCH_target)  \
        { value = i; goto found; }                \
  value = -1;                                     \
 found:                                           \
  value;                                          \
})
```

Local label declarations also make the labels they declare visible to nested functions, if there are any. See Section 6.4 [Nested Functions], page 312, for details.

6.3 Labels as Values

You can get the address of a label defined in the current function (or a containing function) with the unary operator '`&&`'. The value has type `void *`. This value is a constant and can be used wherever a constant of that type is valid. For example:

```
void *ptr;
```

```
/* ... */
ptr = &&foo;
```

To use these values, you need to be able to jump to one. This is done with the computed goto statement[1], `goto *exp;`. For example,

```
goto *ptr;
```

Any expression of type `void *` is allowed.

One way of using these constants is in initializing a static array that will serve as a jump table:

```
static void *array[] = { &&foo, &&bar, &&hack };
```

Then you can select a label with indexing, like this:

```
goto *array[i];
```

Note that this does not check whether the subscript is in bounds—array indexing in C never does that.

Such an array of label values serves a purpose much like that of the `switch` statement. The `switch` statement is cleaner, so use that rather than an array unless the problem does not fit a `switch` statement very well.

Another use of label values is in an interpreter for threaded code. The labels within the interpreter function can be stored in the threaded code for super-fast dispatching.

You may not use this mechanism to jump to code in a different function. If you do that, totally unpredictable things will happen. The best way to avoid this is to store the label address only in automatic variables and never pass it as an argument.

An alternate way to write the above example is

```
static const int array[] = { &&foo - &&foo, &&bar - &&foo,
                             &&hack - &&foo };
goto *(&&foo + array[i]);
```

This is more friendly to code living in shared libraries, as it reduces the number of dynamic relocations that are needed, and by consequence, allows the data to be read-only.

The `&&foo` expressions for the same label might have different values if the containing function is inlined or cloned. If a program relies on them being always the same, `__attribute__((__noinline__,__noclone__))` should be used to prevent inlining and cloning. If `&&foo` is used in a static variable initializer, inlining and cloning is forbidden.

6.4 Nested Functions

A *nested function* is a function defined inside another function. (Nested functions are not supported for GNU C++.) The nested function's name is local to the block where it is defined. For example, here we define a nested function named `square`, and call it twice:

```
foo (double a, double b)
{
  double square (double z) { return z * z; }

  return square (a) + square (b);
}
```

[1] The analogous feature in Fortran is called an assigned goto, but that name seems inappropriate in C, where one can do more than simply store label addresses in label variables.

The nested function can access all the variables of the containing function that are visible at the point of its definition. This is called *lexical scoping*. For example, here we show a nested function which uses an inherited variable named `offset`:

```
bar (int *array, int offset, int size)
{
  int access (int *array, int index)
    { return array[index + offset]; }
  int i;
  /* ... */
  for (i = 0; i < size; i++)
    /* ... */ access (array, i) /* ... */
}
```

Nested function definitions are permitted within functions in the places where variable definitions are allowed; that is, in any block, mixed with the other declarations and statements in the block.

It is possible to call the nested function from outside the scope of its name by storing its address or passing the address to another function:

```
hack (int *array, int size)
{
  void store (int index, int value)
    { array[index] = value; }

  intermediate (store, size);
}
```

Here, the function `intermediate` receives the address of `store` as an argument. If `intermediate` calls `store`, the arguments given to `store` are used to store into `array`. But this technique works only so long as the containing function (`hack`, in this example) does not exit.

If you try to call the nested function through its address after the containing function has exited, all hell will break loose. If you try to call it after a containing scope level has exited, and if it refers to some of the variables that are no longer in scope, you may be lucky, but it's not wise to take the risk. If, however, the nested function does not refer to anything that has gone out of scope, you should be safe.

GCC implements taking the address of a nested function using a technique called *trampolines*. This technique was described in *Lexical Closures for C++* (Thomas M. Breuel, USENIX C++ Conference Proceedings, October 17-21, 1988).

A nested function can jump to a label inherited from a containing function, provided the label was explicitly declared in the containing function (see Section 6.2 [Local Labels], page 310). Such a jump returns instantly to the containing function, exiting the nested function which did the `goto` and any intermediate functions as well. Here is an example:

```
bar (int *array, int offset, int size)
{
  __label__ failure;
  int access (int *array, int index)
    {
      if (index > size)
        goto failure;
      return array[index + offset];
    }
  int i;
  /* ... */
  for (i = 0; i < size; i++)
    /* ... */ access (array, i) /* ... */
  /* ... */
  return 0;

 /* Control comes here from access
    if it detects an error.  */
 failure:
  return -1;
}
```

A nested function always has no linkage. Declaring one with `extern` or `static` is erroneous. If you need to declare the nested function before its definition, use `auto` (which is otherwise meaningless for function declarations).

```
bar (int *array, int offset, int size)
{
  __label__ failure;
  auto int access (int *, int);
  /* ... */
  int access (int *array, int index)
    {
      if (index > size)
        goto failure;
      return array[index + offset];
    }
  /* ... */
}
```

6.5 Constructing Function Calls

Using the built-in functions described below, you can record the arguments a function received, and call another function with the same arguments, without knowing the number or types of the arguments.

You can also record the return value of that function call, and later return that value, without knowing what data type the function tried to return (as long as your caller expects that data type).

However, these built-in functions may interact badly with some sophisticated features or other extensions of the language. It is, therefore, not recommended to use them outside very simple functions acting as mere forwarders for their arguments.

`void * __builtin_apply_args ()` [Built-in Function]

This built-in function returns a pointer to data describing how to perform a call with the same arguments as were passed to the current function.

The function saves the arg pointer register, structure value address, and all registers that might be used to pass arguments to a function into a block of memory allocated on the stack. Then it returns the address of that block.

`void * __builtin_apply (`*void (*function)()*`, `*void *arguments*`, `*size_t size*`)` [Built-in Function]

This built-in function invokes *function* with a copy of the parameters described by *arguments* and *size*.

The value of *arguments* should be the value returned by `__builtin_apply_args`. The argument *size* specifies the size of the stack argument data, in bytes.

This function returns a pointer to data describing how to return whatever value was returned by *function*. The data is saved in a block of memory allocated on the stack.

It is not always simple to compute the proper value for *size*. The value is used by `__builtin_apply` to compute the amount of data that should be pushed on the stack and copied from the incoming argument area.

`void __builtin_return (`*void *result*`)` [Built-in Function]

This built-in function returns the value described by *result* from the containing function. You should specify, for *result*, a value returned by `__builtin_apply`.

`__builtin_va_arg_pack ()` [Built-in Function]

This built-in function represents all anonymous arguments of an inline function. It can be used only in inline functions which will be always inlined, never compiled as a separate function, such as those using `__attribute__ ((__always_inline__))` or `__attribute__ ((__gnu_inline__))` extern inline functions. It must be only passed as last argument to some other function with variable arguments. This is useful for writing small wrapper inlines for variable argument functions, when using preprocessor macros is undesirable. For example:

```
extern int myprintf (FILE *f, const char *format, ...);
extern inline __attribute__ ((__gnu_inline__)) int
myprintf (FILE *f, const char *format, ...)
{
  int r = fprintf (f, "myprintf: ");
  if (r < 0)
    return r;
  int s = fprintf (f, format, __builtin_va_arg_pack ());
  if (s < 0)
    return s;
  return r + s;
}
```

`size_t __builtin_va_arg_pack_len ()` [Built-in Function]

This built-in function returns the number of anonymous arguments of an inline function. It can be used only in inline functions which will be always inlined, never compiled as a separate function, such as those using `__attribute__ ((__always_inline__))` or `__attribute__ ((__gnu_inline__))` extern inline functions. For example following will do link or runtime checking of open arguments for optimized code:

```
#ifdef __OPTIMIZE__
extern inline __attribute__((__gnu_inline__)) int
```

```
myopen (const char *path, int oflag, ...)
{
  if (__builtin_va_arg_pack_len () > 1)
    warn_open_too_many_arguments ();

  if (__builtin_constant_p (oflag))
    {
      if ((oflag & O_CREAT) != 0 && __builtin_va_arg_pack_len () < 1)
        {
          warn_open_missing_mode ();
          return __open_2 (path, oflag);
        }
      return open (path, oflag, __builtin_va_arg_pack ());
    }

  if (__builtin_va_arg_pack_len () < 1)
    return __open_2 (path, oflag);

  return open (path, oflag, __builtin_va_arg_pack ());
}
#endif
```

6.6 Referring to a Type with `typeof`

Another way to refer to the type of an expression is with `typeof`. The syntax of using of this keyword looks like `sizeof`, but the construct acts semantically like a type name defined with `typedef`.

There are two ways of writing the argument to `typeof`: with an expression or with a type. Here is an example with an expression:

```
typeof (x[0](1))
```

This assumes that `x` is an array of pointers to functions; the type described is that of the values of the functions.

Here is an example with a typename as the argument:

```
typeof (int *)
```

Here the type described is that of pointers to `int`.

If you are writing a header file that must work when included in ISO C programs, write `__typeof__` instead of `typeof`. See Section 6.45 [Alternate Keywords], page 420.

A `typeof`-construct can be used anywhere a typedef name could be used. For example, you can use it in a declaration, in a cast, or inside of `sizeof` or `typeof`.

The operand of `typeof` is evaluated for its side effects if and only if it is an expression of variably modified type or the name of such a type.

`typeof` is often useful in conjunction with the statements-within-expressions feature. Here is how the two together can be used to define a safe "maximum" macro that operates on any arithmetic type and evaluates each of its arguments exactly once:

```
#define max(a,b) \
  ({ typeof (a) _a = (a); \
      typeof (b) _b = (b); \
    _a > _b ? _a : _b; })
```

The reason for using names that start with underscores for the local variables is to avoid conflicts with variable names that occur within the expressions that are substituted for `a`

and `b`. Eventually we hope to design a new form of declaration syntax that allows you to declare variables whose scopes start only after their initializers; this will be a more reliable way to prevent such conflicts.

Some more examples of the use of `typeof`:

- This declares `y` with the type of what `x` points to.

  ```
  typeof (*x) y;
  ```

- This declares `y` as an array of such values.

  ```
  typeof (*x) y[4];
  ```

- This declares `y` as an array of pointers to characters:

  ```
  typeof (typeof (char *)[4]) y;
  ```

  It is equivalent to the following traditional C declaration:

  ```
  char *y[4];
  ```

  To see the meaning of the declaration using `typeof`, and why it might be a useful way to write, rewrite it with these macros:

  ```
  #define pointer(T)  typeof(T *)
  #define array(T, N) typeof(T [N])
  ```

  Now the declaration can be rewritten this way:

  ```
  array (pointer (char), 4) y;
  ```

  Thus, `array (pointer (char), 4)` is the type of arrays of 4 pointers to `char`.

Compatibility Note: In addition to `typeof`, GCC 2 supported a more limited extension which permitted one to write

```
typedef T = expr;
```

with the effect of declaring *T* to have the type of the expression *expr*. This extension does not work with GCC 3 (versions between 3.0 and 3.2 will crash; 3.2.1 and later give an error). Code which relies on it should be rewritten to use `typeof`:

```
typedef typeof(expr) T;
```

This will work with all versions of GCC.

6.7 Conditionals with Omitted Operands

The middle operand in a conditional expression may be omitted. Then if the first operand is nonzero, its value is the value of the conditional expression.

Therefore, the expression

```
x ? : y
```

has the value of `x` if that is nonzero; otherwise, the value of `y`.

This example is perfectly equivalent to

```
x ? x : y
```

In this simple case, the ability to omit the middle operand is not especially useful. When it becomes useful is when the first operand does, or may (if it is a macro argument), contain a side effect. Then repeating the operand in the middle would perform the side effect twice. Omitting the middle operand uses the value already computed without the undesirable effects of recomputing it.

6.8 128-bits integers

As an extension the integer scalar type `__int128` is supported for targets having an integer mode wide enough to hold 128-bit. Simply write `__int128` for a signed 128-bit integer, or `unsigned __int128` for an unsigned 128-bit integer. There is no support in GCC to express an integer constant of type `__int128` for targets having `long long` integer with less then 128 bit width.

6.9 Double-Word Integers

ISO C99 supports data types for integers that are at least 64 bits wide, and as an extension GCC supports them in C90 mode and in C++. Simply write `long long int` for a signed integer, or `unsigned long long int` for an unsigned integer. To make an integer constant of type `long long int`, add the suffix 'LL' to the integer. To make an integer constant of type `unsigned long long int`, add the suffix 'ULL' to the integer.

You can use these types in arithmetic like any other integer types. Addition, subtraction, and bitwise boolean operations on these types are open-coded on all types of machines. Multiplication is open-coded if the machine supports fullword-to-doubleword a widening multiply instruction. Division and shifts are open-coded only on machines that provide special support. The operations that are not open-coded use special library routines that come with GCC.

There may be pitfalls when you use `long long` types for function arguments, unless you declare function prototypes. If a function expects type `int` for its argument, and you pass a value of type `long long int`, confusion will result because the caller and the subroutine will disagree about the number of bytes for the argument. Likewise, if the function expects `long long int` and you pass `int`. The best way to avoid such problems is to use prototypes.

6.10 Complex Numbers

ISO C99 supports complex floating data types, and as an extension GCC supports them in C90 mode and in C++, and supports complex integer data types which are not part of ISO C99. You can declare complex types using the keyword `_Complex`. As an extension, the older GNU keyword `__complex__` is also supported.

For example, '`_Complex double x;`' declares `x` as a variable whose real part and imaginary part are both of type `double`. '`_Complex short int y;`' declares `y` to have real and imaginary parts of type `short int`; this is not likely to be useful, but it shows that the set of complex types is complete.

To write a constant with a complex data type, use the suffix '`i`' or '`j`' (either one; they are equivalent). For example, `2.5fi` has type `_Complex float` and `3i` has type `_Complex int`. Such a constant always has a pure imaginary value, but you can form any complex value you like by adding one to a real constant. This is a GNU extension; if you have an ISO C99 conforming C library (such as GNU libc), and want to construct complex constants of floating type, you should include `<complex.h>` and use the macros `I` or `_Complex_I` instead.

To extract the real part of a complex-valued expression *exp*, write `__real__` *exp*. Likewise, use `__imag__` to extract the imaginary part. This is a GNU extension; for values of floating type, you should use the ISO C99 functions `crealf`, `creal`, `creall`, `cimagf`, `cimag` and `cimagl`, declared in `<complex.h>` and also provided as built-in functions by GCC.

The operator '~' performs complex conjugation when used on a value with a complex type. This is a GNU extension; for values of floating type, you should use the ISO C99 functions `conjf`, `conj` and `conjl`, declared in `<complex.h>` and also provided as built-in functions by GCC.

GCC can allocate complex automatic variables in a noncontiguous fashion; it's even possible for the real part to be in a register while the imaginary part is on the stack (or vice-versa). Only the DWARF2 debug info format can represent this, so use of DWARF2 is recommended. If you are using the stabs debug info format, GCC describes a noncontiguous complex variable as if it were two separate variables of noncomplex type. If the variable's actual name is `foo`, the two fictitious variables are named `foo$real` and `foo$imag`. You can examine and set these two fictitious variables with your debugger.

6.11 Additional Floating Types

As an extension, the GNU C compiler supports additional floating types, `__float80` and `__float128` to support 80bit (`XFmode`) and 128 bit (`TFmode`) floating types. Support for additional types includes the arithmetic operators: add, subtract, multiply, divide; unary arithmetic operators; relational operators; equality operators; and conversions to and from integer and other floating types. Use a suffix '`w`' or '`W`' in a literal constant of type `__float80` and '`q`' or '`Q`' for `_float128`. You can declare complex types using the corresponding internal complex type, `XCmode` for `__float80` type and `TCmode` for `__float128` type:

```
typedef _Complex float __attribute__((mode(TC))) _Complex128;
typedef _Complex float __attribute__((mode(XC))) _Complex80;
```

Not all targets support additional floating point types. `__float80` and `__float128` types are supported on i386, x86_64 and ia64 targets. The `__float128` type is supported on hppa HP-UX targets.

6.12 Half-Precision Floating Point

On ARM targets, GCC supports half-precision (16-bit) floating point via the `__fp16` type. You must enable this type explicitly with the '`-mfp16-format`' command-line option in order to use it.

ARM supports two incompatible representations for half-precision floating-point values. You must choose one of the representations and use it consistently in your program.

Specifying '`-mfp16-format=ieee`' selects the IEEE 754-2008 format. This format can represent normalized values in the range of 2^{-14} to 65504. There are 11 bits of significand precision, approximately 3 decimal digits.

Specifying '`-mfp16-format=alternative`' selects the ARM alternative format. This representation is similar to the IEEE format, but does not support infinities or NaNs. Instead, the range of exponents is extended, so that this format can represent normalized values in the range of 2^{-14} to 131008.

The `__fp16` type is a storage format only. For purposes of arithmetic and other operations, `__fp16` values in C or C++ expressions are automatically promoted to `float`. In addition, you cannot declare a function with a return value or parameters of type `__fp16`.

Note that conversions from `double` to `__fp16` involve an intermediate conversion to `float`. Because of rounding, this can sometimes produce a different result than a direct conversion.

ARM provides hardware support for conversions between `__fp16` and `float` values as an extension to VFP and NEON (Advanced SIMD). GCC generates code using these hardware instructions if you compile with options to select an FPU that provides them; for example, '`-mfpu=neon-fp16 -mfloat-abi=softfp`', in addition to the '`-mfp16-format`' option to select a half-precision format.

Language-level support for the `__fp16` data type is independent of whether GCC generates code using hardware floating-point instructions. In cases where hardware support is not specified, GCC implements conversions between `__fp16` and `float` values as library calls.

6.13 Decimal Floating Types

As an extension, the GNU C compiler supports decimal floating types as defined in the N1312 draft of ISO/IEC WDTR24732. Support for decimal floating types in GCC will evolve as the draft technical report changes. Calling conventions for any target might also change. Not all targets support decimal floating types.

The decimal floating types are `_Decimal32`, `_Decimal64`, and `_Decimal128`. They use a radix of ten, unlike the floating types `float`, `double`, and `long double` whose radix is not specified by the C standard but is usually two.

Support for decimal floating types includes the arithmetic operators add, subtract, multiply, divide; unary arithmetic operators; relational operators; equality operators; and conversions to and from integer and other floating types. Use a suffix '`df`' or '`DF`' in a literal constant of type `_Decimal32`, '`dd`' or '`DD`' for `_Decimal64`, and '`dl`' or '`DL`' for `_Decimal128`.

GCC support of decimal float as specified by the draft technical report is incomplete:

- When the value of a decimal floating type cannot be represented in the integer type to which it is being converted, the result is undefined rather than the result value specified by the draft technical report.
- GCC does not provide the C library functionality associated with '`math.h`', '`fenv.h`', '`stdio.h`', '`stdlib.h`', and '`wchar.h`', which must come from a separate C library implementation. Because of this the GNU C compiler does not define macro `__STDC_DEC_FP__` to indicate that the implementation conforms to the technical report.

Types `_Decimal32`, `_Decimal64`, and `_Decimal128` are supported by the DWARF2 debug information format.

6.14 Hex Floats

ISO C99 supports floating-point numbers written not only in the usual decimal notation, such as `1.55e1`, but also numbers such as `0x1.fp3` written in hexadecimal format. As a GNU extension, GCC supports this in C90 mode (except in some cases when strictly conforming) and in C++. In that format the '`0x`' hex introducer and the '`p`' or '`P`' exponent field are mandatory. The exponent is a decimal number that indicates the power of 2 by which the significant part will be multiplied. Thus '`0x1.f`' is $1\frac{15}{16}$, '`p3`' multiplies it by 8, and the value of `0x1.fp3` is the same as `1.55e1`.

Unlike for floating-point numbers in the decimal notation the exponent is always required in the hexadecimal notation. Otherwise the compiler would not be able to resolve the

ambiguity of, e.g., `0x1.f`. This could mean `1.0f` or `1.9375` since 'f' is also the extension for floating-point constants of type `float`.

6.15 Fixed-Point Types

As an extension, the GNU C compiler supports fixed-point types as defined in the N1169 draft of ISO/IEC DTR 18037. Support for fixed-point types in GCC will evolve as the draft technical report changes. Calling conventions for any target might also change. Not all targets support fixed-point types.

The fixed-point types are `short _Fract`, `_Fract`, `long _Fract`, `long long _Fract`, `unsigned short _Fract`, `unsigned _Fract`, `unsigned long _Fract`, `unsigned long long _Fract`, `_Sat short _Fract`, `_Sat _Fract`, `_Sat long _Fract`, `_Sat long long _Fract`, `_Sat unsigned short _Fract`, `_Sat unsigned _Fract`, `_Sat unsigned long _Fract`, `_Sat unsigned long long _Fract`, `short _Accum`, `_Accum`, `long _Accum`, `long long _Accum`, `unsigned short _Accum`, `unsigned _Accum`, `unsigned long _Accum`, `unsigned long long _Accum`, `_Sat short _Accum`, `_Sat _Accum`, `_Sat long _Accum`, `_Sat long long _Accum`, `_Sat unsigned short _Accum`, `_Sat unsigned _Accum`, `_Sat unsigned long _Accum`, `_Sat unsigned long long _Accum`.

Fixed-point data values contain fractional and optional integral parts. The format of fixed-point data varies and depends on the target machine.

Support for fixed-point types includes:

- prefix and postfix increment and decrement operators (`++`, `--`)
- unary arithmetic operators (`+`, `-`, `!`)
- binary arithmetic operators (`+`, `-`, `*`, `/`)
- binary shift operators (`<<`, `>>`)
- relational operators (`<`, `<=`, `>=`, `>`)
- equality operators (`==`, `!=`)
- assignment operators (`+=`, `-=`, `*=`, `/=`, `<<=`, `>>=`)
- conversions to and from integer, floating-point, or fixed-point types

Use a suffix in a fixed-point literal constant:

- '`hr`' or '`HR`' for `short _Fract` and `_Sat short _Fract`
- '`r`' or '`R`' for `_Fract` and `_Sat _Fract`
- '`lr`' or '`LR`' for `long _Fract` and `_Sat long _Fract`
- '`llr`' or '`LLR`' for `long long _Fract` and `_Sat long long _Fract`
- '`uhr`' or '`UHR`' for `unsigned short _Fract` and `_Sat unsigned short _Fract`
- '`ur`' or '`UR`' for `unsigned _Fract` and `_Sat unsigned _Fract`
- '`ulr`' or '`ULR`' for `unsigned long _Fract` and `_Sat unsigned long _Fract`
- '`ullr`' or '`ULLR`' for `unsigned long long _Fract` and `_Sat unsigned long long _Fract`
- '`hk`' or '`HK`' for `short _Accum` and `_Sat short _Accum`
- '`k`' or '`K`' for `_Accum` and `_Sat _Accum`
- '`lk`' or '`LK`' for `long _Accum` and `_Sat long _Accum`

- 'llk' or 'LLK' for long long _Accum and _Sat long long _Accum
- 'uhk' or 'UHK' for unsigned short _Accum and _Sat unsigned short _Accum
- 'uk' or 'UK' for unsigned _Accum and _Sat unsigned _Accum
- 'ulk' or 'ULK' for unsigned long _Accum and _Sat unsigned long _Accum
- 'ullk' or 'ULLK' for unsigned long long _Accum and _Sat unsigned long long _Accum

GCC support of fixed-point types as specified by the draft technical report is incomplete:

- Pragmas to control overflow and rounding behaviors are not implemented.

Fixed-point types are supported by the DWARF2 debug information format.

6.16 Named Address Spaces

As an extension, the GNU C compiler supports named address spaces as defined in the N1275 draft of ISO/IEC DTR 18037. Support for named address spaces in GCC will evolve as the draft technical report changes. Calling conventions for any target might also change. At present, only the AVR, SPU, M32C, and RL78 targets support address spaces other than the generic address space.

Address space identifiers may be used exactly like any other C type qualifier (e.g., const or volatile). See the N1275 document for more details.

6.16.1 AVR Named Address Spaces

On the AVR target, there are several address spaces that can be used in order to put read-only data into the flash memory and access that data by means of the special instructions LPM or ELPM needed to read from flash.

Per default, any data including read-only data is located in RAM (the generic address space) so that non-generic address spaces are needed to locate read-only data in flash memory *and* to generate the right instructions to access this data without using (inline) assembler code.

__flash
: The __flash qualifier will locate data in the .progmem.data section. Data will be read using the LPM instruction. Pointers to this address space are 16 bits wide.

__flash1

__flash2

__flash3

__flash4

__flash5
: These are 16-bit address spaces locating data in section .progmem*N*.data where *N* refers to address space __flash*N*. The compiler will set the RAMPZ segment register appropriately before reading data by means of the ELPM instruction.

__memx
: This is a 24-bit address space that linearizes flash and RAM: If the high bit of the address is set, data is read from RAM using the lower two bytes as RAM address. If the high bit of the address is clear, data is read from flash with RAMPZ set according to the high byte of the address.

  Objects in this address space will be located in .progmem.data.

Example

```
char my_read (const __flash char ** p)
{
    /* p is a pointer to RAM that points to a pointer to flash.
       The first indirection of p will read that flash pointer
       from RAM and the second indirection reads a char from this
       flash address.  */

    return **p;
}

/* Locate array[] in flash memory */
const __flash int array[] = { 3, 5, 7, 11, 13, 17, 19 };

int i = 1;

int main (void)
{
   /* Return 17 by reading from flash memory */
   return array[array[i]];
}
```

For each named address space supported by avr-gcc there is an equally named but uppercase built-in macro defined. The purpose is to facilitate testing if respective address space support is available or not:

```
#ifdef __FLASH
const __flash int var = 1;

int read_var (void)
{
    return var;
}
#else
#include <avr/pgmspace.h> /* From AVR-LibC */

const int var PROGMEM = 1;

int read_var (void)
{
    return (int) pgm_read_word (&var);
}
#endif /* __FLASH */
```

Notice that attribute [AVR Variable Attributes], page 370 locates data in flash but accesses to these data will read from generic address space, i.e. from RAM, so that you need special accessors like `pgm_read_byte` from AVR-LibC (`http://nongnu.org/avr-libc/user-manual`) together with attribute `progmem`.

Limitations and caveats

- Reading across the 64 KiB section boundary of the `__flash` or `__flashN` address spaces will show undefined behaviour. The only address space that supports reading across the 64 KiB flash segment boundaries is `__memx`.
- If you use one of the `__flashN` address spaces you will have to arrange your linker skript to locate the `.progmemN.data` sections according to your needs.
- Any data or pointers to the non-generic address spaces must be qualified as `const`, i.e. as read-only data. This still applies if the data in one of these address spaces like software version number or calibration lookup table are intended to be changed after load time by, say, a boot loader. In this case the right qualification is `const volatile` so that the compiler must not optimize away known values or insert them as immediates into operands of instructions.
- Code like the following is not yet supported because of missing support in avr-binutils, see PR13503 (`http://sourceware.org/PR13503`).

  ```
  extern const __memx char foo;
  const __memx void *pfoo = &foo;
  ```

  The code will throw an assembler warning and the high byte of `pfoo` will be initialized with `0`, i.e. the initialization will be as if `foo` was located in the first 64 KiB chunk of flash.

6.16.2 M32C Named Address Spaces

On the M32C target, with the R8C and M16C cpu variants, variables qualified with `__far` are accessed using 32-bit addresses in order to access memory beyond the first 64 Ki bytes. If `__far` is used with the M32CM or M32C cpu variants, it has no effect.

6.16.3 RL78 Named Address Spaces

On the RL78 target, variables qualified with `__far` are accessed with 32-bit pointers (20-bit addresses) rather than the default 16-bit addresses. Non-far variables are assumed to appear in the topmost 64 KiB of the address space.

6.16.4 SPU Named Address Spaces

On the SPU target variables may be declared as belonging to another address space by qualifying the type with the `__ea` address space identifier:

```
extern int __ea i;
```

When the variable `i` is accessed, the compiler will generate special code to access this variable. It may use runtime library support, or generate special machine instructions to access that address space.

6.17 Arrays of Length Zero

Zero-length arrays are allowed in GNU C. They are very useful as the last element of a structure which is really a header for a variable-length object:

```
struct line {
  int length;
  char contents[0];
};
```

```
struct line *thisline = (struct line *)
  malloc (sizeof (struct line) + this_length);
thisline->length = this_length;
```

In ISO C90, you would have to give `contents` a length of 1, which means either you waste space or complicate the argument to `malloc`.

In ISO C99, you would use a *flexible array member*, which is slightly different in syntax and semantics:

- Flexible array members are written as `contents[]` without the `0`.
- Flexible array members have incomplete type, and so the `sizeof` operator may not be applied. As a quirk of the original implementation of zero-length arrays, `sizeof` evaluates to zero.
- Flexible array members may only appear as the last member of a `struct` that is otherwise non-empty.
- A structure containing a flexible array member, or a union containing such a structure (possibly recursively), may not be a member of a structure or an element of an array. (However, these uses are permitted by GCC as extensions.)

GCC versions before 3.0 allowed zero-length arrays to be statically initialized, as if they were flexible arrays. In addition to those cases that were useful, it also allowed initializations in situations that would corrupt later data. Non-empty initialization of zero-length arrays is now treated like any case where there are more initializer elements than the array holds, in that a suitable warning about "excess elements in array" is given, and the excess elements (all of them, in this case) are ignored.

Instead GCC allows static initialization of flexible array members. This is equivalent to defining a new structure containing the original structure followed by an array of sufficient size to contain the data. I.e. in the following, `f1` is constructed as if it were declared like `f2`.

```
struct f1 {
  int x; int y[];
} f1 = { 1, { 2, 3, 4 } };

struct f2 {
  struct f1 f1; int data[3];
} f2 = { { 1 }, { 2, 3, 4 } };
```

The convenience of this extension is that `f1` has the desired type, eliminating the need to consistently refer to `f2.f1`.

This has symmetry with normal static arrays, in that an array of unknown size is also written with `[]`.

Of course, this extension only makes sense if the extra data comes at the end of a top-level object, as otherwise we would be overwriting data at subsequent offsets. To avoid undue complication and confusion with initialization of deeply nested arrays, we simply disallow any non-empty initialization except when the structure is the top-level object. For example:

```
struct foo { int x; int y[]; };
struct bar { struct foo z; };

struct foo a = { 1, { 2, 3, 4 } };        // Valid.
struct bar b = { { 1, { 2, 3, 4 } } };    // Invalid.
```

```
struct bar c = { { 1, { } } };            // Valid.
struct foo d[1] = { { 1 { 2, 3, 4 } } };  // Invalid.
```

6.18 Structures With No Members

GCC permits a C structure to have no members:

```
struct empty {
};
```

The structure will have size zero. In C++, empty structures are part of the language. G++ treats empty structures as if they had a single member of type `char`.

6.19 Arrays of Variable Length

Variable-length automatic arrays are allowed in ISO C99, and as an extension GCC accepts them in C90 mode and in C++. These arrays are declared like any other automatic arrays, but with a length that is not a constant expression. The storage is allocated at the point of declaration and deallocated when the brace-level is exited. For example:

```
FILE *
concat_fopen (char *s1, char *s2, char *mode)
{
  char str[strlen (s1) + strlen (s2) + 1];
  strcpy (str, s1);
  strcat (str, s2);
  return fopen (str, mode);
}
```

Jumping or breaking out of the scope of the array name deallocates the storage. Jumping into the scope is not allowed; you get an error message for it.

You can use the function `alloca` to get an effect much like variable-length arrays. The function `alloca` is available in many other C implementations (but not in all). On the other hand, variable-length arrays are more elegant.

There are other differences between these two methods. Space allocated with `alloca` exists until the containing *function* returns. The space for a variable-length array is deallocated as soon as the array name's scope ends. (If you use both variable-length arrays and `alloca` in the same function, deallocation of a variable-length array will also deallocate anything more recently allocated with `alloca`.)

You can also use variable-length arrays as arguments to functions:

```
struct entry
tester (int len, char data[len][len])
{
  /* ... */
}
```

The length of an array is computed once when the storage is allocated and is remembered for the scope of the array in case you access it with `sizeof`.

If you want to pass the array first and the length afterward, you can use a forward declaration in the parameter list—another GNU extension.

```
struct entry
tester (int len; char data[len][len], int len)
{
  /* ... */
```

```
}
```

The '`int len`' before the semicolon is a *parameter forward declaration*, and it serves the purpose of making the name `len` known when the declaration of `data` is parsed.

You can write any number of such parameter forward declarations in the parameter list. They can be separated by commas or semicolons, but the last one must end with a semicolon, which is followed by the "real" parameter declarations. Each forward declaration must match a "real" declaration in parameter name and data type. ISO C99 does not support parameter forward declarations.

6.20 Macros with a Variable Number of Arguments.

In the ISO C standard of 1999, a macro can be declared to accept a variable number of arguments much as a function can. The syntax for defining the macro is similar to that of a function. Here is an example:

```
#define debug(format, ...) fprintf (stderr, format, __VA_ARGS__)
```

Here '`...`' is a *variable argument*. In the invocation of such a macro, it represents the zero or more tokens until the closing parenthesis that ends the invocation, including any commas. This set of tokens replaces the identifier `__VA_ARGS__` in the macro body wherever it appears. See the CPP manual for more information.

GCC has long supported variadic macros, and used a different syntax that allowed you to give a name to the variable arguments just like any other argument. Here is an example:

```
#define debug(format, args...) fprintf (stderr, format, args)
```

This is in all ways equivalent to the ISO C example above, but arguably more readable and descriptive.

GNU CPP has two further variadic macro extensions, and permits them to be used with either of the above forms of macro definition.

In standard C, you are not allowed to leave the variable argument out entirely; but you are allowed to pass an empty argument. For example, this invocation is invalid in ISO C, because there is no comma after the string:

```
debug ("A message")
```

GNU CPP permits you to completely omit the variable arguments in this way. In the above examples, the compiler would complain, though since the expansion of the macro still has the extra comma after the format string.

To help solve this problem, CPP behaves specially for variable arguments used with the token paste operator, '`##`'. If instead you write

```
#define debug(format, ...) fprintf (stderr, format, ## __VA_ARGS__)
```

and if the variable arguments are omitted or empty, the '`##`' operator causes the preprocessor to remove the comma before it. If you do provide some variable arguments in your macro invocation, GNU CPP does not complain about the paste operation and instead places the variable arguments after the comma. Just like any other pasted macro argument, these arguments are not macro expanded.

6.21 Slightly Looser Rules for Escaped Newlines

Recently, the preprocessor has relaxed its treatment of escaped newlines. Previously, the newline had to immediately follow a backslash. The current implementation allows whitespace in the form of spaces, horizontal and vertical tabs, and form feeds between the backslash and the subsequent newline. The preprocessor issues a warning, but treats it as a valid escaped newline and combines the two lines to form a single logical line. This works within comments and tokens, as well as between tokens. Comments are *not* treated as whitespace for the purposes of this relaxation, since they have not yet been replaced with spaces.

6.22 Non-Lvalue Arrays May Have Subscripts

In ISO C99, arrays that are not lvalues still decay to pointers, and may be subscripted, although they may not be modified or used after the next sequence point and the unary '`&`' operator may not be applied to them. As an extension, GCC allows such arrays to be subscripted in C90 mode, though otherwise they do not decay to pointers outside C99 mode. For example, this is valid in GNU C though not valid in C90:

```
struct foo {int a[4];};

struct foo f();

bar (int index)
{
  return f().a[index];
}
```

6.23 Arithmetic on `void`- and Function-Pointers

In GNU C, addition and subtraction operations are supported on pointers to `void` and on pointers to functions. This is done by treating the size of a `void` or of a function as 1.

A consequence of this is that `sizeof` is also allowed on `void` and on function types, and returns 1.

The option '`-Wpointer-arith`' requests a warning if these extensions are used.

6.24 Non-Constant Initializers

As in standard C++ and ISO C99, the elements of an aggregate initializer for an automatic variable are not required to be constant expressions in GNU C. Here is an example of an initializer with run-time varying elements:

```
foo (float f, float g)
{
  float beat_freqs[2] = { f-g, f+g };
  /* ... */
}
```

6.25 Compound Literals

ISO C99 supports compound literals. A compound literal looks like a cast containing an initializer. Its value is an object of the type specified in the cast, containing the elements specified in the initializer; it is an lvalue. As an extension, GCC supports compound literals in C90 mode and in C++, though the semantics are somewhat different in C++.

Usually, the specified type is a structure. Assume that `struct foo` and `structure` are declared as shown:

```
struct foo {int a; char b[2];} structure;
```

Here is an example of constructing a `struct foo` with a compound literal:

```
structure = ((struct foo) {x + y, 'a', 0});
```

This is equivalent to writing the following:

```
{
  struct foo temp = {x + y, 'a', 0};
  structure = temp;
}
```

You can also construct an array, though this is dangerous in C++, as explained below. If all the elements of the compound literal are (made up of) simple constant expressions, suitable for use in initializers of objects of static storage duration, then the compound literal can be coerced to a pointer to its first element and used in such an initializer, as shown here:

```
char **foo = (char *[]) { "x", "y", "z" };
```

Compound literals for scalar types and union types are also allowed, but then the compound literal is equivalent to a cast.

As a GNU extension, GCC allows initialization of objects with static storage duration by compound literals (which is not possible in ISO C99, because the initializer is not a constant). It is handled as if the object was initialized only with the bracket enclosed list if the types of the compound literal and the object match. The initializer list of the compound literal must be constant. If the object being initialized has array type of unknown size, the size is determined by compound literal size.

```
static struct foo x = (struct foo) {1, 'a', 'b'};
static int y[] = (int []) {1, 2, 3};
static int z[] = (int [3]) {1};
```

The above lines are equivalent to the following:

```
static struct foo x = {1, 'a', 'b'};
static int y[] = {1, 2, 3};
static int z[] = {1, 0, 0};
```

In C, a compound literal designates an unnamed object with static or automatic storage duration. In C++, a compound literal designates a temporary object, which only lives until the end of its full-expression. As a result, well-defined C code that takes the address of a subobject of a compound literal can be undefined in C++. For instance, if the array compound literal example above appeared inside a function, any subsequent use of '`foo`' in C++ has undefined behavior because the lifetime of the array ends after the declaration of '`foo`'. As a result, the C++ compiler now rejects the conversion of a temporary array to a pointer.

As an optimization, the C++ compiler sometimes gives array compound literals longer lifetimes: when the array either appears outside a function or has const-qualified type. If '`foo`' and its initializer had elements of '`char *const`' type rather than '`char *`', or if '`foo`' were a global variable, the array would have static storage duration. But it is probably safest just to avoid the use of array compound literals in code compiled as C++.

6.26 Designated Initializers

Standard C90 requires the elements of an initializer to appear in a fixed order, the same as the order of the elements in the array or structure being initialized.

In ISO C99 you can give the elements in any order, specifying the array indices or structure field names they apply to, and GNU C allows this as an extension in C90 mode as well. This extension is not implemented in GNU C++.

To specify an array index, write '`[index] =`' before the element value. For example,

```
int a[6] = { [4] = 29, [2] = 15 };
```

is equivalent to

```
int a[6] = { 0, 0, 15, 0, 29, 0 };
```

The index values must be constant expressions, even if the array being initialized is automatic.

An alternative syntax for this which has been obsolete since GCC 2.5 but GCC still accepts is to write '`[index]`' before the element value, with no '`=`'.

To initialize a range of elements to the same value, write '`[first ... last] = value`'. This is a GNU extension. For example,

```
int widths[] = { [0 ... 9] = 1, [10 ... 99] = 2, [100] = 3 };
```

If the value in it has side-effects, the side-effects will happen only once, not for each initialized field by the range initializer.

Note that the length of the array is the highest value specified plus one.

In a structure initializer, specify the name of a field to initialize with '`.fieldname =`' before the element value. For example, given the following structure,

```
struct point { int x, y; };
```

the following initialization

```
struct point p = { .y = yvalue, .x = xvalue };
```

is equivalent to

```
struct point p = { xvalue, yvalue };
```

Another syntax which has the same meaning, obsolete since GCC 2.5, is '`fieldname:`', as shown here:

```
struct point p = { y: yvalue, x: xvalue };
```

The '`[index]`' or '`.fieldname`' is known as a *designator*. You can also use a designator (or the obsolete colon syntax) when initializing a union, to specify which element of the union should be used. For example,

```
union foo { int i; double d; };

union foo f = { .d = 4 };
```

will convert 4 to a `double` to store it in the union using the second element. By contrast, casting 4 to type `union foo` would store it into the union as the integer `i`, since it is an integer. (See Section 6.28 [Cast to Union], page 331.)

You can combine this technique of naming elements with ordinary C initialization of successive elements. Each initializer element that does not have a designator applies to the next consecutive element of the array or structure. For example,

```
int a[6] = { [1] = v1, v2, [4] = v4 };
```

is equivalent to

```
int a[6] = { 0, v1, v2, 0, v4, 0 };
```

Labeling the elements of an array initializer is especially useful when the indices are characters or belong to an `enum` type. For example:

```
int whitespace[256]
  = { [' '] = 1, ['\t'] = 1, ['\h'] = 1,
      ['\f'] = 1, ['\n'] = 1, ['\r'] = 1 };
```

You can also write a series of '`.`*`fieldname`*' and '`[`*`index`*`]`' designators before an '`=`' to specify a nested subobject to initialize; the list is taken relative to the subobject corresponding to the closest surrounding brace pair. For example, with the '`struct point`' declaration above:

```
struct point ptarray[10] = { [2].y = yv2, [2].x = xv2, [0].x = xv0 };
```

If the same field is initialized multiple times, it will have value from the last initialization. If any such overridden initialization has side-effect, it is unspecified whether the side-effect happens or not. Currently, GCC will discard them and issue a warning.

6.27 Case Ranges

You can specify a range of consecutive values in a single `case` label, like this:

```
case low ... high:
```

This has the same effect as the proper number of individual `case` labels, one for each integer value from *low* to *high*, inclusive.

This feature is especially useful for ranges of ASCII character codes:

```
case 'A' ... 'Z':
```

Be careful: Write spaces around the `...`, for otherwise it may be parsed wrong when you use it with integer values. For example, write this:

```
case 1 ... 5:
```

rather than this:

```
case 1...5:
```

6.28 Cast to a Union Type

A cast to union type is similar to other casts, except that the type specified is a union type. You can specify the type either with `union` *`tag`* or with a typedef name. A cast to union is actually a constructor though, not a cast, and hence does not yield an lvalue like normal casts. (See Section 6.25 [Compound Literals], page 328.)

The types that may be cast to the union type are those of the members of the union. Thus, given the following union and variables:

```
union foo { int i; double d; };
int x;
double y;
```

both `x` and `y` can be cast to type `union foo`.

Using the cast as the right-hand side of an assignment to a variable of union type is equivalent to storing in a member of the union:

```
union foo u;
/* ... */
u = (union foo) x  ≡  u.i = x
u = (union foo) y  ≡  u.d = y
```

You can also use the union cast as a function argument:

```
void hack (union foo);
/* ... */
hack ((union foo) x);
```

6.29 Mixed Declarations and Code

ISO C99 and ISO C++ allow declarations and code to be freely mixed within compound statements. As an extension, GCC also allows this in C90 mode. For example, you could do:

```
int i;
/* ... */
i++;
int j = i + 2;
```

Each identifier is visible from where it is declared until the end of the enclosing block.

6.30 Declaring Attributes of Functions

In GNU C, you declare certain things about functions called in your program which help the compiler optimize function calls and check your code more carefully.

The keyword `__attribute__` allows you to specify special attributes when making a declaration. This keyword is followed by an attribute specification inside double parentheses. The following attributes are currently defined for functions on all targets: `aligned`, `alloc_size`, `noreturn`, `returns_twice`, `noinline`, `noclone`, `always_inline`, `flatten`, `pure`, `const`, `nothrow`, `sentinel`, `format`, `format_arg`, `no_instrument_function`, `no_split_stack`, `section`, `constructor`, `destructor`, `used`, `unused`, `deprecated`, `weak`, `malloc`, `alias`, `ifunc`, `warn_unused_result`, `nonnull`, `gnu_inline`, `externally_visible`, `hot`, `cold`, `artificial`, `error` and `warning`. Several other attributes are defined for functions on particular target systems. Other attributes, including `section` are supported for variables declarations (see Section 6.36 [Variable Attributes], page 365) and for types (see Section 6.37 [Type Attributes], page 374).

GCC plugins may provide their own attributes.

You may also specify attributes with '`__`' preceding and following each keyword. This allows you to use them in header files without being concerned about a possible macro of the same name. For example, you may use `__noreturn__` instead of `noreturn`.

See Section 6.31 [Attribute Syntax], page 361, for details of the exact syntax for using attributes.

`alias ("`*target*`")`

The `alias` attribute causes the declaration to be emitted as an alias for another symbol, which must be specified. For instance,

```
void __f () { /* Do something. */; }
void f () __attribute__ ((weak, alias ("__f")));
```

defines '`f`' to be a weak alias for '`__f`'. In C++, the mangled name for the target must be used. It is an error if '`__f`' is not defined in the same translation unit.

Not all target machines support this attribute.

`aligned (`*`alignment`*`)`
: This attribute specifies a minimum alignment for the function, measured in bytes.

  You cannot use this attribute to decrease the alignment of a function, only to increase it. However, when you explicitly specify a function alignment this will override the effect of the '`-falign-functions`' (see Section 3.10 [Optimize Options], page 92) option for this function.

  Note that the effectiveness of `aligned` attributes may be limited by inherent limitations in your linker. On many systems, the linker is only able to arrange for functions to be aligned up to a certain maximum alignment. (For some linkers, the maximum supported alignment may be very very small.) See your linker documentation for further information.

  The `aligned` attribute can also be used for variables and fields (see Section 6.36 [Variable Attributes], page 365.)

`alloc_size`
: The `alloc_size` attribute is used to tell the compiler that the function return value points to memory, where the size is given by one or two of the functions parameters. GCC uses this information to improve the correctness of `__builtin_object_size`.

  The function parameter(s) denoting the allocated size are specified by one or two integer arguments supplied to the attribute. The allocated size is either the value of the single function argument specified or the product of the two function arguments specified. Argument numbering starts at one.

  For instance,

  ```
  void* my_calloc(size_t, size_t) __attribute__((alloc_size(1,2)))
  void my_realloc(void*, size_t) __attribute__((alloc_size(2)))
  ```

  declares that my_calloc will return memory of the size given by the product of parameter 1 and 2 and that my_realloc will return memory of the size given by parameter 2.

`always_inline`
: Generally, functions are not inlined unless optimization is specified. For functions declared inline, this attribute inlines the function even if no optimization level was specified.

`gnu_inline`
: This attribute should be used with a function which is also declared with the `inline` keyword. It directs GCC to treat the function as if it were defined in gnu90 mode even when compiling in C99 or gnu99 mode.

  If the function is declared `extern`, then this definition of the function is used only for inlining. In no case is the function compiled as a standalone function, not even if you take its address explicitly. Such an address becomes an external reference, as if you had only declared the function, and had not defined it. This has almost the effect of a macro. The way to use this is to put a function definition in a header file with this attribute, and put another copy of the

function, without `extern`, in a library file. The definition in the header file will cause most calls to the function to be inlined. If any uses of the function remain, they will refer to the single copy in the library. Note that the two definitions of the functions need not be precisely the same, although if they do not have the same effect your program may behave oddly.

In C, if the function is neither `extern` nor `static`, then the function is compiled as a standalone function, as well as being inlined where possible.

This is how GCC traditionally handled functions declared `inline`. Since ISO C99 specifies a different semantics for `inline`, this function attribute is provided as a transition measure and as a useful feature in its own right. This attribute is available in GCC 4.1.3 and later. It is available if either of the preprocessor macros `__GNUC_GNU_INLINE__` or `__GNUC_STDC_INLINE__` are defined. See Section 6.39 [An Inline Function is As Fast As a Macro], page 380.

In C++, this attribute does not depend on `extern` in any way, but it still requires the `inline` keyword to enable its special behavior.

`artificial`
: This attribute is useful for small inline wrappers which if possible should appear during debugging as a unit, depending on the debug info format it will either mean marking the function as artificial or using the caller location for all instructions within the inlined body.

`bank_switch`
: When added to an interrupt handler with the M32C port, causes the prologue and epilogue to use bank switching to preserve the registers rather than saving them on the stack.

`flatten`
: Generally, inlining into a function is limited. For a function marked with this attribute, every call inside this function will be inlined, if possible. Whether the function itself is considered for inlining depends on its size and the current inlining parameters.

`error ("`*message*`")`
: If this attribute is used on a function declaration and a call to such a function is not eliminated through dead code elimination or other optimizations, an error which will include *message* will be diagnosed. This is useful for compile time checking, especially together with `__builtin_constant_p` and inline functions where checking the inline function arguments is not possible through `extern char [(condition) ? 1 : -1];` tricks. While it is possible to leave the function undefined and thus invoke a link failure, when using this attribute the problem will be diagnosed earlier and with exact location of the call even in presence of inline functions or when not emitting debugging information.

`warning ("`*message*`")`
: If this attribute is used on a function declaration and a call to such a function is not eliminated through dead code elimination or other optimizations, a warning which will include *message* will be diagnosed. This is useful for compile time checking, especially together with `__builtin_constant_p` and inline functions. While it is possible to define the function with a message in `.gnu.warning*`

section, when using this attribute the problem will be diagnosed earlier and with exact location of the call even in presence of inline functions or when not emitting debugging information.

`cdecl` On the Intel 386, the `cdecl` attribute causes the compiler to assume that the calling function will pop off the stack space used to pass arguments. This is useful to override the effects of the '`-mrtd`' switch.

`const` Many functions do not examine any values except their arguments, and have no effects except the return value. Basically this is just slightly more strict class than the `pure` attribute below, since function is not allowed to read global memory.

Note that a function that has pointer arguments and examines the data pointed to must *not* be declared `const`. Likewise, a function that calls a non-`const` function usually must not be `const`. It does not make sense for a `const` function to return `void`.

The attribute `const` is not implemented in GCC versions earlier than 2.5. An alternative way to declare that a function has no side effects, which works in the current version and in some older versions, is as follows:

```
typedef int intfn ();

extern const intfn square;
```

This approach does not work in GNU C++ from 2.6.0 on, since the language specifies that the '`const`' must be attached to the return value.

`constructor`
`destructor`
`constructor (`*priority*`)`
`destructor (`*priority*`)`

The `constructor` attribute causes the function to be called automatically before execution enters `main ()`. Similarly, the `destructor` attribute causes the function to be called automatically after `main ()` has completed or `exit ()` has been called. Functions with these attributes are useful for initializing data that will be used implicitly during the execution of the program.

You may provide an optional integer priority to control the order in which constructor and destructor functions are run. A constructor with a smaller priority number runs before a constructor with a larger priority number; the opposite relationship holds for destructors. So, if you have a constructor that allocates a resource and a destructor that deallocates the same resource, both functions typically have the same priority. The priorities for constructor and destructor functions are the same as those specified for namespace-scope C++ objects (see Section 7.7 [C++ Attributes], page 629).

These attributes are not currently implemented for Objective-C.

`deprecated`
`deprecated (`*msg*`)`

The `deprecated` attribute results in a warning if the function is used anywhere in the source file. This is useful when identifying functions that are expected

to be removed in a future version of a program. The warning also includes the location of the declaration of the deprecated function, to enable users to easily find further information about why the function is deprecated, or what they should do instead. Note that the warnings only occurs for uses:

```
int old_fn () __attribute__ ((deprecated));
int old_fn ();
int (*fn_ptr)() = old_fn;
```

results in a warning on line 3 but not line 2. The optional msg argument, which must be a string, will be printed in the warning if present.

The `deprecated` attribute can also be used for variables and types (see Section 6.36 [Variable Attributes], page 365, see Section 6.37 [Type Attributes], page 374.)

`disinterrupt`

On Epiphany and MeP targets, this attribute causes the compiler to emit instructions to disable interrupts for the duration of the given function.

`dllexport`

On Microsoft Windows targets and Symbian OS targets the `dllexport` attribute causes the compiler to provide a global pointer to a pointer in a DLL, so that it can be referenced with the `dllimport` attribute. On Microsoft Windows targets, the pointer name is formed by combining `_imp__` and the function or variable name.

You can use `__declspec(dllexport)` as a synonym for `__attribute__ ((dllexport))` for compatibility with other compilers.

On systems that support the `visibility` attribute, this attribute also implies "default" visibility. It is an error to explicitly specify any other visibility.

In previous versions of GCC, the `dllexport` attribute was ignored for inlined functions, unless the '`-fkeep-inline-functions`' flag had been used. The default behaviour now is to emit all dllexported inline functions; however, this can cause object file-size bloat, in which case the old behaviour can be restored by using '`-fno-keep-inline-dllexport`'.

The attribute is also ignored for undefined symbols.

When applied to C++ classes, the attribute marks defined non-inlined member functions and static data members as exports. Static consts initialized in-class are not marked unless they are also defined out-of-class.

For Microsoft Windows targets there are alternative methods for including the symbol in the DLL's export table such as using a '`.def`' file with an `EXPORTS` section or, with GNU ld, using the '`--export-all`' linker flag.

`dllimport`

On Microsoft Windows and Symbian OS targets, the `dllimport` attribute causes the compiler to reference a function or variable via a global pointer to a pointer that is set up by the DLL exporting the symbol. The attribute implies `extern`. On Microsoft Windows targets, the pointer name is formed by combining `_imp__` and the function or variable name.

You can use `__declspec(dllimport)` as a synonym for `__attribute__ ((dllimport))` for compatibility with other compilers.

On systems that support the `visibility` attribute, this attribute also implies "default" visibility. It is an error to explicitly specify any other visibility.

Currently, the attribute is ignored for inlined functions. If the attribute is applied to a symbol *definition*, an error is reported. If a symbol previously declared `dllimport` is later defined, the attribute is ignored in subsequent references, and a warning is emitted. The attribute is also overridden by a subsequent declaration as `dllexport`.

When applied to C++ classes, the attribute marks non-inlined member functions and static data members as imports. However, the attribute is ignored for virtual methods to allow creation of vtables using thunks.

On the SH Symbian OS target the `dllimport` attribute also has another affect—it can cause the vtable and run-time type information for a class to be exported. This happens when the class has a dllimport'ed constructor or a non-inline, non-pure virtual function and, for either of those two conditions, the class also has an inline constructor or destructor and has a key function that is defined in the current translation unit.

For Microsoft Windows based targets the use of the `dllimport` attribute on functions is not necessary, but provides a small performance benefit by eliminating a thunk in the DLL. The use of the `dllimport` attribute on imported variables was required on older versions of the GNU linker, but can now be avoided by passing the '`--enable-auto-import`' switch to the GNU linker. As with functions, using the attribute for a variable eliminates a thunk in the DLL.

One drawback to using this attribute is that a pointer to a *variable* marked as `dllimport` cannot be used as a constant address. However, a pointer to a *function* with the `dllimport` attribute can be used as a constant initializer; in this case, the address of a stub function in the import lib is referenced. On Microsoft Windows targets, the attribute can be disabled for functions by setting the '`-mnop-fun-dllimport`' flag.

`eightbit_data`

Use this attribute on the H8/300, H8/300H, and H8S to indicate that the specified variable should be placed into the eight bit data section. The compiler will generate more efficient code for certain operations on data in the eight bit data area. Note the eight bit data area is limited to 256 bytes of data.

You must use GAS and GLD from GNU binutils version 2.7 or later for this attribute to work correctly.

`exception_handler`

Use this attribute on the Blackfin to indicate that the specified function is an exception handler. The compiler will generate function entry and exit sequences suitable for use in an exception handler when this attribute is present.

`externally_visible`

This attribute, attached to a global variable or function, nullifies the effect of the '`-fwhole-program`' command-line option, so the object remains visible outside the current compilation unit. If '`-fwhole-program`' is used together with '`-flto`' and `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.

`far` On 68HC11 and 68HC12 the `far` attribute causes the compiler to use a calling convention that takes care of switching memory banks when entering and leaving a function. This calling convention is also the default when using the '`-mlong-calls`' option.

On 68HC12 the compiler will use the `call` and `rtc` instructions to call and return from a function.

On 68HC11 the compiler will generate a sequence of instructions to invoke a board-specific routine to switch the memory bank and call the real function. The board-specific routine simulates a `call`. At the end of a function, it will jump to a board-specific routine instead of using `rts`. The board-specific return routine simulates the `rtc`.

On MeP targets this causes the compiler to use a calling convention which assumes the called function is too far away for the built-in addressing modes.

`fast_interrupt`
Use this attribute on the M32C and RX ports to indicate that the specified function is a fast interrupt handler. This is just like the `interrupt` attribute, except that `freit` is used to return instead of `reit`.

`fastcall` On the Intel 386, the `fastcall` attribute causes the compiler to pass the first argument (if of integral type) in the register ECX and the second argument (if of integral type) in the register EDX. Subsequent and other typed arguments are passed on the stack. The called function will pop the arguments off the stack. If the number of arguments is variable all arguments are pushed on the stack.

`thiscall` On the Intel 386, the `thiscall` attribute causes the compiler to pass the first argument (if of integral type) in the register ECX. Subsequent and other typed arguments are passed on the stack. The called function will pop the arguments off the stack. If the number of arguments is variable all arguments are pushed on the stack. The `thiscall` attribute is intended for C++ non-static member functions. As gcc extension this calling convention can be used for C-functions and for static member methods.

`format (`*`archetype`*`, `*`string-index`*`, `*`first-to-check`*`)`
The `format` attribute specifies that a function takes `printf`, `scanf`, `strftime` or `strfmon` style arguments which should be type-checked against a format string. For example, the declaration:

```
extern int
my_printf (void *my_object, const char *my_format, ...)
      __attribute__ ((format (printf, 2, 3)));
```

causes the compiler to check the arguments in calls to `my_printf` for consistency with the `printf` style format string argument `my_format`.

The parameter *archetype* determines how the format string is interpreted, and should be `printf`, `scanf`, `strftime`, `gnu_printf`, `gnu_scanf`, `gnu_strftime`

or `strfmon`. (You can also use `__printf__`, `__scanf__`, `__strftime__` or `__strfmon__`.) On MinGW targets, `ms_printf`, `ms_scanf`, and `ms_strftime` are also present. *archtype* values such as `printf` refer to the formats accepted by the system's C run-time library, while `gnu_` values always refer to the formats accepted by the GNU C Library. On Microsoft Windows targets, `ms_` values refer to the formats accepted by the '`msvcrt.dll`' library. The parameter *string-index* specifies which argument is the format string argument (starting from 1), while *first-to-check* is the number of the first argument to check against the format string. For functions where the arguments are not available to be checked (such as `vprintf`), specify the third parameter as zero. In this case the compiler only checks the format string for consistency. For `strftime` formats, the third parameter is required to be zero. Since non-static C++ methods have an implicit `this` argument, the arguments of such methods should be counted from two, not one, when giving values for *string-index* and *first-to-check*.

In the example above, the format string (`my_format`) is the second argument of the function `my_print`, and the arguments to check start with the third argument, so the correct parameters for the format attribute are 2 and 3.

The `format` attribute allows you to identify your own functions which take format strings as arguments, so that GCC can check the calls to these functions for errors. The compiler always (unless '`-ffreestanding`' or '`-fno-builtin`' is used) checks formats for the standard library functions `printf`, `fprintf`, `sprintf`, `scanf`, `fscanf`, `sscanf`, `strftime`, `vprintf`, `vfprintf` and `vsprintf` whenever such warnings are requested (using '`-Wformat`'), so there is no need to modify the header file '`stdio.h`'. In C99 mode, the functions `snprintf`, `vsnprintf`, `vscanf`, `vfscanf` and `vsscanf` are also checked. Except in strictly conforming C standard modes, the X/Open function `strfmon` is also checked as are `printf_unlocked` and `fprintf_unlocked`. See Section 3.4 [Options Controlling C Dialect], page 29.

For Objective-C dialects, `NSString` (or `__NSString__`) is recognized in the same context. Declarations including these format attributes will be parsed for correct syntax, however the result of checking of such format strings is not yet defined, and will not be carried out by this version of the compiler.

The target may also provide additional types of format checks. See Section 6.56 [Format Checks Specific to Particular Target Machines], page 609.

`format_arg (`*string-index*`)`

The `format_arg` attribute specifies that a function takes a format string for a `printf`, `scanf`, `strftime` or `strfmon` style function and modifies it (for example, to translate it into another language), so the result can be passed to a `printf`, `scanf`, `strftime` or `strfmon` style function (with the remaining arguments to the format function the same as they would have been for the unmodified string). For example, the declaration:

```
extern char *
my_dgettext (char *my_domain, const char *my_format)
      __attribute__ ((format_arg (2)));
```

causes the compiler to check the arguments in calls to a `printf`, `scanf`, `strftime` or `strfmon` type function, whose format string argument is a

call to the `my_dgettext` function, for consistency with the format string argument `my_format`. If the `format_arg` attribute had not been specified, all the compiler could tell in such calls to format functions would be that the format string argument is not constant; this would generate a warning when '`-Wformat-nonliteral`' is used, but the calls could not be checked without the attribute.

The parameter *string-index* specifies which argument is the format string argument (starting from one). Since non-static C++ methods have an implicit `this` argument, the arguments of such methods should be counted from two.

The `format-arg` attribute allows you to identify your own functions which modify format strings, so that GCC can check the calls to `printf`, `scanf`, `strftime` or `strfmon` type function whose operands are a call to one of your own function. The compiler always treats `gettext`, `dgettext`, and `dcgettext` in this manner except when strict ISO C support is requested by '`-ansi`' or an appropriate '`-std`' option, or '`-ffreestanding`' or '`-fno-builtin`' is used. See Section 3.4 [Options Controlling C Dialect], page 29.

For Objective-C dialects, the `format-arg` attribute may refer to an `NSString` reference for compatibility with the `format` attribute above.

The target may also allow additional types in `format-arg` attributes. See Section 6.56 [Format Checks Specific to Particular Target Machines], page 609.

`function_vector`

Use this attribute on the H8/300, H8/300H, and H8S to indicate that the specified function should be called through the function vector. Calling a function through the function vector will reduce code size, however; the function vector has a limited size (maximum 128 entries on the H8/300 and 64 entries on the H8/300H and H8S) and shares space with the interrupt vector.

In SH2A target, this attribute declares a function to be called using the TBR relative addressing mode. The argument to this attribute is the entry number of the same function in a vector table containing all the TBR relative addressable functions. For the successful jump, register TBR should contain the start address of this TBR relative vector table. In the startup routine of the user application, user needs to care of this TBR register initialization. The TBR relative vector table can have at max 256 function entries. The jumps to these functions will be generated using a SH2A specific, non delayed branch instruction JSR/N `@`(disp8,TBR). You must use GAS and GLD from GNU binutils version 2.7 or later for this attribute to work correctly.

Please refer the example of M16C target, to see the use of this attribute while declaring a function,

In an application, for a function being called once, this attribute will save at least 8 bytes of code; and if other successive calls are being made to the same function, it will save 2 bytes of code per each of these calls.

On M16C/M32C targets, the `function_vector` attribute declares a special page subroutine call function. Use of this attribute reduces the code size by 2 bytes for each call generated to the subroutine. The argument to the attribute is the vector number entry from the special page vector table which contains

the 16 low-order bits of the subroutine's entry address. Each vector table has special page number (18 to 255) which are used in `jsrs` instruction. Jump addresses of the routines are generated by adding 0x0F0000 (in case of M16C targets) or 0xFF0000 (in case of M32C targets), to the 2 byte addresses set in the vector table. Therefore you need to ensure that all the special page vector routines should get mapped within the address range 0x0F0000 to 0x0FFFFF (for M16C) and 0xFF0000 to 0xFFFFFF (for M32C).

In the following example 2 bytes will be saved for each call to function `foo`.

```
void foo (void) __attribute__((function_vector(0x18)));
void foo (void)
{
}

void bar (void)
{
    foo();
}
```

If functions are defined in one file and are called in another file, then be sure to write this declaration in both files.

This attribute is ignored for R8C target.

`ifunc ("`*resolver*`")`

The `ifunc` attribute is used to mark a function as an indirect function using the STT_GNU_IFUNC symbol type extension to the ELF standard. This allows the resolution of the symbol value to be determined dynamically at load time, and an optimized version of the routine can be selected for the particular processor or other system characteristics determined then. To use this attribute, first define the implementation functions available, and a resolver function that returns a pointer to the selected implementation function. The implementation functions' declarations must match the API of the function being implemented, the resolver's declaration is be a function returning pointer to void function returning void:

```
void *my_memcpy (void *dst, const void *src, size_t len)
{
  ...
}

static void (*resolve_memcpy (void)) (void)
{
  return my_memcpy; // we'll just always select this routine
}
```

The exported header file declaring the function the user calls would contain:

```
extern void *memcpy (void *, const void *, size_t);
```

allowing the user to call this as a regular function, unaware of the implementation. Finally, the indirect function needs to be defined in the same translation unit as the resolver function:

```
void *memcpy (void *, const void *, size_t)
     __attribute__ ((ifunc ("resolve_memcpy")));
```

Indirect functions cannot be weak, and require a recent binutils (at least version 2.20.1), and GNU C library (at least version 2.11.1).

`interrupt`

Use this attribute on the ARM, AVR, CR16, Epiphany, M32C, M32R/D, m68k, MeP, MIPS, RL78, RX and Xstormy16 ports to indicate that the specified function is an interrupt handler. The compiler will generate function entry and exit sequences suitable for use in an interrupt handler when this attribute is present. With Epiphany targets it may also generate a special section with code to initialize the interrupt vector table.

Note, interrupt handlers for the Blackfin, H8/300, H8/300H, H8S, MicroBlaze, and SH processors can be specified via the `interrupt_handler` attribute.

Note, on the AVR, the hardware globally disables interrupts when an interrupt is executed. The first instruction of an interrupt handler declared with this attribute will be a `SEI` instruction to re-enable interrupts. See also the `signal` function attribute that does not insert a `SEI` instuction. If both `signal` and `interrupt` are specified for the same function, `signal` will be silently ignored.

Note, for the ARM, you can specify the kind of interrupt to be handled by adding an optional parameter to the interrupt attribute like this:

```
void f () __attribute__ ((interrupt ("IRQ")));
```

Permissible values for this parameter are: IRQ, FIQ, SWI, ABORT and UNDEF.

On ARMv7-M the interrupt type is ignored, and the attribute means the function may be called with a word aligned stack pointer.

On Epiphany targets one or more optional parameters can be added like this:

```
void __attribute__ ((interrupt ("dma0, dma1"))) universal_dma_handler ();
```

Permissible values for these parameters are: `reset`, `software_exception`, `page_miss`, `timer0`, `timer1`, `message`, `dma0`, `dma1`, `wand` and `swi`. Multiple parameters indicate that multiple entries in the interrupt vector table should be initialized for this function, i.e. for each parameter *name*, a jump to the function will be emitted in the section ivt_entry_*name*. The parameter(s) may be omitted entirely, in which case no interrupt vector table entry will be provided.

Note, on Epiphany targets, interrupts are enabled inside the function unless the `disinterrupt` attribute is also specified.

On Epiphany targets, you can also use the following attribute to modify the behavior of an interrupt handler:

`forwarder_section`

The interrupt handler may be in external memory which cannot be reached by a branch instruction, so generate a local memory trampoline to transfer control. The single parameter identifies the section where the trampoline will be placed.

The following examples are all valid uses of these attributes on Epiphany targets:

```
void __attribute__ ((interrupt)) universal_handler ();
void __attribute__ ((interrupt ("dma1"))) dma1_handler ();
void __attribute__ ((interrupt ("dma0, dma1"))) universal_dma_handler ();
```

```
void __attribute__ ((interrupt ("timer0"), disinterrupt))
  fast_timer_handler ();
void __attribute__ ((interrupt ("dma0, dma1"), forwarder_section ("tramp")))
  external_dma_handler ();
```

On MIPS targets, you can use the following attributes to modify the behavior of an interrupt handler:

`use_shadow_register_set`
: Assume that the handler uses a shadow register set, instead of the main general-purpose registers.

`keep_interrupts_masked`
: Keep interrupts masked for the whole function. Without this attribute, GCC tries to reenable interrupts for as much of the function as it can.

`use_debug_exception_return`
: Return using the `deret` instruction. Interrupt handlers that don't have this attribute return using `eret` instead.

You can use any combination of these attributes, as shown below:

```
void __attribute__ ((interrupt)) v0 ();
void __attribute__ ((interrupt, use_shadow_register_set)) v1 ();
void __attribute__ ((interrupt, keep_interrupts_masked)) v2 ();
void __attribute__ ((interrupt, use_debug_exception_return)) v3 ();
void __attribute__ ((interrupt, use_shadow_register_set,
                     keep_interrupts_masked)) v4 ();
void __attribute__ ((interrupt, use_shadow_register_set,
                     use_debug_exception_return)) v5 ();
void __attribute__ ((interrupt, keep_interrupts_masked,
                     use_debug_exception_return)) v6 ();
void __attribute__ ((interrupt, use_shadow_register_set,
                     keep_interrupts_masked,
                     use_debug_exception_return)) v7 ();
```

On RL78, use `brk_interrupt` instead of `interrupt` for handlers intended to be used with the `BRK` opcode (i.e. those that must end with `RETB` instead of `RETI`).

`interrupt_handler`
: Use this attribute on the Blackfin, m68k, H8/300, H8/300H, H8S, and SH to indicate that the specified function is an interrupt handler. The compiler will generate function entry and exit sequences suitable for use in an interrupt handler when this attribute is present.

`interrupt_thread`
: Use this attribute on fido, a subarchitecture of the m68k, to indicate that the specified function is an interrupt handler that is designed to run as a thread. The compiler omits generate prologue/epilogue sequences and replaces the return instruction with a `sleep` instruction. This attribute is available only on fido.

`isr`
: Use this attribute on ARM to write Interrupt Service Routines. This is an alias to the `interrupt` attribute above.

`kspisusp` When used together with `interrupt_handler`, `exception_handler` or `nmi_handler`, code will be generated to load the stack pointer from the USP register in the function prologue.

`l1_text` This attribute specifies a function to be placed into L1 Instruction SRAM. The function will be put into a specific section named `.l1.text`. With '`-mfdpic`', function calls with a such function as the callee or caller will use inlined PLT.

`l2` On the Blackfin, this attribute specifies a function to be placed into L2 SRAM. The function will be put into a specific section named `.l1.text`. With '`-mfdpic`', callers of such functions will use an inlined PLT.

`leaf` Calls to external functions with this attribute must return to the current compilation unit only by return or by exception handling. In particular, leaf functions are not allowed to call callback function passed to it from the current compilation unit or directly call functions exported by the unit or longjmp into the unit. Leaf function might still call functions from other compilation units and thus they are not necessarily leaf in the sense that they contain no function calls at all.

The attribute is intended for library functions to improve dataflow analysis. The compiler takes the hint that any data not escaping the current compilation unit can not be used or modified by the leaf function. For example, the `sin` function is a leaf function, but `qsort` is not.

Note that leaf functions might invoke signals and signal handlers might be defined in the current compilation unit and use static variables. The only compliant way to write such a signal handler is to declare such variables `volatile`.

The attribute has no effect on functions defined within the current compilation unit. This is to allow easy merging of multiple compilation units into one, for example, by using the link time optimization. For this reason the attribute is not allowed on types to annotate indirect calls.

`long_call/short_call`
This attribute specifies how a particular function is called on ARM and Epiphany. Both attributes override the '`-mlong-calls`' (see Section 3.17.2 [ARM Options], page 169) command-line switch and `#pragma long_calls` settings. The `long_call` attribute indicates that the function might be far away from the call site and require a different (more expensive) calling sequence. The `short_call` attribute always places the offset to the function from the call site into the '`BL`' instruction directly.

`longcall/shortcall`
On the Blackfin, RS/6000 and PowerPC, the `longcall` attribute indicates that the function might be far away from the call site and require a different (more expensive) calling sequence. The `shortcall` attribute indicates that the function is always close enough for the shorter calling sequence to be used. These attributes override both the '`-mlongcall`' switch and, on the RS/6000 and PowerPC, the `#pragma longcall` setting.

See Section 3.17.34 [RS/6000 and PowerPC Options], page 249, for more information on whether long calls are necessary.

`long_call/near/far`
: These attributes specify how a particular function is called on MIPS. The attributes override the '`-mlong-calls`' (see Section 3.17.27 [MIPS Options], page 233) command-line switch. The `long_call` and `far` attributes are synonyms, and cause the compiler to always call the function by first loading its address into a register, and then using the contents of that register. The `near` attribute has the opposite effect; it specifies that non-PIC calls should be made using the more efficient `jal` instruction.

`malloc`
: The `malloc` attribute is used to tell the compiler that a function may be treated as if any non-`NULL` pointer it returns cannot alias any other pointer valid when the function returns and that the memory has undefined content. This will often improve optimization. Standard functions with this property include `malloc` and `calloc`. `realloc`-like functions do not have this property as the memory pointed to does not have undefined content.

`mips16/nomips16`
: On MIPS targets, you can use the `mips16` and `nomips16` function attributes to locally select or turn off MIPS16 code generation. A function with the `mips16` attribute is emitted as MIPS16 code, while MIPS16 code generation is disabled for functions with the `nomips16` attribute. These attributes override the '`-mips16`' and '`-mno-mips16`' options on the command line (see Section 3.17.27 [MIPS Options], page 233).

  When compiling files containing mixed MIPS16 and non-MIPS16 code, the preprocessor symbol `__mips16` reflects the setting on the command line, not that within individual functions. Mixed MIPS16 and non-MIPS16 code may interact badly with some GCC extensions such as `__builtin_apply` (see Section 6.5 [Constructing Calls], page 314).

`model (`*`model-name`*`)`
: On the M32R/D, use this attribute to set the addressability of an object, and of the code generated for a function. The identifier *model-name* is one of `small`, `medium`, or `large`, representing each of the code models.

  Small model objects live in the lower 16MB of memory (so that their addresses can be loaded with the `ld24` instruction), and are callable with the `bl` instruction.

  Medium model objects may live anywhere in the 32-bit address space (the compiler will generate `seth/add3` instructions to load their addresses), and are callable with the `bl` instruction.

  Large model objects may live anywhere in the 32-bit address space (the compiler will generate `seth/add3` instructions to load their addresses), and may not be reachable with the `bl` instruction (the compiler will generate the much slower `seth/add3/jl` instruction sequence).

  On IA-64, use this attribute to set the addressability of an object. At present, the only supported identifier for *model-name* is `small`, indicating addressability via "small" (22-bit) addresses (so that their addresses can be loaded with the `addl` instruction). Caveat: such addressing is by definition not position

independent and hence this attribute must not be used for objects defined by shared libraries.

`ms_abi/sysv_abi`
On 32-bit and 64-bit (i?86|x86_64)-*-* targets, you can use an ABI attribute to indicate which calling convention should be used for a function. The `ms_abi` attribute tells the compiler to use the Microsoft ABI, while the `sysv_abi` attribute tells the compiler to use the ABI used on GNU/Linux and other systems. The default is to use the Microsoft ABI when targeting Windows. On all other systems, the default is the x86/AMD ABI.

Note, the `ms_abi` attribute for Windows 64-bit targets currently requires the '`-maccumulate-outgoing-args`' option.

`callee_pop_aggregate_return (`*number*`)`
On 32-bit i?86-*-* targets, you can control by those attribute for aggregate return in memory, if the caller is responsible to pop the hidden pointer together with the rest of the arguments - *number* equal to zero -, or if the callee is responsible to pop hidden pointer - *number* equal to one. The default i386 ABI assumes that the callee pops the stack for hidden pointer.

Note, that on 32-bit i386 Windows targets the compiler assumes that the caller pops the stack for hidden pointer.

`ms_hook_prologue`
On 32 bit i[34567]86-*-* targets and 64 bit x86_64-*-* targets, you can use this function attribute to make gcc generate the "hot-patching" function prologue used in Win32 API functions in Microsoft Windows XP Service Pack 2 and newer.

`naked` Use this attribute on the ARM, AVR, MCORE, RX and SPU ports to indicate that the specified function does not need prologue/epilogue sequences generated by the compiler. It is up to the programmer to provide these sequences. The only statements that can be safely included in naked functions are `asm` statements that do not have operands. All other statements, including declarations of local variables, `if` statements, and so forth, should be avoided. Naked functions should be used to implement the body of an assembly function, while allowing the compiler to construct the requisite function declaration for the assembler.

`near` On 68HC11 and 68HC12 the `near` attribute causes the compiler to use the normal calling convention based on `jsr` and `rts`. This attribute can be used to cancel the effect of the '`-mlong-calls`' option.

On MeP targets this attribute causes the compiler to assume the called function is close enough to use the normal calling convention, overriding the `-mtf` command line option.

`nesting` Use this attribute together with `interrupt_handler`, `exception_handler` or `nmi_handler` to indicate that the function entry code should enable nested interrupts or exceptions.

`nmi_handler`
: Use this attribute on the Blackfin to indicate that the specified function is an NMI handler. The compiler will generate function entry and exit sequences suitable for use in an NMI handler when this attribute is present.

`no_instrument_function`
: If '`-finstrument-functions`' is given, profiling function calls will be generated at entry and exit of most user-compiled functions. Functions with this attribute will not be so instrumented.

`no_split_stack`
: If '`-fsplit-stack`' is given, functions will have a small prologue which decides whether to split the stack. Functions with the `no_split_stack` attribute will not have that prologue, and thus may run with only a small amount of stack space available.

`noinline`
: This function attribute prevents a function from being considered for inlining. If the function does not have side-effects, there are optimizations other than inlining that causes function calls to be optimized away, although the function call is live. To keep such calls from being optimized away, put

```
asm ("");
```

(see Section 6.41 [Extended Asm], page 383) in the called function, to serve as a special side-effect.

`noclone`
: This function attribute prevents a function from being considered for cloning - a mechanism which produces specialized copies of functions and which is (currently) performed by interprocedural constant propagation.

`nonnull (`*`arg-index`*`, ...)`
: The `nonnull` attribute specifies that some function parameters should be non-null pointers. For instance, the declaration:

```
extern void *
my_memcpy (void *dest, const void *src, size_t len)
        __attribute__((nonnull (1, 2)));
```

causes the compiler to check that, in calls to `my_memcpy`, arguments *dest* and *src* are non-null. If the compiler determines that a null pointer is passed in an argument slot marked as non-null, and the '`-Wnonnull`' option is enabled, a warning is issued. The compiler may also choose to make optimizations based on the knowledge that certain function arguments will not be null.

If no argument index list is given to the `nonnull` attribute, all pointer arguments are marked as non-null. To illustrate, the following declaration is equivalent to the previous example:

```
extern void *
my_memcpy (void *dest, const void *src, size_t len)
        __attribute__((nonnull));
```

`noreturn`
: A few standard library functions, such as `abort` and `exit`, cannot return. GCC knows this automatically. Some programs define their own functions that never return. You can declare them `noreturn` to tell the compiler this fact. For example,

```
void fatal () __attribute__ ((noreturn));

void
fatal (/* ... */)
{
  /* ... */ /* Print error message. */ /* ... */
  exit (1);
}
```

The `noreturn` keyword tells the compiler to assume that `fatal` cannot return. It can then optimize without regard to what would happen if `fatal` ever did return. This makes slightly better code. More importantly, it helps avoid spurious warnings of uninitialized variables.

The `noreturn` keyword does not affect the exceptional path when that applies: a `noreturn`-marked function may still return to the caller by throwing an exception or calling `longjmp`.

Do not assume that registers saved by the calling function are restored before calling the `noreturn` function.

It does not make sense for a `noreturn` function to have a return type other than `void`.

The attribute `noreturn` is not implemented in GCC versions earlier than 2.5. An alternative way to declare that a function does not return, which works in the current version and in some older versions, is as follows:

```
typedef void voidfn ();

volatile voidfn fatal;
```

This approach does not work in GNU C++.

`nothrow` The `nothrow` attribute is used to inform the compiler that a function cannot throw an exception. For example, most functions in the standard C library can be guaranteed not to throw an exception with the notable exceptions of `qsort` and `bsearch` that take function pointer arguments. The `nothrow` attribute is not implemented in GCC versions earlier than 3.3.

`optimize` The `optimize` attribute is used to specify that a function is to be compiled with different optimization options than specified on the command line. Arguments can either be numbers or strings. Numbers are assumed to be an optimization level. Strings that begin with `O` are assumed to be an optimization option, while other options are assumed to be used with a `-f` prefix. You can also use the '`#pragma GCC optimize`' pragma to set the optimization options that affect more than one function. See Section 6.57.13 [Function Specific Option Pragmas], page 616, for details about the '`#pragma GCC optimize`' pragma.

This can be used for instance to have frequently executed functions compiled with more aggressive optimization options that produce faster and larger code, while other functions can be called with less aggressive options.

`OS_main/OS_task`
On AVR, functions with the `OS_main` or `OS_task` attribute do not save/restore any call-saved register in their prologue/epilogue.

The `OS_main` attribute can be used when there *is guarantee* that interrupts are disabled at the time when the function is entered. This will save resources when the stack pointer has to be changed to set up a frame for local variables.

The `OS_task` attribute can be used when there is *no guarantee* that interrupts are disabled at that time when the function is entered like for, e.g. task functions in a multi-threading operating system. In that case, changing the stack pointer register will be guarded by save/clear/restore of the global interrupt enable flag.

The differences to the `naked` function attribute are:

- `naked` functions do not have a return instruction whereas `OS_main` and `OS_task` functions will have a `RET` or `RETI` return instruction.
- `naked` functions do not set up a frame for local variables or a frame pointer whereas `OS_main` and `OS_task` do this as needed.

`pcs`

The `pcs` attribute can be used to control the calling convention used for a function on ARM. The attribute takes an argument that specifies the calling convention to use.

When compiling using the AAPCS ABI (or a variant of that) then valid values for the argument are `"aapcs"` and `"aapcs-vfp"`. In order to use a variant other than `"aapcs"` then the compiler must be permitted to use the appropriate co-processor registers (i.e., the VFP registers must be available in order to use `"aapcs-vfp"`). For example,

```
/* Argument passed in r0, and result returned in r0+r1.  */
double f2d (float) __attribute__((pcs("aapcs")));
```

Variadic functions always use the `"aapcs"` calling convention and the compiler will reject attempts to specify an alternative.

`pure` Many functions have no effects except the return value and their return value depends only on the parameters and/or global variables. Such a function can be subject to common subexpression elimination and loop optimization just as an arithmetic operator would be. These functions should be declared with the attribute `pure`. For example,

```
int square (int) __attribute__ ((pure));
```

says that the hypothetical function `square` is safe to call fewer times than the program says.

Some of common examples of pure functions are `strlen` or `memcmp`. Interesting non-pure functions are functions with infinite loops or those depending on volatile memory or other system resource, that may change between two consecutive calls (such as `feof` in a multithreading environment).

The attribute `pure` is not implemented in GCC versions earlier than 2.96.

`hot` The `hot` attribute is used to inform the compiler that a function is a hot spot of the compiled program. The function is optimized more aggressively and on many target it is placed into special subsection of the text section so all hot functions appears close together improving locality.

When profile feedback is available, via '`-fprofile-use`', hot functions are automatically detected and this attribute is ignored.

The `hot` attribute is not implemented in GCC versions earlier than 4.3.

`cold` The `cold` attribute is used to inform the compiler that a function is unlikely executed. The function is optimized for size rather than speed and on many targets it is placed into special subsection of the text section so all cold functions appears close together improving code locality of non-cold parts of program. The paths leading to call of cold functions within code are marked as unlikely by the branch prediction mechanism. It is thus useful to mark functions used to handle unlikely conditions, such as `perror`, as cold to improve optimization of hot functions that do call marked functions in rare occasions.

When profile feedback is available, via '`-fprofile-use`', hot functions are automatically detected and this attribute is ignored.

The `cold` attribute is not implemented in GCC versions earlier than 4.3.

`regparm (`*number*`)`
On the Intel 386, the `regparm` attribute causes the compiler to pass arguments number one to *number* if they are of integral type in registers EAX, EDX, and ECX instead of on the stack. Functions that take a variable number of arguments will continue to be passed all of their arguments on the stack.

Beware that on some ELF systems this attribute is unsuitable for global functions in shared libraries with lazy binding (which is the default). Lazy binding will send the first call via resolving code in the loader, which might assume EAX, EDX and ECX can be clobbered, as per the standard calling conventions. Solaris 8 is affected by this. GNU systems with GLIBC 2.1 or higher, and FreeBSD, are believed to be safe since the loaders there save EAX, EDX and ECX. (Lazy binding can be disabled with the linker or the loader if desired, to avoid the problem.)

`sseregparm`
On the Intel 386 with SSE support, the `sseregparm` attribute causes the compiler to pass up to 3 floating point arguments in SSE registers instead of on the stack. Functions that take a variable number of arguments will continue to pass all of their floating point arguments on the stack.

`force_align_arg_pointer`
On the Intel x86, the `force_align_arg_pointer` attribute may be applied to individual function definitions, generating an alternate prologue and epilogue that realigns the runtime stack if necessary. This supports mixing legacy codes that run with a 4-byte aligned stack with modern codes that keep a 16-byte stack for SSE compatibility.

`resbank` On the SH2A target, this attribute enables the high-speed register saving and restoration using a register bank for `interrupt_handler` routines. Saving to the bank is performed automatically after the CPU accepts an interrupt that uses a register bank.

The nineteen 32-bit registers comprising general register R0 to R14, control register GBR, and system registers MACH, MACL, and PR and the vector table address offset are saved into a register bank. Register banks are stacked

in first-in last-out (FILO) sequence. Restoration from the bank is executed by issuing a RESBANK instruction.

`returns_twice`
: The `returns_twice` attribute tells the compiler that a function may return more than one time. The compiler will ensure that all registers are dead before calling such a function and will emit a warning about the variables that may be clobbered after the second return from the function. Examples of such functions are `setjmp` and `vfork`. The `longjmp`-like counterpart of such function, if any, might need to be marked with the `noreturn` attribute.

`saveall`
: Use this attribute on the Blackfin, H8/300, H8/300H, and H8S to indicate that all registers except the stack pointer should be saved in the prologue regardless of whether they are used or not.

`save_volatiles`
: Use this attribute on the MicroBlaze to indicate that the function is an interrupt handler. All volatile registers (in addition to non-volatile registers) will be saved in the function prologue. If the function is a leaf function, only volatiles used by the function are saved. A normal function return is generated instead of a return from interrupt.

`section ("`*section-name*`")`
: Normally, the compiler places the code it generates in the `text` section. Sometimes, however, you need additional sections, or you need certain particular functions to appear in special sections. The `section` attribute specifies that a function lives in a particular section. For example, the declaration:

```
extern void foobar (void) __attribute__ ((section ("bar")));
```

puts the function `foobar` in the `bar` section.

Some file formats do not support arbitrary sections so the `section` attribute is not available on all platforms. If you need to map the entire contents of a module to a particular section, consider using the facilities of the linker instead.

`sentinel`
: This function attribute ensures that a parameter in a function call is an explicit `NULL`. The attribute is only valid on variadic functions. By default, the sentinel is located at position zero, the last parameter of the function call. If an optional integer position argument P is supplied to the attribute, the sentinel must be located at position P counting backwards from the end of the argument list.

```
__attribute__ ((sentinel))
is equivalent to
__attribute__ ((sentinel(0)))
```

The attribute is automatically set with a position of 0 for the built-in functions `execl` and `execlp`. The built-in function `execle` has the attribute set with a position of 1.

A valid `NULL` in this context is defined as zero with any pointer type. If your system defines the `NULL` macro with an integer type then you need to add an explicit cast. GCC replaces `stddef.h` with a copy that redefines NULL appropriately.

The warnings for missing or incorrect sentinels are enabled with '`-Wformat`'.

`short_call`
: See long_call/short_call.

`shortcall`
: See longcall/shortcall.

`signal`
: Use this attribute on the AVR to indicate that the specified function is an interrupt handler. The compiler will generate function entry and exit sequences suitable for use in an interrupt handler when this attribute is present.

  See also the `interrupt` function attribute.

  The AVR hardware globally disables interrupts when an interrupt is executed. Interrupt handler functions defined with the `signal` attribute do not re-enable interrupts. It is save to enable interrupts in a `signal` handler. This "save" only applies to the code generated by the compiler and not to the IRQ-layout of the application which is responsibility of the application.

  If both `signal` and `interrupt` are specified for the same function, `signal` will be silently ignored.

`sp_switch`
: Use this attribute on the SH to indicate an `interrupt_handler` function should switch to an alternate stack. It expects a string argument that names a global variable holding the address of the alternate stack.

```
void *alt_stack;
void f () __attribute__ ((interrupt_handler,
                          sp_switch ("alt_stack")));
```

`stdcall`
: On the Intel 386, the `stdcall` attribute causes the compiler to assume that the called function will pop off the stack space used to pass arguments, unless it takes a variable number of arguments.

`syscall_linkage`
: This attribute is used to modify the IA64 calling convention by marking all input registers as live at all function exits. This makes it possible to restart a system call after an interrupt without having to save/restore the input registers. This also prevents kernel data from leaking into application code.

`target`
: The `target` attribute is used to specify that a function is to be compiled with different target options than specified on the command line. This can be used for instance to have functions compiled with a different ISA (instruction set architecture) than the default. You can also use the '`#pragma GCC target`' pragma to set more than one function to be compiled with specific target options. See Section 6.57.13 [Function Specific Option Pragmas], page 616, for details about the '`#pragma GCC target`' pragma.

  For instance on a 386, you could compile one function with `target("sse4.1,arch=core2")` and another with `target("sse4a,arch=amdfam10")` that would be equivalent to compiling the first function with '`-msse4.1`' and '`-march=core2`' options, and the second function with '`-msse4a`' and '`-march=amdfam10`' options. It is up to the user to make sure that a function is only invoked on a machine that supports the particular ISA it was compiled

for (for example by using `cpuid` on 386 to determine what feature bits and architecture family are used).

```
int core2_func (void) __attribute__ ((__target__ ("arch=core2")));
int sse3_func (void) __attribute__ ((__target__ ("sse3")));
```

On the 386, the following options are allowed:

‘`abm`’
‘`no-abm`’ Enable/disable the generation of the advanced bit instructions.

‘`aes`’
‘`no-aes`’ Enable/disable the generation of the AES instructions.

‘`mmx`’
‘`no-mmx`’ Enable/disable the generation of the MMX instructions.

‘`pclmul`’
‘`no-pclmul`’
Enable/disable the generation of the PCLMUL instructions.

‘`popcnt`’
‘`no-popcnt`’
Enable/disable the generation of the POPCNT instruction.

‘`sse`’
‘`no-sse`’ Enable/disable the generation of the SSE instructions.

‘`sse2`’
‘`no-sse2`’ Enable/disable the generation of the SSE2 instructions.

‘`sse3`’
‘`no-sse3`’ Enable/disable the generation of the SSE3 instructions.

‘`sse4`’
‘`no-sse4`’ Enable/disable the generation of the SSE4 instructions (both SSE4.1 and SSE4.2).

‘`sse4.1`’
‘`no-sse4.1`’
Enable/disable the generation of the sse4.1 instructions.

‘`sse4.2`’
‘`no-sse4.2`’
Enable/disable the generation of the sse4.2 instructions.

‘`sse4a`’
‘`no-sse4a`’
Enable/disable the generation of the SSE4A instructions.

‘`fma4`’
‘`no-fma4`’ Enable/disable the generation of the FMA4 instructions.

‘`xop`’
‘`no-xop`’ Enable/disable the generation of the XOP instructions.

‘`lwp`’
‘`no-lwp`’ Enable/disable the generation of the LWP instructions.

'`ssse3`'
'`no-ssse3`'
: Enable/disable the generation of the SSSE3 instructions.

'`cld`'
'`no-cld`'
: Enable/disable the generation of the CLD before string moves.

'`fancy-math-387`'
'`no-fancy-math-387`'
: Enable/disable the generation of the `sin`, `cos`, and `sqrt` instructions on the 387 floating point unit.

'`fused-madd`'
'`no-fused-madd`'
: Enable/disable the generation of the fused multiply/add instructions.

'`ieee-fp`'
'`no-ieee-fp`'
: Enable/disable the generation of floating point that depends on IEEE arithmetic.

'`inline-all-stringops`'
'`no-inline-all-stringops`'
: Enable/disable inlining of string operations.

'`inline-stringops-dynamically`'
'`no-inline-stringops-dynamically`'
: Enable/disable the generation of the inline code to do small string operations and calling the library routines for large operations.

'`align-stringops`'
'`no-align-stringops`'
: Do/do not align destination of inlined string operations.

'`recip`'
'`no-recip`'
: Enable/disable the generation of RCPSS, RCPPS, RSQRTSS and RSQRTPS instructions followed an additional Newton-Raphson step instead of doing a floating point division.

'`arch=`*`ARCH`*'
: Specify the architecture to generate code for in compiling the function.

'`tune=`*`TUNE`*'
: Specify the architecture to tune for in compiling the function.

'`fpmath=`*`FPMATH`*'
: Specify which floating point unit to use. The `target("fpmath=sse,387")` option must be specified as `target("fpmath=sse+387")` because the comma would separate different options.

On the PowerPC, the following options are allowed:

‘`altivec`’
‘`no-altivec`’
: Generate code that uses (does not use) AltiVec instructions. In 32-bit code, you cannot enable Altivec instructions unless ‘`-mabi=altivec`’ was used on the command line.

‘`cmpb`’
‘`no-cmpb`’
: Generate code that uses (does not use) the compare bytes instruction implemented on the POWER6 processor and other processors that support the PowerPC V2.05 architecture.

‘`dlmzb`’
‘`no-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.

‘`fprnd`’
‘`no-fprnd`’
: Generate code that uses (does not use) the FP round to integer instructions implemented on the POWER5+ processor and other processors that support the PowerPC V2.03 architecture.

‘`hard-dfp`’
‘`no-hard-dfp`’
: Generate code that uses (does not use) the decimal floating point instructions implemented on some POWER processors.

‘`isel`’
‘`no-isel`’
: Generate code that uses (does not use) ISEL instruction.

‘`mfcrf`’
‘`no-mfcrf`’
: Generate code that uses (does not use) the move from condition register field instruction implemented on the POWER4 processor and other processors that support the PowerPC V2.01 architecture.

‘`mfpgpr`’
‘`no-mfpgpr`’
: Generate code that uses (does not use) 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.

‘`mulhw`’
‘`no-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.

‘multiple’
‘no-multiple’
Generate code that uses (does not use) the load multiple word instructions and the store multiple word instructions.

‘update’
‘no-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.

‘popcntb’
‘no-popcntb’
Generate code that uses (does not use) the popcount and double precision FP reciprocal estimate instruction implemented on the POWER5 processor and other processors that support the PowerPC V2.02 architecture.

‘popcntd’
‘no-popcntd’
Generate code that uses (does not use) the popcount instruction implemented on the POWER7 processor and other processors that support the PowerPC V2.06 architecture.

‘powerpc-gfxopt’
‘no-powerpc-gfxopt’
Generate code that uses (does not use) the optional PowerPC architecture instructions in the Graphics group, including floating-point select.

‘powerpc-gpopt’
‘no-powerpc-gpopt’
Generate code that uses (does not use) the optional PowerPC architecture instructions in the General Purpose group, including floating-point square root.

‘recip-precision’
‘no-recip-precision’
Assume (do not assume) that the reciprocal estimate instructions provide higher precision estimates than is mandated by the powerpc ABI.

‘string’
‘no-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.

‘vsx’
‘no-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. In 32-bit code, you cannot enable VSX or Altivec instructions unless '`-mabi=altivec`' was used on the command line.

'`friz`'
'`no-friz`' Generate (do not generate) the `friz` instruction when the '`-funsafe-math-optimizations`' option is used to optimize rounding a floating point value 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.

'`avoid-indexed-addresses`'
'`no-avoid-indexed-addresses`'
Generate code that tries to avoid (not avoid) the use of indexed load or store instructions.

'`paired`'
'`no-paired`'
Generate code that uses (does not use) the generation of PAIRED simd instructions.

'`longcall`'
'`no-longcall`'
Generate code that assumes (does not assume) that all calls are far away so that a longer more expensive calling sequence is required.

'`cpu=`*CPU*' Specify the architecture to generate code for when compiling the function. If you select the `target("cpu=power7")` attribute when generating 32-bit code, VSX and Altivec instructions are not generated unless you use the '`-mabi=altivec`' option on the command line.

'`tune=`*TUNE*'
Specify the architecture to tune for when compiling the function. If you do not specify the `target("tune=`*TUNE*`")` attribute and you do specify the `target("cpu=`*CPU*`")` attribute, compilation will tune for the *CPU* architecture, and not the default tuning specified on the command line.

On the 386/x86_64 and PowerPC backends, you can use either multiple strings to specify multiple options, or you can separate the option with a comma (`,`).

On the 386/x86_64 and PowerPC backends, the inliner will not inline a function that has different target options than the caller, unless the callee has a subset of the target options of the caller. For example a function declared with `target("sse3")` can inline a function with `target("sse2")`, since `-msse3` implies `-msse2`.

The `target` attribute is not implemented in GCC versions earlier than 4.4 for the i386/x86_64 and 4.6 for the PowerPC backends. It is not currently implemented for other backends.

`tiny_data`
: Use this attribute on the H8/300H and H8S to indicate that the specified variable should be placed into the tiny data section. The compiler will generate more efficient code for loads and stores on data in the tiny data section. Note the tiny data area is limited to slightly under 32kbytes of data.

`trap_exit`
: Use this attribute on the SH for an `interrupt_handler` to return using `trapa` instead of `rte`. This attribute expects an integer argument specifying the trap number to be used.

`unused`
: This attribute, attached to a function, means that the function is meant to be possibly unused. GCC will not produce a warning for this function.

`used`
: This attribute, attached to a function, means that code must be emitted for the function even if it appears that the function is not referenced. This is useful, for example, when the function is referenced only in inline assembly.

  When applied to a member function of a C++ class template, the attribute also means that the function will be instantiated if the class itself is instantiated.

`version_id`
: This IA64 HP-UX attribute, attached to a global variable or function, renames a symbol to contain a version string, thus allowing for function level versioning. HP-UX system header files may use version level functioning for some system calls.

  ```
  extern int foo () __attribute__((version_id ("20040821")));
  ```

  Calls to *foo* will be mapped to calls to *foo*`{`*20040821*`}`.

`visibility ("`*visibility_type*`")`
: This attribute affects the linkage of the declaration to which it is attached. There are four supported *visibility_type* values: default, hidden, protected or internal visibility.

  ```
  void __attribute__ ((visibility ("protected")))
  f () { /* Do something. */; }
  int i __attribute__ ((visibility ("hidden")));
  ```

  The possible values of *visibility_type* correspond to the visibility settings in the ELF gABI.

  default
  : Default visibility is the normal case for the object file format. This value is available for the visibility attribute to override other options that may change the assumed visibility of entities.

    On ELF, default visibility means that the declaration is visible to other modules and, in shared libraries, means that the declared entity may be overridden.

    On Darwin, default visibility means that the declaration is visible to other modules.

    Default visibility corresponds to “external linkage” in the language.

  hidden
  : Hidden visibility indicates that the entity declared will have a new form of linkage, which we'll call “hidden linkage”. Two declarations

of an object with hidden linkage refer to the same object if they are in the same shared object.

internal Internal visibility is like hidden visibility, but with additional processor specific semantics. Unless otherwise specified by the psABI, GCC defines internal visibility to mean that a function is *never* called from another module. Compare this with hidden functions which, while they cannot be referenced directly by other modules, can be referenced indirectly via function pointers. By indicating that a function cannot be called from outside the module, GCC may for instance omit the load of a PIC register since it is known that the calling function loaded the correct value.

protected Protected visibility is like default visibility except that it indicates that references within the defining module will bind to the definition in that module. That is, the declared entity cannot be overridden by another module.

All visibilities are supported on many, but not all, ELF targets (supported when the assembler supports the '`.visibility`' pseudo-op). Default visibility is supported everywhere. Hidden visibility is supported on Darwin targets.

The visibility attribute should be applied only to declarations which would otherwise have external linkage. The attribute should be applied consistently, so that the same entity should not be declared with different settings of the attribute.

In C++, the visibility attribute applies to types as well as functions and objects, because in C++ types have linkage. A class must not have greater visibility than its non-static data member types and bases, and class members default to the visibility of their class. Also, a declaration without explicit visibility is limited to the visibility of its type.

In C++, you can mark member functions and static member variables of a class with the visibility attribute. This is useful if you know a particular method or static member variable should only be used from one shared object; then you can mark it hidden while the rest of the class has default visibility. Care must be taken to avoid breaking the One Definition Rule; for example, it is usually not useful to mark an inline method as hidden without marking the whole class as hidden.

A C++ namespace declaration can also have the visibility attribute. This attribute applies only to the particular namespace body, not to other definitions of the same namespace; it is equivalent to using '`#pragma GCC visibility`' before and after the namespace definition (see Section 6.57.11 [Visibility Pragmas], page 615).

In C++, if a template argument has limited visibility, this restriction is implicitly propagated to the template instantiation. Otherwise, template instantiations and specializations default to the visibility of their template.

If both the template and enclosing class have explicit visibility, the visibility from the template is used.

`vliw` On MeP, the `vliw` attribute tells the compiler to emit instructions in VLIW mode instead of core mode. Note that this attribute is not allowed unless a VLIW coprocessor has been configured and enabled through command line options.

`warn_unused_result`
: The `warn_unused_result` attribute causes a warning to be emitted if a caller of the function with this attribute does not use its return value. This is useful for functions where not checking the result is either a security problem or always a bug, such as `realloc`.

```
int fn () __attribute__ ((warn_unused_result));
int foo ()
{
  if (fn () < 0) return -1;
  fn ();
  return 0;
}
```

results in warning on line 5.

`weak` The `weak` attribute causes the declaration to be emitted as a weak symbol rather than a global. This is primarily useful in defining library functions which can be overridden in user code, though it can also be used with non-function declarations. Weak symbols are supported for ELF targets, and also for a.out targets when using the GNU assembler and linker.

`weakref`
`weakref ("`*target*`")`
: The `weakref` attribute marks a declaration as a weak reference. Without arguments, it should be accompanied by an `alias` attribute naming the target symbol. Optionally, the *target* may be given as an argument to `weakref` itself. In either case, `weakref` implicitly marks the declaration as `weak`. Without a *target*, given as an argument to `weakref` or to `alias`, `weakref` is equivalent to `weak`.

```
static int x() __attribute__ ((weakref ("y")));
/* is equivalent to... */
static int x() __attribute__ ((weak, weakref, alias ("y")));
/* and to... */
static int x() __attribute__ ((weakref));
static int x() __attribute__ ((alias ("y")));
```

A weak reference is an alias that does not by itself require a definition to be given for the target symbol. If the target symbol is only referenced through weak references, then it becomes a `weak` undefined symbol. If it is directly referenced, however, then such strong references prevail, and a definition will be required for the symbol, not necessarily in the same translation unit.

The effect is equivalent to moving all references to the alias to a separate translation unit, renaming the alias to the aliased symbol, declaring it as weak, compiling the two separate translation units and performing a reloadable link on them.

At present, a declaration to which `weakref` is attached can only be `static`.

You can specify multiple attributes in a declaration by separating them by commas within the double parentheses or by immediately following an attribute declaration with another attribute declaration.

Some people object to the `__attribute__` feature, suggesting that ISO C's `#pragma` should be used instead. At the time `__attribute__` was designed, there were two reasons for not doing this.

1. It is impossible to generate `#pragma` commands from a macro.
2. There is no telling what the same `#pragma` might mean in another compiler.

These two reasons applied to almost any application that might have been proposed for `#pragma`. It was basically a mistake to use `#pragma` for *anything*.

The ISO C99 standard includes `_Pragma`, which now allows pragmas to be generated from macros. In addition, a `#pragma GCC` namespace is now in use for GCC-specific pragmas. However, it has been found convenient to use `__attribute__` to achieve a natural attachment of attributes to their corresponding declarations, whereas `#pragma GCC` is of use for constructs that do not naturally form part of the grammar. See Section "Miscellaneous Preprocessing Directives" in *The GNU C Preprocessor*.

6.31 Attribute Syntax

This section describes the syntax with which `__attribute__` may be used, and the constructs to which attribute specifiers bind, for the C language. Some details may vary for C++ and Objective-C. Because of infelicities in the grammar for attributes, some forms described here may not be successfully parsed in all cases.

There are some problems with the semantics of attributes in C++. For example, there are no manglings for attributes, although they may affect code generation, so problems may arise when attributed types are used in conjunction with templates or overloading. Similarly, `typeid` does not distinguish between types with different attributes. Support for attributes in C++ may be restricted in future to attributes on declarations only, but not on nested declarators.

See Section 6.30 [Function Attributes], page 332, for details of the semantics of attributes applying to functions. See Section 6.36 [Variable Attributes], page 365, for details of the semantics of attributes applying to variables. See Section 6.37 [Type Attributes], page 374, for details of the semantics of attributes applying to structure, union and enumerated types.

An *attribute specifier* is of the form `__attribute__ ((attribute-list))`. An *attribute list* is a possibly empty comma-separated sequence of *attributes*, where each attribute is one of the following:

- Empty. Empty attributes are ignored.
- A word (which may be an identifier such as `unused`, or a reserved word such as `const`).
- A word, followed by, in parentheses, parameters for the attribute. These parameters take one of the following forms:
  - An identifier. For example, `mode` attributes use this form.
  - An identifier followed by a comma and a non-empty comma-separated list of expressions. For example, `format` attributes use this form.

- A possibly empty comma-separated list of expressions. For example, `format_arg` attributes use this form with the list being a single integer constant expression, and `alias` attributes use this form with the list being a single string constant.

An *attribute specifier list* is a sequence of one or more attribute specifiers, not separated by any other tokens.

In GNU C, an attribute specifier list may appear after the colon following a label, other than a `case` or `default` label. The only attribute it makes sense to use after a label is `unused`. This feature is intended for code generated by programs which contains labels that may be unused but which is compiled with '`-Wall`'. It would not normally be appropriate to use in it human-written code, though it could be useful in cases where the code that jumps to the label is contained within an `#ifdef` conditional. GNU C++ only permits attributes on labels if the attribute specifier is immediately followed by a semicolon (i.e., the label applies to an empty statement). If the semicolon is missing, C++ label attributes are ambiguous, as it is permissible for a declaration, which could begin with an attribute list, to be labelled in C++. Declarations cannot be labelled in C90 or C99, so the ambiguity does not arise there.

An attribute specifier list may appear as part of a `struct`, `union` or `enum` specifier. It may go either immediately after the `struct`, `union` or `enum` keyword, or after the closing brace. The former syntax is preferred. Where attribute specifiers follow the closing brace, they are considered to relate to the structure, union or enumerated type defined, not to any enclosing declaration the type specifier appears in, and the type defined is not complete until after the attribute specifiers.

Otherwise, an attribute specifier appears as part of a declaration, counting declarations of unnamed parameters and type names, and relates to that declaration (which may be nested in another declaration, for example in the case of a parameter declaration), or to a particular declarator within a declaration. Where an attribute specifier is applied to a parameter declared as a function or an array, it should apply to the function or array rather than the pointer to which the parameter is implicitly converted, but this is not yet correctly implemented.

Any list of specifiers and qualifiers at the start of a declaration may contain attribute specifiers, whether or not such a list may in that context contain storage class specifiers. (Some attributes, however, are essentially in the nature of storage class specifiers, and only make sense where storage class specifiers may be used; for example, `section`.) There is one necessary limitation to this syntax: the first old-style parameter declaration in a function definition cannot begin with an attribute specifier, because such an attribute applies to the function instead by syntax described below (which, however, is not yet implemented in this case). In some other cases, attribute specifiers are permitted by this grammar but not yet supported by the compiler. All attribute specifiers in this place relate to the declaration as a whole. In the obsolescent usage where a type of `int` is implied by the absence of type specifiers, such a list of specifiers and qualifiers may be an attribute specifier list with no other specifiers or qualifiers.

At present, the first parameter in a function prototype must have some type specifier which is not an attribute specifier; this resolves an ambiguity in the interpretation of `void f(int (__attribute__((foo)) x))`, but is subject to change. At present, if the parentheses of a function declarator contain only attributes then those attributes are ignored, rather

than yielding an error or warning or implying a single parameter of type int, but this is subject to change.

An attribute specifier list may appear immediately before a declarator (other than the first) in a comma-separated list of declarators in a declaration of more than one identifier using a single list of specifiers and qualifiers. Such attribute specifiers apply only to the identifier before whose declarator they appear. For example, in

```
__attribute__((noreturn)) void d0 (void),
    __attribute__((format(printf, 1, 2))) d1 (const char *, ...),
     d2 (void)
```

the `noreturn` attribute applies to all the functions declared; the `format` attribute only applies to `d1`.

An attribute specifier list may appear immediately before the comma, `=` or semicolon terminating the declaration of an identifier other than a function definition. Such attribute specifiers apply to the declared object or function. Where an assembler name for an object or function is specified (see Section 6.43 [Asm Labels], page 417), the attribute must follow the `asm` specification.

An attribute specifier list may, in future, be permitted to appear after the declarator in a function definition (before any old-style parameter declarations or the function body).

Attribute specifiers may be mixed with type qualifiers appearing inside the `[]` of a parameter array declarator, in the C99 construct by which such qualifiers are applied to the pointer to which the array is implicitly converted. Such attribute specifiers apply to the pointer, not to the array, but at present this is not implemented and they are ignored.

An attribute specifier list may appear at the start of a nested declarator. At present, there are some limitations in this usage: the attributes correctly apply to the declarator, but for most individual attributes the semantics this implies are not implemented. When attribute specifiers follow the `*` of a pointer declarator, they may be mixed with any type qualifiers present. The following describes the formal semantics of this syntax. It will make the most sense if you are familiar with the formal specification of declarators in the ISO C standard.

Consider (as in C99 subclause 6.7.5 paragraph 4) a declaration `T D1`, where `T` contains declaration specifiers that specify a type *Type* (such as `int`) and `D1` is a declarator that contains an identifier *ident*. The type specified for *ident* for derived declarators whose type does not include an attribute specifier is as in the ISO C standard.

If `D1` has the form `( attribute-specifier-list D )`, and the declaration `T D` specifies the type "*derived-declarator-type-list Type*" for *ident*, then `T D1` specifies the type "*derived-declarator-type-list attribute-specifier-list Type*" for *ident*.

If `D1` has the form `* type-qualifier-and-attribute-specifier-list D`, and the declaration `T D` specifies the type "*derived-declarator-type-list Type*" for *ident*, then `T D1` specifies the type "*derived-declarator-type-list type-qualifier-and-attribute-specifier-list* pointer to *Type*" for *ident*.

For example,

```
void (__attribute__((noreturn)) ****f) (void);
```

specifies the type "pointer to pointer to pointer to pointer to non-returning function returning `void`". As another example,

```
char *__attribute__((aligned(8))) *f;
```

specifies the type "pointer to 8-byte-aligned pointer to `char`". Note again that this does not work with most attributes; for example, the usage of '`aligned`' and '`noreturn`' attributes given above is not yet supported.

For compatibility with existing code written for compiler versions that did not implement attributes on nested declarators, some laxity is allowed in the placing of attributes. If an attribute that only applies to types is applied to a declaration, it will be treated as applying to the type of that declaration. If an attribute that only applies to declarations is applied to the type of a declaration, it will be treated as applying to that declaration; and, for compatibility with code placing the attributes immediately before the identifier declared, such an attribute applied to a function return type will be treated as applying to the function type, and such an attribute applied to an array element type will be treated as applying to the array type. If an attribute that only applies to function types is applied to a pointer-to-function type, it will be treated as applying to the pointer target type; if such an attribute is applied to a function return type that is not a pointer-to-function type, it will be treated as applying to the function type.

6.32 Prototypes and Old-Style Function Definitions

GNU C extends ISO C to allow a function prototype to override a later old-style non-prototype definition. Consider the following example:

```
/* Use prototypes unless the compiler is old-fashioned.  */
#ifdef __STDC__
#define P(x) x
#else
#define P(x) ()
#endif

/* Prototype function declaration.  */
int isroot P((uid_t));

/* Old-style function definition.  */
int
isroot (x)   /* ??? lossage here ???  */
     uid_t x;
{
  return x == 0;
}
```

Suppose the type `uid_t` happens to be `short`. ISO C does not allow this example, because subword arguments in old-style non-prototype definitions are promoted. Therefore in this example the function definition's argument is really an `int`, which does not match the prototype argument type of `short`.

This restriction of ISO C makes it hard to write code that is portable to traditional C compilers, because the programmer does not know whether the `uid_t` type is `short`, `int`, or `long`. Therefore, in cases like these GNU C allows a prototype to override a later old-style definition. More precisely, in GNU C, a function prototype argument type overrides the argument type specified by a later old-style definition if the former type is the same as the latter type before promotion. Thus in GNU C the above example is equivalent to the following:

```
int isroot (uid_t);

int
isroot (uid_t x)
{
  return x == 0;
}
```

GNU C++ does not support old-style function definitions, so this extension is irrelevant.

6.33 C++ Style Comments

In GNU C, you may use C++ style comments, which start with '`//`' and continue until the end of the line. Many other C implementations allow such comments, and they are included in the 1999 C standard. However, C++ style comments are not recognized if you specify an '`-std`' option specifying a version of ISO C before C99, or '`-ansi`' (equivalent to '`-std=c90`').

6.34 Dollar Signs in Identifier Names

In GNU C, you may normally use dollar signs in identifier names. This is because many traditional C implementations allow such identifiers. However, dollar signs in identifiers are not supported on a few target machines, typically because the target assembler does not allow them.

6.35 The Character ESC in Constants

You can use the sequence '`\e`' in a string or character constant to stand for the ASCII character ESC.

6.36 Specifying Attributes of Variables

The keyword `__attribute__` allows you to specify special attributes of variables or structure fields. This keyword is followed by an attribute specification inside double parentheses. Some attributes are currently defined generically for variables. Other attributes are defined for variables on particular target systems. Other attributes are available for functions (see Section 6.30 [Function Attributes], page 332) and for types (see Section 6.37 [Type Attributes], page 374). Other front ends might define more attributes (see Chapter 7 [Extensions to the C++ Language], page 623).

You may also specify attributes with '`__`' preceding and following each keyword. This allows you to use them in header files without being concerned about a possible macro of the same name. For example, you may use `__aligned__` instead of `aligned`.

See Section 6.31 [Attribute Syntax], page 361, for details of the exact syntax for using attributes.

`aligned (`*`alignment`*`)`

This attribute specifies a minimum alignment for the variable or structure field, measured in bytes. For example, the declaration:

```
int x __attribute__ ((aligned (16))) = 0;
```

causes the compiler to allocate the global variable `x` on a 16-byte boundary. On a 68040, this could be used in conjunction with an `asm` expression to access the `move16` instruction which requires 16-byte aligned operands.

You can also specify the alignment of structure fields. For example, to create a double-word aligned `int` pair, you could write:

```
struct foo { int x[2] __attribute__ ((aligned (8))); };
```

This is an alternative to creating a union with a `double` member that forces the union to be double-word aligned.

As in the preceding examples, you can explicitly specify the alignment (in bytes) that you wish the compiler to use for a given variable or structure field. Alternatively, you can leave out the alignment factor and just ask the compiler to align a variable or field to the default alignment for the target architecture you are compiling for. The default alignment is sufficient for all scalar types, but may not be enough for all vector types on a target which supports vector operations. The default alignment is fixed for a particular target ABI.

Gcc also provides a target specific macro `__BIGGEST_ALIGNMENT__`, which is the largest alignment ever used for any data type on the target machine you are compiling for. For example, you could write:

```
short array[3] __attribute__ ((aligned (__BIGGEST_ALIGNMENT__)));
```

The compiler automatically sets the alignment for the declared variable or field to `__BIGGEST_ALIGNMENT__`. Doing this can often make copy operations more efficient, because the compiler can use whatever instructions copy the biggest chunks of memory when performing copies to or from the variables or fields that you have aligned this way. Note that the value of `__BIGGEST_ALIGNMENT__` may change depending on command line options.

When used on a struct, or struct member, the `aligned` attribute can only increase the alignment; in order to decrease it, the `packed` attribute must be specified as well. When used as part of a typedef, the `aligned` attribute can both increase and decrease alignment, and specifying the `packed` attribute will generate a warning.

Note that the effectiveness of `aligned` attributes may be limited by inherent limitations in your linker. On many systems, the linker is only able to arrange for variables to be aligned up to a certain maximum alignment. (For some linkers, the maximum supported alignment may be very very small.) If your linker is only able to align variables up to a maximum of 8 byte alignment, then specifying `aligned(16)` in an `__attribute__` will still only provide you with 8 byte alignment. See your linker documentation for further information.

The `aligned` attribute can also be used for functions (see Section 6.30 [Function Attributes], page 332.)

`cleanup (`*`cleanup_function`*`)`

The `cleanup` attribute runs a function when the variable goes out of scope. This attribute can only be applied to auto function scope variables; it may not be applied to parameters or variables with static storage duration. The function must take one parameter, a pointer to a type compatible with the variable. The return value of the function (if any) is ignored.

If '`-fexceptions`' is enabled, then *cleanup_function* will be run during the stack unwinding that happens during the processing of the exception. Note that the `cleanup` attribute does not allow the exception to be caught, only to perform an action. It is undefined what happens if *cleanup_function* does not return normally.

`common`
`nocommon`
: The `common` attribute requests GCC to place a variable in "common" storage. The `nocommon` attribute requests the opposite—to allocate space for it directly.

  These attributes override the default chosen by the '`-fno-common`' and '`-fcommon`' flags respectively.

`deprecated`
`deprecated (`*msg*`)`
: The `deprecated` attribute results in a warning if the variable is used anywhere in the source file. This is useful when identifying variables that are expected to be removed in a future version of a program. The warning also includes the location of the declaration of the deprecated variable, to enable users to easily find further information about why the variable is deprecated, or what they should do instead. Note that the warning only occurs for uses:

```
extern int old_var __attribute__ ((deprecated));
extern int old_var;
int new_fn () { return old_var; }
```

  results in a warning on line 3 but not line 2. The optional msg argument, which must be a string, will be printed in the warning if present.

  The `deprecated` attribute can also be used for functions and types (see Section 6.30 [Function Attributes], page 332, see Section 6.37 [Type Attributes], page 374.)

`mode (`*mode*`)`
: This attribute specifies the data type for the declaration—whichever type corresponds to the mode *mode*. This in effect lets you request an integer or floating point type according to its width.

  You may also specify a mode of '`byte`' or '`__byte__`' to indicate the mode corresponding to a one-byte integer, '`word`' or '`__word__`' for the mode of a one-word integer, and '`pointer`' or '`__pointer__`' for the mode used to represent pointers.

`packed`
: The `packed` attribute specifies that a variable or structure field should have the smallest possible alignment—one byte for a variable, and one bit for a field, unless you specify a larger value with the `aligned` attribute.

  Here is a structure in which the field `x` is packed, so that it immediately follows `a`:

```
struct foo
{
  char a;
  int x[2] __attribute__ ((packed));
};
```

  Note: 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. See the documentation of '`-Wpacked-bitfield-compat`' for more information.

`section ("`*section-name*`")`

Normally, the compiler places the objects it generates in sections like `data` and `bss`. Sometimes, however, you need additional sections, or you need certain particular variables to appear in special sections, for example to map to special hardware. The `section` attribute specifies that a variable (or function) lives in a particular section. For example, this small program uses several specific section names:

```
struct duart a __attribute__ ((section ("DUART_A"))) = { 0 };
struct duart b __attribute__ ((section ("DUART_B"))) = { 0 };
char stack[10000] __attribute__ ((section ("STACK"))) = { 0 };
int init_data __attribute__ ((section ("INITDATA")));

main()
{
  /* Initialize stack pointer */
  init_sp (stack + sizeof (stack));

  /* Initialize initialized data */
  memcpy (&init_data, &data, &edata - &data);

  /* Turn on the serial ports */
  init_duart (&a);
  init_duart (&b);
}
```

Use the `section` attribute with *global* variables and not *local* variables, as shown in the example.

You may use the `section` attribute with initialized or uninitialized global variables but the linker requires each object be defined once, with the exception that uninitialized variables tentatively go in the `common` (or `bss`) section and can be multiply "defined". Using the `section` attribute will change what section the variable goes into and may cause the linker to issue an error if an uninitialized variable has multiple definitions. You can force a variable to be initialized with the '`-fno-common`' flag or the `nocommon` attribute.

Some file formats do not support arbitrary sections so the `section` attribute is not available on all platforms. If you need to map the entire contents of a module to a particular section, consider using the facilities of the linker instead.

`shared`

On Microsoft Windows, in addition to putting variable definitions in a named section, the section can also be shared among all running copies of an executable or DLL. For example, this small program defines shared data by putting it in a named section `shared` and marking the section shareable:

```
int foo __attribute__((section ("shared"), shared)) = 0;

int
main()
{
  /* Read and write foo. All running
   copies see the same value.  */
  return 0;
```

```
}
```

You may only use the `shared` attribute along with `section` attribute with a fully initialized global definition because of the way linkers work. See `section` attribute for more information.

The `shared` attribute is only available on Microsoft Windows.

`tls_model ("tls_model")`
The `tls_model` attribute sets thread-local storage model (see Section 6.59 [Thread-Local], page 618) of a particular `__thread` variable, overriding '`-ftls-model=`' command-line switch on a per-variable basis. The *tls_model* argument should be one of `global-dynamic`, `local-dynamic`, `initial-exec` or `local-exec`.

Not all targets support this attribute.

`unused` This attribute, attached to a variable, means that the variable is meant to be possibly unused. GCC will not produce a warning for this variable.

`used` This attribute, attached to a variable, means that the variable must be emitted even if it appears that the variable is not referenced.

When applied to a static data member of a C++ class template, the attribute also means that the member will be instantiated if the class itself is instantiated.

`vector_size (bytes)`
This attribute specifies the vector size for the variable, measured in bytes. For example, the declaration:

```
int foo __attribute__ ((vector_size (16)));
```

causes the compiler to set the mode for `foo`, to be 16 bytes, divided into `int` sized units. Assuming a 32-bit int (a vector of 4 units of 4 bytes), the corresponding mode of `foo` will be V4SI.

This attribute is only applicable to integral and float scalars, although arrays, pointers, and function return values are allowed in conjunction with this construct.

Aggregates with this attribute are invalid, even if they are of the same size as a corresponding scalar. For example, the declaration:

```
struct S { int a; };
struct S  __attribute__ ((vector_size (16))) foo;
```

is invalid even if the size of the structure is the same as the size of the `int`.

`selectany`
The `selectany` attribute causes an initialized global variable to have link-once semantics. When multiple definitions of the variable are encountered by the linker, the first is selected and the remainder are discarded. Following usage by the Microsoft compiler, the linker is told *not* to warn about size or content differences of the multiple definitions.

Although the primary usage of this attribute is for POD types, the attribute can also be applied to global C++ objects that are initialized by a constructor. In this case, the static initialization and destruction code for the object is emitted

in each translation defining the object, but the calls to the constructor and destructor are protected by a link-once guard variable.

The `selectany` attribute is only available on Microsoft Windows targets. You can use `__declspec (selectany)` as a synonym for `__attribute__ ((selectany))` for compatibility with other compilers.

`weak` The `weak` attribute is described in Section 6.30 [Function Attributes], page 332.

`dllimport`
The `dllimport` attribute is described in Section 6.30 [Function Attributes], page 332.

`dllexport`
The `dllexport` attribute is described in Section 6.30 [Function Attributes], page 332.

6.36.1 AVR Variable Attributes

`progmem` The `progmem` attribute is used on the AVR to place read-only data in the non-volatile program memory (flash). The `progmem` attribute accomplishes this by putting respective variables into a section whose name starts with `.progmem`.

This attribute works similar to the `section` attribute but adds additional checking. Notice that just like the `section` attribute, `progmem` affects the location of the data but not how this data is accessed.

In order to read data located with the `progmem` attribute (inline) assembler must be used.

```
/* Use custom macros from AVR-LibC (http://nongnu.org/avr-libc/user-manual)
#include <avr/pgmspace.h>

/* Locate var in flash memory */
const int var[2] PROGMEM = { 1, 2 };

int read_var (int i)
{
    /* Access var[] by accessor macro from avr/pgmspace.h */
    return (int) pgm_read_word (& var[i]);
}
```

AVR is a Harvard architecture processor and data and read-only data normally resides in the data memory (RAM).

See also the [AVR Named Address Spaces], page 322 section for an alternate way to locate and access data in flash memory.

6.36.2 Blackfin Variable Attributes

Three attributes are currently defined for the Blackfin.

`l1_data`
`l1_data_A`
`l1_data_B`

Use these attributes on the Blackfin to place the variable into L1 Data SRAM. Variables with `l1_data` attribute will be put into the specific section named `.l1.data`. Those with `l1_data_A` attribute will be put into the specific section named `.l1.data.A`. Those with `l1_data_B` attribute will be put into the specific section named `.l1.data.B`.

`l2` Use this attribute on the Blackfin to place the variable into L2 SRAM. Variables with `l2` attribute will be put into the specific section named `.l2.data`.

6.36.3 M32R/D Variable Attributes

One attribute is currently defined for the M32R/D.

`model (`*model-name*`)`

Use this attribute on the M32R/D to set the addressability of an object. The identifier *model-name* is one of `small`, `medium`, or `large`, representing each of the code models.

Small model objects live in the lower 16MB of memory (so that their addresses can be loaded with the `ld24` instruction).

Medium and large model objects may live anywhere in the 32-bit address space (the compiler will generate `seth/add3` instructions to load their addresses).

6.36.4 MeP Variable Attributes

The MeP target has a number of addressing modes and busses. The `near` space spans the standard memory space's first 16 megabytes (24 bits). The `far` space spans the entire 32-bit memory space. The `based` space is a 128 byte region in the memory space which is addressed relative to the `$tp` register. The `tiny` space is a 65536 byte region relative to the `$gp` register. In addition to these memory regions, the MeP target has a separate 16-bit control bus which is specified with `cb` attributes.

`based` Any variable with the `based` attribute will be assigned to the `.based` section, and will be accessed with relative to the `$tp` register.

`tiny` Likewise, the `tiny` attribute assigned variables to the `.tiny` section, relative to the `$gp` register.

`near` Variables with the `near` attribute are assumed to have addresses that fit in a 24-bit addressing mode. This is the default for large variables (`-mtiny=4` is the default) but this attribute can override `-mtiny=` for small variables, or override `-ml`.

`far` Variables with the `far` attribute are addressed using a full 32-bit address. Since this covers the entire memory space, this allows modules to make no assumptions about where variables might be stored.

`io`
`io (`*addr*`)`

Variables with the `io` attribute are used to address memory-mapped peripherals. If an address is specified, the variable is assigned that address, else it is not

assigned an address (it is assumed some other module will assign an address). Example:

```
int timer_count __attribute__((io(0x123)));
```

`cb`
`cb (addr)`

Variables with the `cb` attribute are used to access the control bus, using special instructions. `addr` indicates the control bus address. Example:

```
int cpu_clock __attribute__((cb(0x123)));
```

6.36.5 i386 Variable Attributes

Two attributes are currently defined for i386 configurations: `ms_struct` and `gcc_struct`

`ms_struct`
`gcc_struct`

If `packed` is used on a structure, or if bit-fields are used it may be that the Microsoft ABI packs them differently than GCC would normally pack them. Particularly when moving packed data between functions compiled with GCC and the native Microsoft compiler (either via function call or as data in a file), it may be necessary to access either format.

Currently '`-m[no-]ms-bitfields`' is provided for the Microsoft Windows X86 compilers to match the native Microsoft compiler.

The Microsoft structure layout algorithm is fairly simple with the exception of the bitfield packing:

The padding and alignment of members of structures and whether a bit field can straddle a storage-unit boundary

1. Structure members are stored sequentially in the order in which they are declared: the first member has the lowest memory address and the last member the highest.
2. Every data object has an alignment-requirement. The alignment-requirement for all data except structures, unions, and arrays is either the size of the object or the current packing size (specified with either the aligned attribute or the pack pragma), whichever is less. For structures, unions, and arrays, the alignment-requirement is the largest alignment-requirement of its members. Every object is allocated an offset so that:

   offset % alignment-requirement == 0
3. Adjacent bit fields are packed into the same 1-, 2-, or 4-byte allocation unit if the integral types are the same size and if the next bit field fits into the current allocation unit without crossing the boundary imposed by the common alignment requirements of the bit fields.

Handling of zero-length bitfields:

MSVC interprets zero-length bitfields in the following ways:

1. If a zero-length bitfield is inserted between two bitfields that would normally be coalesced, the bitfields will not be coalesced.

For example:

```
struct
 {
   unsigned long bf_1 : 12;
   unsigned long : 0;
   unsigned long bf_2 : 12;
 } t1;
```

The size of `t1` would be 8 bytes with the zero-length bitfield. If the zero-length bitfield were removed, `t1`'s size would be 4 bytes.

2. If a zero-length bitfield is inserted after a bitfield, `foo`, and the alignment of the zero-length bitfield is greater than the member that follows it, `bar`, `bar` will be aligned as the type of the zero-length bitfield.

   For example:

   ```
   struct
    {
      char foo : 4;
      short : 0;
      char bar;
    } t2;

   struct
    {
      char foo : 4;
      short : 0;
      double bar;
    } t3;
   ```

   For `t2`, `bar` will be placed at offset 2, rather than offset 1. Accordingly, the size of `t2` will be 4. For `t3`, the zero-length bitfield will not affect the alignment of `bar` or, as a result, the size of the structure.

   Taking this into account, it is important to note the following:

   1. If a zero-length bitfield follows a normal bitfield, the type of the zero-length bitfield may affect the alignment of the structure as whole. For example, `t2` has a size of 4 bytes, since the zero-length bitfield follows a normal bitfield, and is of type short.
   2. Even if a zero-length bitfield is not followed by a normal bitfield, it may still affect the alignment of the structure:

      ```
      struct
       {
         char foo : 6;
         long : 0;
       } t4;
      ```

      Here, `t4` will take up 4 bytes.

3. Zero-length bitfields following non-bitfield members are ignored:

   ```
   struct
    {
      char foo;
      long : 0;
      char bar;
    } t5;
   ```

   Here, `t5` will take up 2 bytes.

6.36.6 PowerPC Variable Attributes

Three attributes currently are defined for PowerPC configurations: `altivec`, `ms_struct` and `gcc_struct`.

For full documentation of the struct attributes please see the documentation in [i386 Variable Attributes], page 372.

For documentation of `altivec` attribute please see the documentation in [PowerPC Type Attributes], page 379.

6.36.7 SPU Variable Attributes

The SPU supports the `spu_vector` attribute for variables. For documentation of this attribute please see the documentation in [SPU Type Attributes], page 379.

6.36.8 Xstormy16 Variable Attributes

One attribute is currently defined for xstormy16 configurations: `below100`.

`below100`

If a variable has the `below100` attribute (`BELOW100` is allowed also), GCC will place the variable in the first 0x100 bytes of memory and use special opcodes to access it. Such variables will be placed in either the `.bss_below100` section or the `.data_below100` section.

6.37 Specifying Attributes of Types

The keyword `__attribute__` allows you to specify special attributes of `struct` and `union` types when you define such types. This keyword is followed by an attribute specification inside double parentheses. Seven attributes are currently defined for types: `aligned`, `packed`, `transparent_union`, `unused`, `deprecated`, `visibility`, and `may_alias`. Other attributes are defined for functions (see Section 6.30 [Function Attributes], page 332) and for variables (see Section 6.36 [Variable Attributes], page 365).

You may also specify any one of these attributes with '`__`' preceding and following its keyword. This allows you to use these attributes in header files without being concerned about a possible macro of the same name. For example, you may use `__aligned__` instead of `aligned`.

You may specify type attributes in an enum, struct or union type declaration or definition, or for other types in a `typedef` declaration.

For an enum, struct or union type, you may specify attributes either between the enum, struct or union tag and the name of the type, or just past the closing curly brace of the *definition*. The former syntax is preferred.

See Section 6.31 [Attribute Syntax], page 361, for details of the exact syntax for using attributes.

`aligned (`*`alignment`*`)`

This attribute specifies a minimum alignment (in bytes) for variables of the specified type. For example, the declarations:

```
struct S { short f[3]; } __attribute__ ((aligned (8)));
typedef int more_aligned_int __attribute__ ((aligned (8)));
```

force the compiler to insure (as far as it can) that each variable whose type is `struct S` or `more_aligned_int` will be allocated and aligned *at least* on a 8-byte boundary. On a SPARC, having all variables of type `struct S` aligned to 8-byte boundaries allows the compiler to use the `ldd` and `std` (doubleword load and store) instructions when copying one variable of type `struct S` to another, thus improving run-time efficiency.

Note that the alignment of any given `struct` or `union` type is required by the ISO C standard to be at least a perfect multiple of the lowest common multiple of the alignments of all of the members of the `struct` or `union` in question. This means that you *can* effectively adjust the alignment of a `struct` or `union` type by attaching an `aligned` attribute to any one of the members of such a type, but the notation illustrated in the example above is a more obvious, intuitive, and readable way to request the compiler to adjust the alignment of an entire `struct` or `union` type.

As in the preceding example, you can explicitly specify the alignment (in bytes) that you wish the compiler to use for a given `struct` or `union` type. Alternatively, you can leave out the alignment factor and just ask the compiler to align a type to the maximum useful alignment for the target machine you are compiling for. For example, you could write:

```
struct S { short f[3]; } __attribute__ ((aligned));
```

Whenever you leave out the alignment factor in an `aligned` attribute specification, the compiler automatically sets the alignment for the type to the largest alignment which is ever used for any data type on the target machine you are compiling for. Doing this can often make copy operations more efficient, because the compiler can use whatever instructions copy the biggest chunks of memory when performing copies to or from the variables which have types that you have aligned this way.

In the example above, if the size of each `short` is 2 bytes, then the size of the entire `struct S` type is 6 bytes. The smallest power of two which is greater than or equal to that is 8, so the compiler sets the alignment for the entire `struct S` type to 8 bytes.

Note that although you can ask the compiler to select a time-efficient alignment for a given type and then declare only individual stand-alone objects of that type, the compiler's ability to select a time-efficient alignment is primarily useful only when you plan to create arrays of variables having the relevant (efficiently aligned) type. If you declare or use arrays of variables of an efficiently-aligned type, then it is likely that your program will also be doing pointer arithmetic (or subscripting, which amounts to the same thing) on pointers to the relevant type, and the code that the compiler generates for these pointer arithmetic operations will often be more efficient for efficiently-aligned types than for other types.

The `aligned` attribute can only increase the alignment; but you can decrease it by specifying `packed` as well. See below.

Note that the effectiveness of `aligned` attributes may be limited by inherent limitations in your linker. On many systems, the linker is only able to arrange for variables to be aligned up to a certain maximum alignment. (For some

linkers, the maximum supported alignment may be very very small.) If your linker is only able to align variables up to a maximum of 8 byte alignment, then specifying `aligned(16)` in an `__attribute__` will still only provide you with 8 byte alignment. See your linker documentation for further information.

`packed` This attribute, attached to `struct` or `union` type definition, specifies that each member (other than zero-width bitfields) of the structure or union is placed to minimize the memory required. When attached to an `enum` definition, it indicates that the smallest integral type should be used.

Specifying this attribute for `struct` and `union` types is equivalent to specifying the `packed` attribute on each of the structure or union members. Specifying the '`-fshort-enums`' flag on the line is equivalent to specifying the `packed` attribute on all `enum` definitions.

In the following example `struct my_packed_struct`'s members are packed closely together, but the internal layout of its `s` member is not packed—to do that, `struct my_unpacked_struct` would need to be packed too.

```
struct my_unpacked_struct
 {
    char c;
    int i;
 };

struct __attribute__ ((__packed__)) my_packed_struct
  {
     char c;
     int  i;
     struct my_unpacked_struct s;
  };
```

You may only specify this attribute on the definition of an `enum`, `struct` or `union`, not on a `typedef` which does not also define the enumerated type, structure or union.

`transparent_union`

This attribute, attached to a `union` type definition, indicates that any function parameter having that union type causes calls to that function to be treated in a special way.

First, the argument corresponding to a transparent union type can be of any type in the union; no cast is required. Also, if the union contains a pointer type, the corresponding argument can be a null pointer constant or a void pointer expression; and if the union contains a void pointer type, the corresponding argument can be any pointer expression. If the union member type is a pointer, qualifiers like `const` on the referenced type must be respected, just as with normal pointer conversions.

Second, the argument is passed to the function using the calling conventions of the first member of the transparent union, not the calling conventions of the union itself. All members of the union must have the same machine representation; this is necessary for this argument passing to work properly.

Transparent unions are designed for library functions that have multiple interfaces for compatibility reasons. For example, suppose the `wait` function must

accept either a value of type `int *` to comply with Posix, or a value of type `union wait *` to comply with the 4.1BSD interface. If `wait`'s parameter were `void *`, `wait` would accept both kinds of arguments, but it would also accept any other pointer type and this would make argument type checking less useful. Instead, `<sys/wait.h>` might define the interface as follows:

```
typedef union __attribute__ ((__transparent_union__))
  {
    int *__ip;
    union wait *__up;
  } wait_status_ptr_t;

pid_t wait (wait_status_ptr_t);
```

This interface allows either `int *` or `union wait *` arguments to be passed, using the `int *` calling convention. The program can call `wait` with arguments of either type:

```
int w1 () { int w; return wait (&w); }
int w2 () { union wait w; return wait (&w); }
```

With this interface, `wait`'s implementation might look like this:

```
pid_t wait (wait_status_ptr_t p)
{
  return waitpid (-1, p.__ip, 0);
}
```

`unused` When attached to a type (including a `union` or a `struct`), this attribute means that variables of that type are meant to appear possibly unused. GCC will not produce a warning for any variables of that type, even if the variable appears to do nothing. This is often the case with lock or thread classes, which are usually defined and then not referenced, but contain constructors and destructors that have nontrivial bookkeeping functions.

`deprecated`
`deprecated (`*`msg`*`)`

The `deprecated` attribute results in a warning if the type is used anywhere in the source file. This is useful when identifying types that are expected to be removed in a future version of a program. If possible, the warning also includes the location of the declaration of the deprecated type, to enable users to easily find further information about why the type is deprecated, or what they should do instead. Note that the warnings only occur for uses and then only if the type is being applied to an identifier that itself is not being declared as deprecated.

```
typedef int T1 __attribute__ ((deprecated));
T1 x;
typedef T1 T2;
T2 y;
typedef T1 T3 __attribute__ ((deprecated));
T3 z __attribute__ ((deprecated));
```

results in a warning on line 2 and 3 but not lines 4, 5, or 6. No warning is issued for line 4 because T2 is not explicitly deprecated. Line 5 has no warning because T3 is explicitly deprecated. Similarly for line 6. The optional msg argument, which must be a string, will be printed in the warning if present.

The `deprecated` attribute can also be used for functions and variables (see Section 6.30 [Function Attributes], page 332, see Section 6.36 [Variable Attributes], page 365.)

`may_alias`

Accesses through pointers to types with this attribute are not subject to type-based alias analysis, but are instead assumed to be able to alias any other type of objects. In the context of 6.5/7 an lvalue expression dereferencing such a pointer is treated like having a character type. See '`-fstrict-aliasing`' for more information on aliasing issues. This extension exists to support some vector APIs, in which pointers to one vector type are permitted to alias pointers to a different vector type.

Note that an object of a type with this attribute does not have any special semantics.

Example of use:

```
typedef short __attribute__((__may_alias__)) short_a;

int
main (void)
{
  int a = 0x12345678;
  short_a *b = (short_a *) &a;

  b[1] = 0;

  if (a == 0x12345678)
    abort();

  exit(0);
}
```

If you replaced `short_a` with `short` in the variable declaration, the above program would abort when compiled with '`-fstrict-aliasing`', which is on by default at '`-O2`' or above in recent GCC versions.

`visibility`

In C++, attribute visibility (see Section 6.30 [Function Attributes], page 332) can also be applied to class, struct, union and enum types. Unlike other type attributes, the attribute must appear between the initial keyword and the name of the type; it cannot appear after the body of the type.

Note that the type visibility is applied to vague linkage entities associated with the class (vtable, typeinfo node, etc.). In particular, if a class is thrown as an exception in one shared object and caught in another, the class must have default visibility. Otherwise the two shared objects will be unable to use the same typeinfo node and exception handling will break.

6.37.1 ARM Type Attributes

On those ARM targets that support `dllimport` (such as Symbian OS), you can use the `notshared` attribute to indicate that the virtual table and other similar data for a class should not be exported from a DLL. For example:

```
class __declspec(notshared) C {
```

```
public:
  __declspec(dllimport) C();
  virtual void f();
}

__declspec(dllexport)
C::C() {}
```

In this code, `C::C` is exported from the current DLL, but the virtual table for `C` is not exported. (You can use `__attribute__` instead of `__declspec` if you prefer, but most Symbian OS code uses `__declspec`.)

6.37.2 MeP Type Attributes

Many of the MeP variable attributes may be applied to types as well. Specifically, the `based`, `tiny`, `near`, and `far` attributes may be applied to either. The `io` and `cb` attributes may not be applied to types.

6.37.3 i386 Type Attributes

Two attributes are currently defined for i386 configurations: `ms_struct` and `gcc_struct`.

`ms_struct`
`gcc_struct`

> If `packed` is used on a structure, or if bit-fields are used it may be that the Microsoft ABI packs them differently than GCC would normally pack them. Particularly when moving packed data between functions compiled with GCC and the native Microsoft compiler (either via function call or as data in a file), it may be necessary to access either format.
>
> Currently '`-m[no-]ms-bitfields`' is provided for the Microsoft Windows X86 compilers to match the native Microsoft compiler.

To specify multiple attributes, separate them by commas within the double parentheses: for example, '`__attribute__ ((aligned (16), packed))`'.

6.37.4 PowerPC Type Attributes

Three attributes currently are defined for PowerPC configurations: `altivec`, `ms_struct` and `gcc_struct`.

For full documentation of the `ms_struct` and `gcc_struct` attributes please see the documentation in [i386 Type Attributes], page 379.

The `altivec` attribute allows one to declare AltiVec vector data types supported by the AltiVec Programming Interface Manual. The attribute requires an argument to specify one of three vector types: `vector__`, `pixel__` (always followed by unsigned short), and `bool__` (always followed by unsigned).

```
__attribute__((altivec(vector__)))
__attribute__((altivec(pixel__))) unsigned short
__attribute__((altivec(bool__))) unsigned
```

These attributes mainly are intended to support the `__vector`, `__pixel`, and `__bool` AltiVec keywords.

6.37.5 SPU Type Attributes

The SPU supports the `spu_vector` attribute for types. This attribute allows one to declare vector data types supported by the Sony/Toshiba/IBM SPU Language Extensions Specification. It is intended to support the `__vector` keyword.

6.38 Inquiring on Alignment of Types or Variables

The keyword `__alignof__` allows you to inquire about how an object is aligned, or the minimum alignment usually required by a type. Its syntax is just like `sizeof`.

For example, if the target machine requires a `double` value to be aligned on an 8-byte boundary, then `__alignof__ (double)` is 8. This is true on many RISC machines. On more traditional machine designs, `__alignof__ (double)` is 4 or even 2.

Some machines never actually require alignment; they allow reference to any data type even at an odd address. For these machines, `__alignof__` reports the smallest alignment that GCC will give the data type, usually as mandated by the target ABI.

If the operand of `__alignof__` is an lvalue rather than a type, its value is the required alignment for its type, taking into account any minimum alignment specified with GCC's `__attribute__` extension (see Section 6.36 [Variable Attributes], page 365). For example, after this declaration:

```
struct foo { int x; char y; } foo1;
```

the value of `__alignof__ (foo1.y)` is 1, even though its actual alignment is probably 2 or 4, the same as `__alignof__ (int)`.

It is an error to ask for the alignment of an incomplete type.

6.39 An Inline Function is As Fast As a Macro

By declaring a function inline, you can direct GCC to make calls to that function faster. One way GCC can achieve this is to integrate that function's code into the code for its callers. This makes execution faster by eliminating the function-call overhead; in addition, if any of the actual argument values are constant, their known values may permit simplifications at compile time so that not all of the inline function's code needs to be included. The effect on code size is less predictable; object code may be larger or smaller with function inlining, depending on the particular case. You can also direct GCC to try to integrate all "simple enough" functions into their callers with the option '`-finline-functions`'.

GCC implements three different semantics of declaring a function inline. One is available with '`-std=gnu89`' or '`-fgnu89-inline`' or when `gnu_inline` attribute is present on all inline declarations, another when '`-std=c99`', '`-std=c11`', '`-std=gnu99`' or '`-std=gnu11`' (without '`-fgnu89-inline`'), and the third is used when compiling C++.

To declare a function inline, use the `inline` keyword in its declaration, like this:

```
static inline int
inc (int *a)
{
  return (*a)++;
}
```

If you are writing a header file to be included in ISO C90 programs, write `__inline__` instead of `inline`. See Section 6.45 [Alternate Keywords], page 420.

The three types of inlining behave similarly in two important cases: when the `inline` keyword is used on a `static` function, like the example above, and when a function is first declared without using the `inline` keyword and then is defined with `inline`, like this:

```
extern int inc (int *a);
inline int
inc (int *a)
{
  return (*a)++;
}
```

In both of these common cases, the program behaves the same as if you had not used the `inline` keyword, except for its speed.

When a function is both inline and `static`, if all calls to the function are integrated into the caller, and the function's address is never used, then the function's own assembler code is never referenced. In this case, GCC does not actually output assembler code for the function, unless you specify the option '`-fkeep-inline-functions`'. Some calls cannot be integrated for various reasons (in particular, calls that precede the function's definition cannot be integrated, and neither can recursive calls within the definition). If there is a nonintegrated call, then the function is compiled to assembler code as usual. The function must also be compiled as usual if the program refers to its address, because that can't be inlined.

Note that certain usages in a function definition can make it unsuitable for inline substitution. Among these usages are: use of varargs, use of alloca, use of variable sized data types (see Section 6.19 [Variable Length], page 326), use of computed goto (see Section 6.3 [Labels as Values], page 311), use of nonlocal goto, and nested functions (see Section 6.4 [Nested Functions], page 312). Using '`-Winline`' will warn when a function marked `inline` could not be substituted, and will give the reason for the failure.

As required by ISO C++, GCC considers member functions defined within the body of a class to be marked inline even if they are not explicitly declared with the `inline` keyword. You can override this with '`-fno-default-inline`'; see Section 3.5 [Options Controlling C++ Dialect], page 35.

GCC does not inline any functions when not optimizing unless you specify the '`always_inline`' attribute for the function, like this:

```
/* Prototype.  */
inline void foo (const char) __attribute__((always_inline));
```

The remainder of this section is specific to GNU C90 inlining.

When an inline function is not `static`, then the compiler must assume that there may be calls from other source files; since a global symbol can be defined only once in any program, the function must not be defined in the other source files, so the calls therein cannot be integrated. Therefore, a non-`static` inline function is always compiled on its own in the usual fashion.

If you specify both `inline` and `extern` in the function definition, then the definition is used only for inlining. In no case is the function compiled on its own, not even if you refer to its address explicitly. Such an address becomes an external reference, as if you had only declared the function, and had not defined it.

This combination of `inline` and `extern` has almost the effect of a macro. The way to use it is to put a function definition in a header file with these keywords, and put another

copy of the definition (lacking `inline` and `extern`) in a library file. The definition in the header file will cause most calls to the function to be inlined. If any uses of the function remain, they will refer to the single copy in the library.

6.40 When is a Volatile Object Accessed?

C has the concept of volatile objects. These are normally accessed by pointers and used for accessing hardware or inter-thread communication. The standard encourages compilers to refrain from optimizations concerning accesses to volatile objects, but leaves it implementation defined as to what constitutes a volatile access. The minimum requirement is that at a sequence point all previous accesses to volatile objects have stabilized and no subsequent accesses have occurred. Thus an implementation is free to reorder and combine volatile accesses which occur between sequence points, but cannot do so for accesses across a sequence point. The use of volatile does not allow you to violate the restriction on updating objects multiple times between two sequence points.

Accesses to non-volatile objects are not ordered with respect to volatile accesses. You cannot use a volatile object as a memory barrier to order a sequence of writes to non-volatile memory. For instance:

```
int *ptr = something;
volatile int vobj;
*ptr = something;
vobj = 1;
```

Unless **ptr* and *vobj* can be aliased, it is not guaranteed that the write to **ptr* will have occurred by the time the update of *vobj* has happened. If you need this guarantee, you must use a stronger memory barrier such as:

```
int *ptr = something;
volatile int vobj;
*ptr = something;
asm volatile ("" : : : "memory");
vobj = 1;
```

A scalar volatile object is read when it is accessed in a void context:

```
volatile int *src = somevalue;
*src;
```

Such expressions are rvalues, and GCC implements this as a read of the volatile object being pointed to.

Assignments are also expressions and have an rvalue. However when assigning to a scalar volatile, the volatile object is not reread, regardless of whether the assignment expression's rvalue is used or not. If the assignment's rvalue is used, the value is that assigned to the volatile object. For instance, there is no read of *vobj* in all the following cases:

```
int obj;
volatile int vobj;
vobj = something;
obj = vobj = something;
obj ? vobj = onething : vobj = anotherthing;
obj = (something, vobj = anotherthing);
```

If you need to read the volatile object after an assignment has occurred, you must use a separate expression with an intervening sequence point.

As bitfields are not individually addressable, volatile bitfields may be implicitly read when written to, or when adjacent bitfields are accessed. Bitfield operations may be optimized

such that adjacent bitfields are only partially accessed, if they straddle a storage unit boundary. For these reasons it is unwise to use volatile bitfields to access hardware.

6.41 Assembler Instructions with C Expression Operands

In an assembler instruction using `asm`, you can specify the operands of the instruction using C expressions. This means you need not guess which registers or memory locations will contain the data you want to use.

You must specify an assembler instruction template much like what appears in a machine description, plus an operand constraint string for each operand.

For example, here is how to use the 68881's `fsinx` instruction:

```
asm ("fsinx %1,%0" : "=f" (result) : "f" (angle));
```

Here `angle` is the C expression for the input operand while `result` is that of the output operand. Each has '`"f"`' as its operand constraint, saying that a floating point register is required. The '`=`' in '`=f`' indicates that the operand is an output; all output operands' constraints must use '`=`'. The constraints use the same language used in the machine description (see Section 6.42 [Constraints], page 390).

Each operand is described by an operand-constraint string followed by the C expression in parentheses. A colon separates the assembler template from the first output operand and another separates the last output operand from the first input, if any. Commas separate the operands within each group. The total number of operands is currently limited to 30; this limitation may be lifted in some future version of GCC.

If there are no output operands but there are input operands, you must place two consecutive colons surrounding the place where the output operands would go.

As of GCC version 3.1, it is also possible to specify input and output operands using symbolic names which can be referenced within the assembler code. These names are specified inside square brackets preceding the constraint string, and can be referenced inside the assembler code using `%[`*name*`]` instead of a percentage sign followed by the operand number. Using named operands the above example could look like:

```
asm ("fsinx %[angle],%[output]"
     : [output] "=f" (result)
     : [angle] "f" (angle));
```

Note that the symbolic operand names have no relation whatsoever to other C identifiers. You may use any name you like, even those of existing C symbols, but you must ensure that no two operands within the same assembler construct use the same symbolic name.

Output operand expressions must be lvalues; the compiler can check this. The input operands need not be lvalues. The compiler cannot check whether the operands have data types that are reasonable for the instruction being executed. It does not parse the assembler instruction template and does not know what it means or even whether it is valid assembler input. The extended `asm` feature is most often used for machine instructions the compiler itself does not know exist. If the output expression cannot be directly addressed (for example, it is a bit-field), your constraint must allow a register. In that case, GCC will use the register as the output of the `asm`, and then store that register into the output.

The ordinary output operands must be write-only; GCC will assume that the values in these operands before the instruction are dead and need not be generated. Extended asm supports input-output or read-write operands. Use the constraint character '`+`' to indicate

such an operand and list it with the output operands. You should only use read-write operands when the constraints for the operand (or the operand in which only some of the bits are to be changed) allow a register.

You may, as an alternative, logically split its function into two separate operands, one input operand and one write-only output operand. The connection between them is expressed by constraints which say they need to be in the same location when the instruction executes. You can use the same C expression for both operands, or different expressions. For example, here we write the (fictitious) '`combine`' instruction with `bar` as its read-only source operand and `foo` as its read-write destination:

```
asm ("combine %2,%0" : "=r" (foo) : "0" (foo), "g" (bar));
```

The constraint '`"0"`' for operand 1 says that it must occupy the same location as operand 0. A number in constraint is allowed only in an input operand and it must refer to an output operand.

Only a number in the constraint can guarantee that one operand will be in the same place as another. The mere fact that `foo` is the value of both operands is not enough to guarantee that they will be in the same place in the generated assembler code. The following would not work reliably:

```
asm ("combine %2,%0" : "=r" (foo) : "r" (foo), "g" (bar));
```

Various optimizations or reloading could cause operands 0 and 1 to be in different registers; GCC knows no reason not to do so. For example, the compiler might find a copy of the value of `foo` in one register and use it for operand 1, but generate the output operand 0 in a different register (copying it afterward to `foo`'s own address). Of course, since the register for operand 1 is not even mentioned in the assembler code, the result will not work, but GCC can't tell that.

As of GCC version 3.1, one may write `[`*name*`]` instead of the operand number for a matching constraint. For example:

```
asm ("cmoveq %1,%2,%[result]"
     : [result] "=r"(result)
     : "r" (test), "r"(new), "[result]"(old));
```

Sometimes you need to make an `asm` operand be a specific register, but there's no matching constraint letter for that register *by itself*. To force the operand into that register, use a local variable for the operand and specify the register in the variable declaration. See Section 6.44 [Explicit Reg Vars], page 418. Then for the `asm` operand, use any register constraint letter that matches the register:

```
register int *p1 asm ("r0") = ...;
register int *p2 asm ("r1") = ...;
register int *result asm ("r0");
asm ("sysint" : "=r" (result) : "0" (p1), "r" (p2));
```

In the above example, beware that a register that is call-clobbered by the target ABI will be overwritten by any function call in the assignment, including library calls for arithmetic operators. Also a register may be clobbered when generating some operations, like variable shift, memory copy or memory move on x86. Assuming it is a call-clobbered register, this may happen to `r0` above by the assignment to `p2`. If you have to use such a register, use temporary variables for expressions between the register assignment and use:

```
int t1 = ...;
register int *p1 asm ("r0") = ...;
```

```
register int *p2 asm ("r1") = t1;
register int *result asm ("r0");
asm ("sysint" : "=r" (result) : "0" (p1), "r" (p2));
```

Some instructions clobber specific hard registers. To describe this, write a third colon after the input operands, followed by the names of the clobbered hard registers (given as strings). Here is a realistic example for the VAX:

```
asm volatile ("movc3 %0,%1,%2"
              : /* no outputs */
              : "g" (from), "g" (to), "g" (count)
              : "r0", "r1", "r2", "r3", "r4", "r5");
```

You may not write a clobber description in a way that overlaps with an input or output operand. For example, you may not have an operand describing a register class with one member if you mention that register in the clobber list. Variables declared to live in specific registers (see Section 6.44 [Explicit Reg Vars], page 418), and used as asm input or output operands must have no part mentioned in the clobber description. There is no way for you to specify that an input operand is modified without also specifying it as an output operand. Note that if all the output operands you specify are for this purpose (and hence unused), you will then also need to specify `volatile` for the `asm` construct, as described below, to prevent GCC from deleting the `asm` statement as unused.

If you refer to a particular hardware register from the assembler code, you will probably have to list the register after the third colon to tell the compiler the register's value is modified. In some assemblers, the register names begin with '`%`'; to produce one '`%`' in the assembler code, you must write '`%%`' in the input.

If your assembler instruction can alter the condition code register, add '`cc`' to the list of clobbered registers. GCC on some machines represents the condition codes as a specific hardware register; '`cc`' serves to name this register. On other machines, the condition code is handled differently, and specifying '`cc`' has no effect. But it is valid no matter what the machine.

If your assembler instructions access memory in an unpredictable fashion, add '`memory`' to the list of clobbered registers. This will cause GCC to not keep memory values cached in registers across the assembler instruction and not optimize stores or loads to that memory. You will also want to add the `volatile` keyword if the memory affected is not listed in the inputs or outputs of the `asm`, as the '`memory`' clobber does not count as a side-effect of the `asm`. If you know how large the accessed memory is, you can add it as input or output but if this is not known, you should add '`memory`'. As an example, if you access ten bytes of a string, you can use a memory input like:

```
{"m"( ({ struct { char x[10]; } *p = (void *)ptr ; *p; }) )}.
```

Note that in the following example the memory input is necessary, otherwise GCC might optimize the store to `x` away:

```
int foo ()
{
  int x = 42;
  int *y = &x;
  int result;
  asm ("magic stuff accessing an 'int' pointed to by '%1'"
        "=&d" (r) : "a" (y), "m" (*y));
  return result;
}
```

You can put multiple assembler instructions together in a single `asm` template, separated by the characters normally used in assembly code for the system. A combination that works in most places is a newline to break the line, plus a tab character to move to the instruction field (written as '`\n\t`'). Sometimes semicolons can be used, if the assembler allows semicolons as a line-breaking character. Note that some assembler dialects use semicolons to start a comment. The input operands are guaranteed not to use any of the clobbered registers, and neither will the output operands' addresses, so you can read and write the clobbered registers as many times as you like. Here is an example of multiple instructions in a template; it assumes the subroutine `_foo` accepts arguments in registers 9 and 10:

```
asm ("movl %0,r9\n\tmovl %1,r10\n\tcall _foo"
     : /* no outputs */
     : "g" (from), "g" (to)
     : "r9", "r10");
```

Unless an output operand has the '`&`' constraint modifier, GCC may allocate it in the same register as an unrelated input operand, on the assumption the inputs are consumed before the outputs are produced. This assumption may be false if the assembler code actually consists of more than one instruction. In such a case, use '`&`' for each output operand that may not overlap an input. See Section 6.42.3 [Modifiers], page 393.

If you want to test the condition code produced by an assembler instruction, you must include a branch and a label in the `asm` construct, as follows:

```
asm ("clr %0\n\tfrob %1\n\tbeq 0f\n\tmov #1,%0\n0:"
     : "g" (result)
     : "g" (input));
```

This assumes your assembler supports local labels, as the GNU assembler and most Unix assemblers do.

Speaking of labels, jumps from one `asm` to another are not supported. The compiler's optimizers do not know about these jumps, and therefore they cannot take account of them when deciding how to optimize. See [Extended asm with goto], page 387.

Usually the most convenient way to use these `asm` instructions is to encapsulate them in macros that look like functions. For example,

```
#define sin(x)       \
({ double __value, __arg = (x);   \
   asm ("fsinx %1,%0": "=f" (__value): "f" (__arg));  \
   __value; })
```

Here the variable `__arg` is used to make sure that the instruction operates on a proper `double` value, and to accept only those arguments `x` which can convert automatically to a `double`.

Another way to make sure the instruction operates on the correct data type is to use a cast in the `asm`. This is different from using a variable `__arg` in that it converts more different types. For example, if the desired type were `int`, casting the argument to `int` would accept a pointer with no complaint, while assigning the argument to an `int` variable named `__arg` would warn about using a pointer unless the caller explicitly casts it.

If an `asm` has output operands, GCC assumes for optimization purposes the instruction has no side effects except to change the output operands. This does not mean instructions with a side effect cannot be used, but you must be careful, because the compiler may eliminate them if the output operands aren't used, or move them out of loops, or replace two with one if they constitute a common subexpression. Also, if your instruction does

have a side effect on a variable that otherwise appears not to change, the old value of the variable may be reused later if it happens to be found in a register.

You can prevent an `asm` instruction from being deleted by writing the keyword `volatile` after the `asm`. For example:

```
#define get_and_set_priority(new)              \
({ int __old;                                  \
   asm volatile ("get_and_set_priority %0, %1" \
                 : "=g" (__old) : "g" (new));  \
   __old; })
```

The `volatile` keyword indicates that the instruction has important side-effects. GCC will not delete a volatile `asm` if it is reachable. (The instruction can still be deleted if GCC can prove that control-flow will never reach the location of the instruction.) Note that even a volatile `asm` instruction can be moved relative to other code, including across jump instructions. For example, on many targets there is a system register which can be set to control the rounding mode of floating point operations. You might try setting it with a volatile `asm`, like this PowerPC example:

```
asm volatile("mtfsf 255,%0" : : "f" (fpenv));
sum = x + y;
```

This will not work reliably, as the compiler may move the addition back before the volatile `asm`. To make it work you need to add an artificial dependency to the `asm` referencing a variable in the code you don't want moved, for example:

```
asm volatile ("mtfsf 255,%1" : "=X"(sum): "f"(fpenv));
sum = x + y;
```

Similarly, you can't expect a sequence of volatile `asm` instructions to remain perfectly consecutive. If you want consecutive output, use a single `asm`. Also, GCC will perform some optimizations across a volatile `asm` instruction; GCC does not "forget everything" when it encounters a volatile `asm` instruction the way some other compilers do.

An `asm` instruction without any output operands will be treated identically to a volatile `asm` instruction.

It is a natural idea to look for a way to give access to the condition code left by the assembler instruction. However, when we attempted to implement this, we found no way to make it work reliably. The problem is that output operands might need reloading, which would result in additional following "store" instructions. On most machines, these instructions would alter the condition code before there was time to test it. This problem doesn't arise for ordinary "test" and "compare" instructions because they don't have any output operands.

For reasons similar to those described above, it is not possible to give an assembler instruction access to the condition code left by previous instructions.

As of GCC version 4.5, `asm goto` may be used to have the assembly jump to one or more C labels. In this form, a fifth section after the clobber list contains a list of all C labels to which the assembly may jump. Each label operand is implicitly self-named. The `asm` is also assumed to fall through to the next statement.

This form of `asm` is restricted to not have outputs. This is due to a internal restriction in the compiler that control transfer instructions cannot have outputs. This restriction on `asm goto` may be lifted in some future version of the compiler. In the mean time, `asm goto` may include a memory clobber, and so leave outputs in memory.

```
int frob(int x)
{
  int y;
  asm goto ("frob %%r5, %1; jc %l[error]; mov (%2), %%r5"
            : : "r"(x), "r"(&y) : "r5", "memory" : error);
  return y;
 error:
  return -1;
}
```

In this (inefficient) example, the `frob` instruction sets the carry bit to indicate an error. The `jc` instruction detects this and branches to the `error` label. Finally, the output of the `frob` instruction (`%r5`) is stored into the memory for variable `y`, which is later read by the `return` statement.

```
void doit(void)
{
  int i = 0;
  asm goto ("mfsr %%r1, 123; jmp %%r1;"
            ".pushsection doit_table;"
            ".long %l0, %l1, %l2, %l3;"
            ".popsection"
            : : : "r1" : label1, label2, label3, label4);
  __builtin_unreachable ();

 label1:
  f1();
  return;
 label2:
  f2();
  return;
 label3:
  i = 1;
 label4:
  f3(i);
}
```

In this (also inefficient) example, the `mfsr` instruction reads an address from some out-of-band machine register, and the following `jmp` instruction branches to that address. The address read by the `mfsr` instruction is assumed to have been previously set via some application-specific mechanism to be one of the four values stored in the `doit_table` section. Finally, the `asm` is followed by a call to `__builtin_unreachable` to indicate that the `asm` does not in fact fall through.

```
#define TRACE1(NUM)                         \
  do {                                      \
    asm goto ("0: nop;"                     \
              ".pushsection trace_table;"   \
              ".long 0b, %l0;"              \
              ".popsection"                 \
              : : : : trace#NUM);           \
    if (0) { trace#NUM: trace(); }          \
  } while (0)
#define TRACE  TRACE1(__COUNTER__)
```

In this example (which in fact inspired the `asm goto` feature) we want on rare occasions to call the `trace` function; on other occasions we'd like to keep the overhead to the absolute minimum. The normal code path consists of a single `nop` instruction. However, we record the address of this `nop` together with the address of a label that calls the `trace` function.

This allows the `nop` instruction to be patched at runtime to be an unconditional branch to the stored label. It is assumed that an optimizing compiler will move the labeled block out of line, to optimize the fall through path from the `asm`.

If you are writing a header file that should be includable in ISO C programs, write `__asm__` instead of `asm`. See Section 6.45 [Alternate Keywords], page 420.

6.41.1 Size of an `asm`

Some targets require that GCC track the size of each instruction used in order to generate correct code. Because the final length of an `asm` is only known by the assembler, GCC must make an estimate as to how big it will be. The estimate is formed by counting the number of statements in the pattern of the `asm` and multiplying that by the length of the longest instruction on that processor. Statements in the `asm` are identified by newline characters and whatever statement separator characters are supported by the assembler; on most processors this is the '`;`' character.

Normally, GCC's estimate is perfectly adequate to ensure that correct code is generated, but it is possible to confuse the compiler if you use pseudo instructions or assembler macros that expand into multiple real instructions or if you use assembler directives that expand to more space in the object file than would be needed for a single instruction. If this happens then the assembler will produce a diagnostic saying that a label is unreachable.

6.41.2 i386 floating point asm operands

There are several rules on the usage of stack-like regs in asm_operands insns. These rules apply only to the operands that are stack-like regs:

1. Given a set of input regs that die in an asm_operands, it is necessary to know which are implicitly popped by the asm, and which must be explicitly popped by gcc.

   An input reg that is implicitly popped by the asm must be explicitly clobbered, unless it is constrained to match an output operand.

2. For any input reg that is implicitly popped by an asm, it is necessary to know how to adjust the stack to compensate for the pop. If any non-popped input is closer to the top of the reg-stack than the implicitly popped reg, it would not be possible to know what the stack looked like—it's not clear how the rest of the stack "slides up".

   All implicitly popped input regs must be closer to the top of the reg-stack than any input that is not implicitly popped.

   It is possible that if an input dies in an insn, reload might use the input reg for an output reload. Consider this example:

   ```
   asm ("foo" : "=t" (a) : "f" (b));
   ```

   This asm says that input B is not popped by the asm, and that the asm pushes a result onto the reg-stack, i.e., the stack is one deeper after the asm than it was before. But, it is possible that reload will think that it can use the same reg for both the input and the output, if input B dies in this insn.

   If any input operand uses the `f` constraint, all output reg constraints must use the `&` earlyclobber.

   The asm above would be written as

   ```
   asm ("foo" : "=&t" (a) : "f" (b));
   ```

3. Some operands need to be in particular places on the stack. All output operands fall in this category—there is no other way to know which regs the outputs appear in unless the user indicates this in the constraints.

   Output operands must specifically indicate which reg an output appears in after an asm. `=f` is not allowed: the operand constraints must select a class with a single reg.

4. Output operands may not be "inserted" between existing stack regs. Since no 387 opcode uses a read/write operand, all output operands are dead before the asm_operands, and are pushed by the asm_operands. It makes no sense to push anywhere but the top of the reg-stack.

   Output operands must start at the top of the reg-stack: output operands may not "skip" a reg.

5. Some asm statements may need extra stack space for internal calculations. This can be guaranteed by clobbering stack registers unrelated to the inputs and outputs.

Here are a couple of reasonable asms to want to write. This asm takes one input, which is internally popped, and produces two outputs.

```
asm ("fsincos" : "=t" (cos), "=u" (sin) : "0" (inp));
```

This asm takes two inputs, which are popped by the `fyl2xp1` opcode, and replaces them with one output. The user must code the `st(1)` clobber for reg-stack.c to know that `fyl2xp1` pops both inputs.

```
asm ("fyl2xp1" : "=t" (result) : "0" (x), "u" (y) : "st(1)");
```

6.42 Constraints for `asm` Operands

Here are specific details on what constraint letters you can use with `asm` operands. Constraints can say whether an operand may be in a register, and which kinds of register; whether the operand can be a memory reference, and which kinds of address; whether the operand may be an immediate constant, and which possible values it may have. Constraints can also require two operands to match. Side-effects aren't allowed in operands of inline `asm`, unless '`<`' or '`>`' constraints are used, because there is no guarantee that the side-effects will happen exactly once in an instruction that can update the addressing register.

6.42.1 Simple Constraints

The simplest kind of constraint is a string full of letters, each of which describes one kind of operand that is permitted. Here are the letters that are allowed:

whitespace
: Whitespace characters are ignored and can be inserted at any position except the first. This enables each alternative for different operands to be visually aligned in the machine description even if they have different number of constraints and modifiers.

'`m`'
: A memory operand is allowed, with any kind of address that the machine supports in general. Note that the letter used for the general memory constraint can be re-defined by a back end using the `TARGET_MEM_CONSTRAINT` macro.

'`o`'
: A memory operand is allowed, but only if the address is *offsettable*. This means that adding a small integer (actually, the width in bytes of the operand,

as determined by its machine mode) may be added to the address and the result is also a valid memory address.

For example, an address which is constant is offsettable; so is an address that is the sum of a register and a constant (as long as a slightly larger constant is also within the range of address-offsets supported by the machine); but an autoincrement or autodecrement address is not offsettable. More complicated indirect/indexed addresses may or may not be offsettable depending on the other addressing modes that the machine supports.

Note that in an output operand which can be matched by another operand, the constraint letter 'o' is valid only when accompanied by both '<' (if the target machine has predecrement addressing) and '>' (if the target machine has preincrement addressing).

'V' A memory operand that is not offsettable. In other words, anything that would fit the 'm' constraint but not the 'o' constraint.

'<' A memory operand with autodecrement addressing (either predecrement or postdecrement) is allowed. In inline `asm` this constraint is only allowed if the operand is used exactly once in an instruction that can handle the side-effects. Not using an operand with '<' in constraint string in the inline `asm` pattern at all or using it in multiple instructions isn't valid, because the side-effects wouldn't be performed or would be performed more than once. Furthermore, on some targets the operand with '<' in constraint string must be accompanied by special instruction suffixes like `%U0` instruction suffix on PowerPC or `%P0` on IA-64.

'>' A memory operand with autoincrement addressing (either preincrement or postincrement) is allowed. In inline `asm` the same restrictions as for '<' apply.

'r' A register operand is allowed provided that it is in a general register.

'i' An immediate integer operand (one with constant value) is allowed. This includes symbolic constants whose values will be known only at assembly time or later.

'n' An immediate integer operand with a known numeric value is allowed. Many systems cannot support assembly-time constants for operands less than a word wide. Constraints for these operands should use 'n' rather than 'i'.

'I', 'J', 'K', ... 'P'
Other letters in the range 'I' through 'P' may be defined in a machine-dependent fashion to permit immediate integer operands with explicit integer values in specified ranges. For example, on the 68000, 'I' is defined to stand for the range of values 1 to 8. This is the range permitted as a shift count in the shift instructions.

'E' An immediate floating operand (expression code `const_double`) is allowed, but only if the target floating point format is the same as that of the host machine (on which the compiler is running).

'F' An immediate floating operand (expression code `const_double` or `const_vector`) is allowed.

'G', 'H' 'G' and 'H' may be defined in a machine-dependent fashion to permit immediate floating operands in particular ranges of values.

's' An immediate integer operand whose value is not an explicit integer is allowed.

This might appear strange; if an insn allows a constant operand with a value not known at compile time, it certainly must allow any known value. So why use 's' instead of 'i'? Sometimes it allows better code to be generated.

For example, on the 68000 in a fullword instruction it is possible to use an immediate operand; but if the immediate value is between −128 and 127, better code results from loading the value into a register and using the register. This is because the load into the register can be done with a 'moveq' instruction. We arrange for this to happen by defining the letter 'K' to mean "any integer outside the range −128 to 127", and then specifying 'Ks' in the operand constraints.

'g' Any register, memory or immediate integer operand is allowed, except for registers that are not general registers.

'X' Any operand whatsoever is allowed.

'0', '1', '2', ... '9'
An operand that matches the specified operand number is allowed. If a digit is used together with letters within the same alternative, the digit should come last.

This number is allowed to be more than a single digit. If multiple digits are encountered consecutively, they are interpreted as a single decimal integer. There is scant chance for ambiguity, since to-date it has never been desirable that '10' be interpreted as matching either operand 1 *or* operand 0. Should this be desired, one can use multiple alternatives instead.

This is called a *matching constraint* and what it really means is that the assembler has only a single operand that fills two roles which `asm` distinguishes. For example, an add instruction uses two input operands and an output operand, but on most CISC machines an add instruction really has only two operands, one of them an input-output operand:

```
addl #35,r12
```

Matching constraints are used in these circumstances. More precisely, the two operands that match must include one input-only operand and one output-only operand. Moreover, the digit must be a smaller number than the number of the operand that uses it in the constraint.

'p' An operand that is a valid memory address is allowed. This is for "load address" and "push address" instructions.

'p' in the constraint must be accompanied by `address_operand` as the predicate in the `match_operand`. This predicate interprets the mode specified in the `match_operand` as the mode of the memory reference for which the address would be valid.

other-letters

Other letters can be defined in machine-dependent fashion to stand for particular classes of registers or other arbitrary operand types. 'd', 'a' and 'f' are defined on the 68000/68020 to stand for data, address and floating point registers.

6.42.2 Multiple Alternative Constraints

Sometimes a single instruction has multiple alternative sets of possible operands. For example, on the 68000, a logical-or instruction can combine register or an immediate value into memory, or it can combine any kind of operand into a register; but it cannot combine one memory location into another.

These constraints are represented as multiple alternatives. An alternative can be described by a series of letters for each operand. The overall constraint for an operand is made from the letters for this operand from the first alternative, a comma, the letters for this operand from the second alternative, a comma, and so on until the last alternative.

If all the operands fit any one alternative, the instruction is valid. Otherwise, for each alternative, the compiler counts how many instructions must be added to copy the operands so that that alternative applies. The alternative requiring the least copying is chosen. If two alternatives need the same amount of copying, the one that comes first is chosen. These choices can be altered with the '?' and '!' characters:

'?' Disparage slightly the alternative that the '?' appears in, as a choice when no alternative applies exactly. The compiler regards this alternative as one unit more costly for each '?' that appears in it.

'!' Disparage severely the alternative that the '!' appears in. This alternative can still be used if it fits without reloading, but if reloading is needed, some other alternative will be used.

6.42.3 Constraint Modifier Characters

Here are constraint modifier characters.

'=' Means that this operand is write-only for this instruction: the previous value is discarded and replaced by output data.

'+' Means that this operand is both read and written by the instruction.

When the compiler fixes up the operands to satisfy the constraints, it needs to know which operands are inputs to the instruction and which are outputs from it. '=' identifies an output; '+' identifies an operand that is both input and output; all other operands are assumed to be input only.

If you specify '=' or '+' in a constraint, you put it in the first character of the constraint string.

'&' Means (in a particular alternative) that this operand is an *earlyclobber* operand, which is modified before the instruction is finished using the input operands. Therefore, this operand may not lie in a register that is used as an input operand or as part of any memory address.

'&' applies only to the alternative in which it is written. In constraints with multiple alternatives, sometimes one alternative requires '&' while others do not. See, for example, the 'movdf' insn of the 68000.

An input operand can be tied to an earlyclobber operand if its only use as an input occurs before the early result is written. Adding alternatives of this form often allows GCC to produce better code when only some of the inputs can be affected by the earlyclobber. See, for example, the 'mulsi3' insn of the ARM.

'&' does not obviate the need to write '='.

'%' Declares the instruction to be commutative for this operand and the following operand. This means that the compiler may interchange the two operands if that is the cheapest way to make all operands fit the constraints. GCC can only handle one commutative pair in an asm; if you use more, the compiler may fail. Note that you need not use the modifier if the two alternatives are strictly identical; this would only waste time in the reload pass. The modifier is not operational after register allocation, so the result of define_peephole2 and define_splits performed after reload cannot rely on '%' to make the intended insn match.

'#' Says that all following characters, up to the next comma, are to be ignored as a constraint. They are significant only for choosing register preferences.

'*' Says that the following character should be ignored when choosing register preferences. '*' has no effect on the meaning of the constraint as a constraint, and no effect on reloading.

6.42.4 Constraints for Particular Machines

Whenever possible, you should use the general-purpose constraint letters in asm arguments, since they will convey meaning more readily to people reading your code. Failing that, use the constraint letters that usually have very similar meanings across architectures. The most commonly used constraints are 'm' and 'r' (for memory and general-purpose registers respectively; see Section 6.42.1 [Simple Constraints], page 390), and 'I', usually the letter indicating the most common immediate-constant format.

Each architecture defines additional constraints. These constraints are used by the compiler itself for instruction generation, as well as for asm statements; therefore, some of the constraints are not particularly useful for asm. Here is a summary of some of the machine-dependent constraints available on some particular machines; it includes both constraints that are useful for asm and constraints that aren't. The compiler source file mentioned in the table heading for each architecture is the definitive reference for the meanings of that architecture's constraints.

ARM family—'config/arm/arm.h'

f	Floating-point register
w	VFP floating-point register
F	One of the floating-point constants 0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0 or 10.0

`G` Floating-point constant that would satisfy the constraint '`F`' if it were negated

`I` Integer that is valid as an immediate operand in a data processing instruction. That is, an integer in the range 0 to 255 rotated by a multiple of 2

`J` Integer in the range −4095 to 4095

`K` Integer that satisfies constraint '`I`' when inverted (ones complement)

`L` Integer that satisfies constraint '`I`' when negated (twos complement)

`M` Integer in the range 0 to 32

`Q` A memory reference where the exact address is in a single register ("`m`" is preferable for `asm` statements)

`R` An item in the constant pool

`S` A symbol in the text segment of the current file

`Uv` A memory reference suitable for VFP load/store insns (reg+constant offset)

`Uy` A memory reference suitable for iWMMXt load/store instructions.

`Uq` A memory reference suitable for the ARMv4 ldrsb instruction.

AVR family—'`config/avr/constraints.md`'

`l` Registers from r0 to r15

`a` Registers from r16 to r23

`d` Registers from r16 to r31

`w` Registers from r24 to r31. These registers can be used in '`adiw`' command

`e` Pointer register (r26–r31)

`b` Base pointer register (r28–r31)

`q` Stack pointer register (SPH:SPL)

`t` Temporary register r0

`x` Register pair X (r27:r26)

`y` Register pair Y (r29:r28)

`z` Register pair Z (r31:r30)

`I` Constant greater than −1, less than 64

`J` Constant greater than −64, less than 1

`K` Constant integer 2

`L` Constant integer 0

`M` Constant that fits in 8 bits

`N` Constant integer −1

`O` Constant integer 8, 16, or 24

`P` Constant integer 1

`G` A floating point constant 0.0

`Q` A memory address based on Y or Z pointer with displacement.

Epiphany—'`config/epiphany/constraints.md`'

`U16` An unsigned 16-bit constant.

`K` An unsigned 5-bit constant.

`L` A signed 11-bit constant.

`Cm1` A signed 11-bit constant added to −1. Can only match when the '`-m1reg-`*`reg`*' option is active.

`Cl1` Left-shift of −1, i.e., a bit mask with a block of leading ones, the rest being a block of trailing zeroes. Can only match when the '`-m1reg-`*`reg`*' option is active.

`Cr1` Right-shift of −1, i.e., a bit mask with a trailing block of ones, the rest being zeroes. Or to put it another way, one less than a power of two. Can only match when the '`-m1reg-`*`reg`*' option is active.

`Cal` Constant for arithmetic/logical operations. This is like `i`, except that for position independent code, no symbols / expressions needing relocations are allowed.

`Csy` Symbolic constant for call/jump instruction.

`Rcs` The register class usable in short insns. This is a register class constraint, and can thus drive register allocation. This constraint won't match unless '`-mprefer-short-insn-regs`' is in effect.

`Rsc` The the register class of registers that can be used to hold a sibcall call address. I.e., a caller-saved register.

`Rct` Core control register class.

`Rgs` The register group usable in short insns. This constraint does not use a register class, so that it only passively matches suitable registers, and doesn't drive register allocation.

`Rra` Matches the return address if it can be replaced with the link register.

`Rcc` Matches the integer condition code register.

`Sra` Matches the return address if it is in a stack slot.

`Cfm` Matches control register values to switch fp mode, which are encapsulated in `UNSPEC_FP_MODE`.

CR16 Architecture—‘`config/cr16/cr16.h`’

`b`	Registers from r0 to r14 (registers without stack pointer)
`t`	Register from r0 to r11 (all 16-bit registers)
`p`	Register from r12 to r15 (all 32-bit registers)
`I`	Signed constant that fits in 4 bits
`J`	Signed constant that fits in 5 bits
`K`	Signed constant that fits in 6 bits
`L`	Unsigned constant that fits in 4 bits
`M`	Signed constant that fits in 32 bits
`N`	Check for 64 bits wide constants for add/sub instructions
`G`	Floating point constant that is legal for store immediate

Hewlett-Packard PA-RISC—‘`config/pa/pa.h`’

`a`	General register 1
`f`	Floating point register
`q`	Shift amount register
`x`	Floating point register (deprecated)
`y`	Upper floating point register (32-bit), floating point register (64-bit)
`Z`	Any register
`I`	Signed 11-bit integer constant
`J`	Signed 14-bit integer constant
`K`	Integer constant that can be deposited with a `zdepi` instruction
`L`	Signed 5-bit integer constant
`M`	Integer constant 0
`N`	Integer constant that can be loaded with a `ldil` instruction
`O`	Integer constant whose value plus one is a power of 2
`P`	Integer constant that can be used for `and` operations in `depi` and `extru` instructions
`S`	Integer constant 31
`U`	Integer constant 63
`G`	Floating-point constant 0.0
`A`	A `lo_sum` data-linkage-table memory operand
`Q`	A memory operand that can be used as the destination operand of an integer store instruction

`R` A scaled or unscaled indexed memory operand

`T` A memory operand for floating-point loads and stores

`W` A register indirect memory operand

picoChip family—'`picochip.h`'

`k` Stack register.

`f` Pointer register. A register which can be used to access memory without supplying an offset. Any other register can be used to access memory, but will need a constant offset. In the case of the offset being zero, it is more efficient to use a pointer register, since this reduces code size.

`t` A twin register. A register which may be paired with an adjacent register to create a 32-bit register.

`a` Any absolute memory address (e.g., symbolic constant, symbolic constant + offset).

`I` 4-bit signed integer.

`J` 4-bit unsigned integer.

`K` 8-bit signed integer.

`M` Any constant whose absolute value is no greater than 4-bits.

`N` 10-bit signed integer

`O` 16-bit signed integer.

PowerPC and IBM RS6000—'`config/rs6000/rs6000.h`'

`b` Address base register

`d` Floating point register (containing 64-bit value)

`f` Floating point register (containing 32-bit value)

`v` Altivec vector register

`wd` VSX vector register to hold vector double data

`wf` VSX vector register to hold vector float data

`ws` VSX vector register to hold scalar float data

`wa` Any VSX register

`h` '`MQ`', '`CTR`', or '`LINK`' register

`q` '`MQ`' register

`c` '`CTR`' register

`l` '`LINK`' register

`x` '`CR`' register (condition register) number 0

`y` '`CR`' register (condition register)

`z` ‘`XER[CA]`’ carry bit (part of the XER register)

`I` Signed 16-bit constant

`J` Unsigned 16-bit constant shifted left 16 bits (use ‘`L`’ instead for `SImode` constants)

`K` Unsigned 16-bit constant

`L` Signed 16-bit constant shifted left 16 bits

`M` Constant larger than 31

`N` Exact power of 2

`O` Zero

`P` Constant whose negation is a signed 16-bit constant

`G` Floating point constant that can be loaded into a register with one instruction per word

`H` Integer/Floating point constant that can be loaded into a register using three instructions

`m` Memory operand. Normally, `m` does not allow addresses that update the base register. If ‘`<`’ or ‘`>`’ constraint is also used, they are allowed and therefore on PowerPC targets in that case it is only safe to use ‘`m<>`’ in an `asm` statement if that `asm` statement accesses the operand exactly once. The `asm` statement must also use ‘`%U<`*opno*`>`’ as a placeholder for the “update” flag in the corresponding load or store instruction. For example:

```
asm ("st%U0 %1,%0" : "=m<>" (mem) : "r" (val));
```

is correct but:

```
asm ("st %1,%0" : "=m<>" (mem) : "r" (val));
```

is not.

`es` A “stable” memory operand; that is, one which does not include any automodification of the base register. This used to be useful when ‘`m`’ allowed automodification of the base register, but as those are now only allowed when ‘`<`’ or ‘`>`’ is used, ‘`es`’ is basically the same as ‘`m`’ without ‘`<`’ and ‘`>`’.

`Q` Memory operand that is an offset from a register (it is usually better to use ‘`m`’ or ‘`es`’ in `asm` statements)

`Z` Memory operand that is an indexed or indirect from a register (it is usually better to use ‘`m`’ or ‘`es`’ in `asm` statements)

`R` AIX TOC entry

`a` Address operand that is an indexed or indirect from a register (‘`p`’ is preferable for `asm` statements)

`S` Constant suitable as a 64-bit mask operand

`T` Constant suitable as a 32-bit mask operand

`U` System V Release 4 small data area reference

`t` AND masks that can be performed by two rldic{l, r} instructions

`W` Vector constant that does not require memory

`j` Vector constant that is all zeros.

Intel 386—'`config/i386/constraints.md`'

`R` Legacy register—the eight integer registers available on all i386 processors (`a`, `b`, `c`, `d`, `si`, `di`, `bp`, `sp`).

`q` Any register accessible as `rl`. In 32-bit mode, `a`, `b`, `c`, and `d`; in 64-bit mode, any integer register.

`Q` Any register accessible as `rh`: `a`, `b`, `c`, and `d`.

`a` The `a` register.

`b` The `b` register.

`c` The `c` register.

`d` The `d` register.

`S` The `si` register.

`D` The `di` register.

`A` The `a` and `d` registers. This class is used for instructions that return double word results in the `ax:dx` register pair. Single word values will be allocated either in `ax` or `dx`. For example on i386 the following implements `rdtsc`:

```
unsigned long long rdtsc (void)
{
  unsigned long long tick;
  __asm__ __volatile__("rdtsc":"=A"(tick));
  return tick;
}
```

This is not correct on x86_64 as it would allocate tick in either `ax` or `dx`. You have to use the following variant instead:

```
unsigned long long rdtsc (void)
{
  unsigned int tickl, tickh;
  __asm__ __volatile__("rdtsc":"=a"(tickl),"=d"(tickh));
  return ((unsigned long long)tickh << 32)|tickl;
}
```

`f` Any 80387 floating-point (stack) register.

`t` Top of 80387 floating-point stack (`%st(0)`).

`u` Second from top of 80387 floating-point stack (`%st(1)`).

`y` Any MMX register.

`x` Any SSE register.

`Yz` First SSE register (`%xmm0`).

`I`	Integer constant in the range 0 . . . 31, for 32-bit shifts.
`J`	Integer constant in the range 0 . . . 63, for 64-bit shifts.
`K`	Signed 8-bit integer constant.
`L`	`0xFF` or `0xFFFF`, for andsi as a zero-extending move.
`M`	0, 1, 2, or 3 (shifts for the `lea` instruction).
`N`	Unsigned 8-bit integer constant (for `in` and `out` instructions).
`G`	Standard 80387 floating point constant.
`C`	Standard SSE floating point constant.
`e`	32-bit signed integer constant, or a symbolic reference known to fit that range (for immediate operands in sign-extending x86-64 instructions).
`Z`	32-bit unsigned integer constant, or a symbolic reference known to fit that range (for immediate operands in zero-extending x86-64 instructions).

Intel IA-64—'`config/ia64/ia64.h`'

`a`	General register `r0` to `r3` for `addl` instruction
`b`	Branch register
`c`	Predicate register ('`c`' as in "conditional")
`d`	Application register residing in M-unit
`e`	Application register residing in I-unit
`f`	Floating-point register
`m`	Memory operand. If used together with '`<`' or '`>`', the operand can have postincrement and postdecrement which require printing with '`%Pn`' on IA-64.
`G`	Floating-point constant 0.0 or 1.0
`I`	14-bit signed integer constant
`J`	22-bit signed integer constant
`K`	8-bit signed integer constant for logical instructions
`L`	8-bit adjusted signed integer constant for compare pseudo-ops
`M`	6-bit unsigned integer constant for shift counts
`N`	9-bit signed integer constant for load and store postincrements
`O`	The constant zero
`P`	0 or −1 for `dep` instruction
`Q`	Non-volatile memory for floating-point loads and stores
`R`	Integer constant in the range 1 to 4 for `shladd` instruction

`S` Memory operand except postincrement and postdecrement. This is now roughly the same as '`m`' when not used together with '`<`' or '`>`'.

FRV—'`config/frv/frv.h`'

`a` Register in the class `ACC_REGS` (`acc0` to `acc7`).

`b` Register in the class `EVEN_ACC_REGS` (`acc0` to `acc7`).

`c` Register in the class `CC_REGS` (`fcc0` to `fcc3` and `icc0` to `icc3`).

`d` Register in the class `GPR_REGS` (`gr0` to `gr63`).

`e` Register in the class `EVEN_REGS` (`gr0` to `gr63`). Odd registers are excluded not in the class but through the use of a machine mode larger than 4 bytes.

`f` Register in the class `FPR_REGS` (`fr0` to `fr63`).

`h` Register in the class `FEVEN_REGS` (`fr0` to `fr63`). Odd registers are excluded not in the class but through the use of a machine mode larger than 4 bytes.

`l` Register in the class `LR_REG` (the `lr` register).

`q` Register in the class `QUAD_REGS` (`gr2` to `gr63`). Register numbers not divisible by 4 are excluded not in the class but through the use of a machine mode larger than 8 bytes.

`t` Register in the class `ICC_REGS` (`icc0` to `icc3`).

`u` Register in the class `FCC_REGS` (`fcc0` to `fcc3`).

`v` Register in the class `ICR_REGS` (`cc4` to `cc7`).

`w` Register in the class `FCR_REGS` (`cc0` to `cc3`).

`x` Register in the class `QUAD_FPR_REGS` (`fr0` to `fr63`). Register numbers not divisible by 4 are excluded not in the class but through the use of a machine mode larger than 8 bytes.

`z` Register in the class `SPR_REGS` (`lcr` and `lr`).

`A` Register in the class `QUAD_ACC_REGS` (`acc0` to `acc7`).

`B` Register in the class `ACCG_REGS` (`accg0` to `accg7`).

`C` Register in the class `CR_REGS` (`cc0` to `cc7`).

`G` Floating point constant zero

`I` 6-bit signed integer constant

`J` 10-bit signed integer constant

`L` 16-bit signed integer constant

`M` 16-bit unsigned integer constant

`N` 12-bit signed integer constant that is negative—i.e. in the range of −2048 to −1

`O` Constant zero

`P` 12-bit signed integer constant that is greater than zero—i.e. in the range of 1 to 2047.

Blackfin family—'`config/bfin/constraints.md`'

`a` P register

`d` D register

`z` A call clobbered P register.

`q`*n* A single register. If *n* is in the range 0 to 7, the corresponding D register. If it is `A`, then the register P0.

`D` Even-numbered D register

`W` Odd-numbered D register

`e` Accumulator register.

`A` Even-numbered accumulator register.

`B` Odd-numbered accumulator register.

`b` I register

`v` B register

`f` M register

`c` Registers used for circular buffering, i.e. I, B, or L registers.

`C` The CC register.

`t` LT0 or LT1.

`k` LC0 or LC1.

`u` LB0 or LB1.

`x` Any D, P, B, M, I or L register.

`y` Additional registers typically used only in prologues and epilogues: RETS, RETN, RETI, RETX, RETE, ASTAT, SEQSTAT and USP.

`w` Any register except accumulators or CC.

`Ksh` Signed 16 bit integer (in the range −32768 to 32767)

`Kuh` Unsigned 16 bit integer (in the range 0 to 65535)

`Ks7` Signed 7 bit integer (in the range −64 to 63)

`Ku7` Unsigned 7 bit integer (in the range 0 to 127)

`Ku5` Unsigned 5 bit integer (in the range 0 to 31)

`Ks4` Signed 4 bit integer (in the range −8 to 7)

`Ks3` Signed 3 bit integer (in the range −3 to 4)

`Ku3` Unsigned 3 bit integer (in the range 0 to 7)

`Pn` Constant *n*, where *n* is a single-digit constant in the range 0 to 4.

`PA` An integer equal to one of the MACFLAG_XXX constants that is suitable for use with either accumulator.

`PB` An integer equal to one of the MACFLAG_XXX constants that is suitable for use only with accumulator A1.

`M1` Constant 255.

`M2` Constant 65535.

`J` An integer constant with exactly a single bit set.

`L` An integer constant with all bits set except exactly one.

`H`

`Q` Any SYMBOL_REF.

M32C—'`config/m32c/m32c.c`'

`Rsp`
`Rfb`
`Rsb` '`$sp`', '`$fb`', '`$sb`'.

`Rcr` Any control register, when they're 16 bits wide (nothing if control registers are 24 bits wide)

`Rcl` Any control register, when they're 24 bits wide.

`R0w`
`R1w`
`R2w`
`R3w` $r0, $r1, $r2, $r3.

`R02` $r0 or $r2, or $r2r0 for 32 bit values.

`R13` $r1 or $r3, or $r3r1 for 32 bit values.

`Rdi` A register that can hold a 64 bit value.

`Rhl` $r0 or $r1 (registers with addressable high/low bytes)

`R23` $r2 or $r3

`Raa` Address registers

`Raw` Address registers when they're 16 bits wide.

`Ral` Address registers when they're 24 bits wide.

`Rqi` Registers that can hold QI values.

`Rad` Registers that can be used with displacements ($a0, $a1, $sb).

`Rsi` Registers that can hold 32 bit values.

`Rhi` Registers that can hold 16 bit values.

`Rhc` Registers chat can hold 16 bit values, including all control registers.

`Rra` $r0 through R1, plus $a0 and $a1.

`Rfl` The flags register.

`Rmm` The memory-based pseudo-registers $mem0 through $mem15.

`Rpi` Registers that can hold pointers (16 bit registers for r8c, m16c; 24 bit registers for m32cm, m32c).

`Rpa` Matches multiple registers in a PARALLEL to form a larger register. Used to match function return values.

`Is3` −8 ... 7

`IS1` −128 ... 127

`IS2` −32768 ... 32767

`IU2` 0 ... 65535

`In4` −8 ... −1 or 1 ... 8

`In5` −16 ... −1 or 1 ... 16

`In6` −32 ... −1 or 1 ... 32

`IM2` −65536 ... −1

`Ilb` An 8 bit value with exactly one bit set.

`Ilw` A 16 bit value with exactly one bit set.

`Sd` The common src/dest memory addressing modes.

`Sa` Memory addressed using $a0 or $a1.

`Si` Memory addressed with immediate addresses.

`Ss` Memory addressed using the stack pointer ($sp).

`Sf` Memory addressed using the frame base register ($fb).

`Ss` Memory addressed using the small base register ($sb).

`S1` $r1h

MeP—'`config/mep/constraints.md`'

`a` The $sp register.

`b` The $tp register.

`c` Any control register.

`d` Either the $hi or the $lo register.

`em` Coprocessor registers that can be directly loaded ($c0-$c15).

`ex` Coprocessor registers that can be moved to each other.

`er` Coprocessor registers that can be moved to core registers.

`h` The $hi register.

`j` The $rpc register.

`l` The $lo register.

`t` Registers which can be used in $tp-relative addressing.

`v` The $gp register.

`x` The coprocessor registers.

`y` The coprocessor control registers.

`z` The $0 register.

`A` User-defined register set A.

`B` User-defined register set B.

`C` User-defined register set C.

`D` User-defined register set D.

`I` Offsets for $gp-rel addressing.

`J` Constants that can be used directly with boolean insns.

`K` Constants that can be moved directly to registers.

`L` Small constants that can be added to registers.

`M` Long shift counts.

`N` Small constants that can be compared to registers.

`O` Constants that can be loaded into the top half of registers.

`S` Signed 8-bit immediates.

`T` Symbols encoded for $tp-rel or $gp-rel addressing.

`U` Non-constant addresses for loading/saving coprocessor registers.

`W` The top half of a symbol's value.

`Y` A register indirect address without offset.

`Z` Symbolic references to the control bus.

MicroBlaze—'`config/microblaze/constraints.md`'

`d` A general register (`r0` to `r31`).

`z` A status register (`rmsr`, `$fcc1` to `$fcc7`).

MIPS—'`config/mips/constraints.md`'

`d` An address register. This is equivalent to `r` unless generating MIPS16 code.

`f` A floating-point register (if available).

`h` Formerly the `hi` register. This constraint is no longer supported.

`l` The `lo` register. Use this register to store values that are no bigger than a word.

`x` The concatenated `hi` and `lo` registers. Use this register to store doubleword values.

`c` A register suitable for use in an indirect jump. This will always be `$25` for '`-mabicalls`'.

`v` Register `$3`. Do not use this constraint in new code; it is retained only for compatibility with glibc.

`y` Equivalent to `r`; retained for backwards compatibility.

`z` A floating-point condition code register.

`I` A signed 16-bit constant (for arithmetic instructions).

`J` Integer zero.

`K` An unsigned 16-bit constant (for logic instructions).

`L` A signed 32-bit constant in which the lower 16 bits are zero. Such constants can be loaded using `lui`.

`M` A constant that cannot be loaded using `lui`, `addiu` or `ori`.

`N` A constant in the range −65535 to −1 (inclusive).

`O` A signed 15-bit constant.

`P` A constant in the range 1 to 65535 (inclusive).

`G` Floating-point zero.

`R` An address that can be used in a non-macro load or store.

Motorola 680x0—'`config/m68k/constraints.md`'

`a` Address register

`d` Data register

`f` 68881 floating-point register, if available

`I` Integer in the range 1 to 8

`J` 16-bit signed number

`K` Signed number whose magnitude is greater than 0x80

`L` Integer in the range −8 to −1

`M` Signed number whose magnitude is greater than 0x100

`N` Range 24 to 31, rotatert:SI 8 to 1 expressed as rotate

`O` 16 (for rotate using swap)

`P` Range 8 to 15, rotatert:HI 8 to 1 expressed as rotate

`R` Numbers that mov3q can handle

`G` Floating point constant that is not a 68881 constant

`S` Operands that satisfy 'm' when -mpcrel is in effect

`T`	Operands that satisfy 's' when -mpcrel is not in effect
`Q`	Address register indirect addressing mode
`U`	Register offset addressing
`W`	const_call_operand
`Cs`	symbol_ref or const
`Ci`	const_int
`C0`	const_int 0
`Cj`	Range of signed numbers that don't fit in 16 bits
`Cmvq`	Integers valid for mvq
`Capsw`	Integers valid for a moveq followed by a swap
`Cmvz`	Integers valid for mvz
`Cmvs`	Integers valid for mvs
`Ap`	push_operand
`Ac`	Non-register operands allowed in clr

Moxie—'`config/moxie/constraints.md`'

`A`	An absolute address
`B`	An offset address
`W`	A register indirect memory operand
`I`	A constant in the range of 0 to 255.
`N`	A constant in the range of 0 to −255.

PDP-11—'`config/pdp11/constraints.md`'

`a`	Floating point registers AC0 through AC3. These can be loaded from/to memory with a single instruction.
`d`	Odd numbered general registers (R1, R3, R5). These are used for 16-bit multiply operations.
`f`	Any of the floating point registers (AC0 through AC5).
`G`	Floating point constant 0.
`I`	An integer constant that fits in 16 bits.
`J`	An integer constant whose low order 16 bits are zero.
`K`	An integer constant that does not meet the constraints for codes '`I`' or '`J`'.
`L`	The integer constant 1.
`M`	The integer constant −1.
`N`	The integer constant 0.

`O` Integer constants −4 through −1 and 1 through 4; shifts by these amounts are handled as multiple single-bit shifts rather than a single variable-length shift.

`Q` A memory reference which requires an additional word (address or offset) after the opcode.

`R` A memory reference that is encoded within the opcode.

RL78—'`config/rl78/constraints.md`'

`Int3` An integer constant in the range 1 . . . 7.

`Int8` An integer constant in the range 0 . . . 255.

`J` An integer constant in the range −255 . . . 0

`K` The integer constant 1.

`L` The integer constant -1.

`M` The integer constant 0.

`N` The integer constant 2.

`O` The integer constant -2.

`P` An integer constant in the range 1 . . . 15.

`Qbi` The built-in compare types–eq, ne, gtu, ltu, geu, and leu.

`Qsc` The synthetic compare types–gt, lt, ge, and le.

`Wab` A memory reference with an absolute address.

`Wbc` A memory reference using `BC` as a base register, with an optional offset.

`Wca` A memory reference using `AX`, `BC`, `DE`, or `HL` for the address, for calls.

`Wcv` A memory reference using any 16-bit register pair for the address, for calls.

`Wd2` A memory reference using `DE` as a base register, with an optional offset.

`Wde` A memory reference using `DE` as a base register, without any offset.

`Wfr` Any memory reference to an address in the far address space.

`Wh1` A memory reference using `HL` as a base register, with an optional one-byte offset.

`Whb` A memory reference using `HL` as a base register, with `B` or `C` as the index register.

`Whl` A memory reference using `HL` as a base register, without any offset.

`Ws1` A memory reference using `SP` as a base register, with an optional one-byte offset.

`Y` Any memory reference to an address in the near address space.

`A` The `AX` register.

`B` The `BC` register.

`D` The `DE` register.

`R` `A` through `L` registers.

`S` The `SP` register.

`T` The `HL` register.

`Z08W` The 16-bit `R8` register.

`Z10W` The 16-bit `R10` register.

`Zint` The registers reserved for interrupts (`R24` to `R31`).

`a` The `A` register.

`b` The `B` register.

`c` The `C` register.

`d` The `D` register.

`e` The `E` register.

`h` The `H` register.

`l` The `L` register.

`v` The virtual registers.

`w` The `PSW` register.

`x` The `X` register.

RX—'`config/rx/constraints.md`'

`Q` An address which does not involve register indirect addressing or pre/post increment/decrement addressing.

`Symbol` A symbol reference.

`Int08` A constant in the range −256 to 255, inclusive.

`Sint08` A constant in the range −128 to 127, inclusive.

`Sint16` A constant in the range −32768 to 32767, inclusive.

`Sint24` A constant in the range −8388608 to 8388607, inclusive.

`Uint04` A constant in the range 0 to 15, inclusive.

SPARC—'`config/sparc/sparc.h`'

`f` Floating-point register on the SPARC-V8 architecture and lower floating-point register on the SPARC-V9 architecture.

`e` Floating-point register. It is equivalent to '`f`' on the SPARC-V8 architecture and contains both lower and upper floating-point registers on the SPARC-V9 architecture.

`c` Floating-point condition code register.

`d` Lower floating-point register. It is only valid on the SPARC-V9 architecture when the Visual Instruction Set is available.

`b` Floating-point register. It is only valid on the SPARC-V9 architecture when the Visual Instruction Set is available.

`h` 64-bit global or out register for the SPARC-V8+ architecture.

`D` A vector constant

`I` Signed 13-bit constant

`J` Zero

`K` 32-bit constant with the low 12 bits clear (a constant that can be loaded with the `sethi` instruction)

`L` A constant in the range supported by `movcc` instructions

`M` A constant in the range supported by `movrcc` instructions

`N` Same as '`K`', except that it verifies that bits that are not in the lower 32-bit range are all zero. Must be used instead of '`K`' for modes wider than `SImode`

`O` The constant 4096

`G` Floating-point zero

`H` Signed 13-bit constant, sign-extended to 32 or 64 bits

`Q` Floating-point constant whose integral representation can be moved into an integer register using a single sethi instruction

`R` Floating-point constant whose integral representation can be moved into an integer register using a single mov instruction

`S` Floating-point constant whose integral representation can be moved into an integer register using a high/lo_sum instruction sequence

`T` Memory address aligned to an 8-byte boundary

`U` Even register

`W` Memory address for '`e`' constraint registers

`Y` Vector zero

SPU—'`config/spu/spu.h`'

`a` An immediate which can be loaded with the il/ila/ilh/ilhu instructions. const_int is treated as a 64 bit value.

`c` An immediate for and/xor/or instructions. const_int is treated as a 64 bit value.

`d` An immediate for the `iohl` instruction. const_int is treated as a 64 bit value.

`f` An immediate which can be loaded with `fsmbi`.

`A` An immediate which can be loaded with the il/ila/ilh/ilhu instructions. const_int is treated as a 32 bit value.

`B` An immediate for most arithmetic instructions. const_int is treated as a 32 bit value.

`C` An immediate for and/xor/or instructions. const_int is treated as a 32 bit value.

`D` An immediate for the `iohl` instruction. const_int is treated as a 32 bit value.

`I` A constant in the range [−64, 63] for shift/rotate instructions.

`J` An unsigned 7-bit constant for conversion/nop/channel instructions.

`K` A signed 10-bit constant for most arithmetic instructions.

`M` A signed 16 bit immediate for `stop`.

`N` An unsigned 16-bit constant for `iohl` and `fsmbi`.

`O` An unsigned 7-bit constant whose 3 least significant bits are 0.

`P` An unsigned 3-bit constant for 16-byte rotates and shifts

`R` Call operand, reg, for indirect calls

`S` Call operand, symbol, for relative calls.

`T` Call operand, const_int, for absolute calls.

`U` An immediate which can be loaded with the il/ila/ilh/ilhu instructions. const_int is sign extended to 128 bit.

`W` An immediate for shift and rotate instructions. const_int is treated as a 32 bit value.

`Y` An immediate for and/xor/or instructions. const_int is sign extended as a 128 bit.

`Z` An immediate for the `iohl` instruction. const_int is sign extended to 128 bit.

S/390 and zSeries—'`config/s390/s390.h`'

`a` Address register (general purpose register except r0)

`c` Condition code register

`d` Data register (arbitrary general purpose register)

`f` Floating-point register

`I` Unsigned 8-bit constant (0–255)

`J` Unsigned 12-bit constant (0–4095)

`K` Signed 16-bit constant (−32768–32767)

`L` Value appropriate as displacement.

`(0..4095)`
: for short displacement

`(-524288..524287)`
: for long displacement

`M` Constant integer with a value of 0x7fffffff.

`N` Multiple letter constraint followed by 4 parameter letters.

`0..9:` number of the part counting from most to least significant

`H,Q:` mode of the part

`D,S,H:` mode of the containing operand

`0,F:` value of the other parts (F—all bits set)

The constraint matches if the specified part of a constant has a value different from its other parts.

`Q` Memory reference without index register and with short displacement.

`R` Memory reference with index register and short displacement.

`S` Memory reference without index register but with long displacement.

`T` Memory reference with index register and long displacement.

`U` Pointer with short displacement.

`W` Pointer with long displacement.

`Y` Shift count operand.

Score family—'`config/score/score.h`'

`d` Registers from r0 to r32.

`e` Registers from r0 to r16.

`t` r8—r11 or r22—r27 registers.

`h` hi register.

`l` lo register.

`x` hi + lo register.

`q` cnt register.

`y` lcb register.

`z` scb register.

`a` cnt + lcb + scb register.

`c` cr0—cr15 register.

`b` cp1 registers.

`f` cp2 registers.

`i` cp3 registers.

`j` cp1 + cp2 + cp3 registers.

`I` High 16-bit constant (32-bit constant with 16 LSBs zero).

`J` Unsigned 5 bit integer (in the range 0 to 31).

`K` Unsigned 16 bit integer (in the range 0 to 65535).

`L` Signed 16 bit integer (in the range −32768 to 32767).

`M` Unsigned 14 bit integer (in the range 0 to 16383).

`N` Signed 14 bit integer (in the range −8192 to 8191).

`Z` Any SYMBOL_REF.

Xstormy16—‘`config/stormy16/stormy16.h`’

`a` Register r0.

`b` Register r1.

`c` Register r2.

`d` Register r8.

`e` Registers r0 through r7.

`t` Registers r0 and r1.

`y` The carry register.

`z` Registers r8 and r9.

`I` A constant between 0 and 3 inclusive.

`J` A constant that has exactly one bit set.

`K` A constant that has exactly one bit clear.

`L` A constant between 0 and 255 inclusive.

`M` A constant between −255 and 0 inclusive.

`N` A constant between −3 and 0 inclusive.

`O` A constant between 1 and 4 inclusive.

`P` A constant between −4 and −1 inclusive.

`Q` A memory reference that is a stack push.

`R` A memory reference that is a stack pop.

`S` A memory reference that refers to a constant address of known value.

`T` The register indicated by Rx (not implemented yet).

`U` A constant that is not between 2 and 15 inclusive.

`Z` The constant 0.

TI C6X family—‘`config/c6x/constraints.md`’

`a` Register file A (A0–A31).

`b` Register file B (B0–B31).

`A` Predicate registers in register file A (A0–A2 on C64X and higher, A1 and A2 otherwise).

`B` Predicate registers in register file B (B0–B2).

`C` A call-used register in register file B (B0–B9, B16–B31).

`Da` Register file A, excluding predicate registers (A3–A31, plus A0 if not C64X or higher).

`Db` Register file B, excluding predicate registers (B3–B31).

`Iu4` Integer constant in the range 0 . . . 15.

`Iu5` Integer constant in the range 0 . . . 31.

`In5` Integer constant in the range −31 . . . 0.

`Is5` Integer constant in the range −16 . . . 15.

`I5x` Integer constant that can be the operand of an ADDA or a SUBA insn.

`IuB` Integer constant in the range 0 . . . 65535.

`IsB` Integer constant in the range −32768 . . . 32767.

`IsC` Integer constant in the range -2^{20} . . . $2^{20} - 1$.

`Jc` Integer constant that is a valid mask for the clr instruction.

`Js` Integer constant that is a valid mask for the set instruction.

`Q` Memory location with A base register.

`R` Memory location with B base register.

`Z` Register B14 (aka DP).

TILE-Gx—‘`config/tilegx/constraints.md`’

`R00`
`R01`
`R02`
`R03`
`R04`
`R05`
`R06`
`R07`
`R08`
`R09`
`R10` Each of these represents a register constraint for an individual register, from r0 to r10.

`I` Signed 8-bit integer constant.

`J` Signed 16-bit integer constant.

`K` Unsigned 16-bit integer constant.

`L` Integer constant that fits in one signed byte when incremented by one (−129 . . . 126).

`m` Memory operand. If used together with '`<`' or '`>`', the operand can have postincrement which requires printing with '`%In`' and '`%in`' on TILE-Gx. For example:

```
asm ("st_add %I0,%1,%i0" : "=m<>" (*mem) : "r" (val));
```

`M` A bit mask suitable for the BFINS instruction.

`N` Integer constant that is a byte tiled out eight times.

`O` The integer zero constant.

`P` Integer constant that is a sign-extended byte tiled out as four shorts.

`Q` Integer constant that fits in one signed byte when incremented (−129 . . . 126), but excluding -1.

`S` Integer constant that has all 1 bits consecutive and starting at bit 0.

`T` A 16-bit fragment of a got, tls, or pc-relative reference.

`U` Memory operand except postincrement. This is roughly the same as '`m`' when not used together with '`<`' or '`>`'.

`W` An 8-element vector constant with identical elements.

`Y` A 4-element vector constant with identical elements.

`Z0` The integer constant 0xffffffff.

`Z1` The integer constant 0xffffffff00000000.

TILEPro—'`config/tilepro/constraints.md`'

`R00`
`R01`
`R02`
`R03`
`R04`
`R05`
`R06`
`R07`
`R08`
`R09`
`R10` Each of these represents a register constraint for an individual register, from r0 to r10.

`I` Signed 8-bit integer constant.

`J` Signed 16-bit integer constant.

`K` Nonzero integer constant with low 16 bits zero.

`L` Integer constant that fits in one signed byte when incremented by one (−129 . . . 126).

`m` Memory operand. If used together with '`<`' or '`>`', the operand can have postincrement which requires printing with '`%In`' and '`%in`' on TILEPro. For example:

```
asm ("swadd %I0,%1,%i0" : "=m<>" (mem) : "r" (val));
```

`M` A bit mask suitable for the MM instruction.

`N` Integer constant that is a byte tiled out four times.

`O` The integer zero constant.

`P` Integer constant that is a sign-extended byte tiled out as two shorts.

`Q` Integer constant that fits in one signed byte when incremented (−129 . . . 126), but excluding -1.

`T` A symbolic operand, or a 16-bit fragment of a got, tls, or pc-relative reference.

`U` Memory operand except postincrement. This is roughly the same as '`m`' when not used together with '`<`' or '`>`'.

`W` A 4-element vector constant with identical elements.

`Y` A 2-element vector constant with identical elements.

Xtensa—'`config/xtensa/constraints.md`'

`a` General-purpose 32-bit register

`b` One-bit boolean register

`A` MAC16 40-bit accumulator register

`I` Signed 12-bit integer constant, for use in MOVI instructions

`J` Signed 8-bit integer constant, for use in ADDI instructions

`K` Integer constant valid for BccI instructions

`L` Unsigned constant valid for BccUI instructions

6.43 Controlling Names Used in Assembler Code

You can specify the name to be used in the assembler code for a C function or variable by writing the `asm` (or `__asm__`) keyword after the declarator as follows:

```
int foo asm ("myfoo") = 2;
```

This specifies that the name to be used for the variable `foo` in the assembler code should be '`myfoo`' rather than the usual '`_foo`'.

On systems where an underscore is normally prepended to the name of a C function or variable, this feature allows you to define names for the linker that do not start with an underscore.

It does not make sense to use this feature with a non-static local variable since such variables do not have assembler names. If you are trying to put the variable in a particular register, see Section 6.44 [Explicit Reg Vars], page 418. GCC presently accepts such code with a warning, but will probably be changed to issue an error, rather than a warning, in the future.

You cannot use `asm` in this way in a function *definition*; but you can get the same effect by writing a declaration for the function before its definition and putting `asm` there, like this:

```
extern func () asm ("FUNC");

func (x, y)
     int x, y;
/* ... */
```

It is up to you to make sure that the assembler names you choose do not conflict with any other assembler symbols. Also, you must not use a register name; that would produce completely invalid assembler code. GCC does not as yet have the ability to store static variables in registers. Perhaps that will be added.

6.44 Variables in Specified Registers

GNU C allows you to put a few global variables into specified hardware registers. You can also specify the register in which an ordinary register variable should be allocated.

- Global register variables reserve registers throughout the program. This may be useful in programs such as programming language interpreters which have a couple of global variables that are accessed very often.
- Local register variables in specific registers do not reserve the registers, except at the point where they are used as input or output operands in an `asm` statement and the `asm` statement itself is not deleted. The compiler's data flow analysis is capable of determining where the specified registers contain live values, and where they are available for other uses. Stores into local register variables may be deleted when they appear to be dead according to dataflow analysis. References to local register variables may be deleted or moved or simplified.

  These local variables are sometimes convenient for use with the extended `asm` feature (see Section 6.41 [Extended Asm], page 383), if you want to write one output of the assembler instruction directly into a particular register. (This will work provided the register you specify fits the constraints specified for that operand in the `asm`.)

6.44.1 Defining Global Register Variables

You can define a global register variable in GNU C like this:

```
register int *foo asm ("a5");
```

Here `a5` is the name of the register which should be used. Choose a register which is normally saved and restored by function calls on your machine, so that library routines will not clobber it.

Naturally the register name is cpu-dependent, so you would need to conditionalize your program according to cpu type. The register `a5` would be a good choice on a 68000 for a variable of pointer type. On machines with register windows, be sure to choose a "global" register that is not affected magically by the function call mechanism.

In addition, operating systems on one type of cpu may differ in how they name the registers; then you would need additional conditionals. For example, some 68000 operating systems call this register `%a5`.

Eventually there may be a way of asking the compiler to choose a register automatically, but first we need to figure out how it should choose and how to enable you to guide the choice. No solution is evident.

Defining a global register variable in a certain register reserves that register entirely for this use, at least within the current compilation. The register will not be allocated for any other purpose in the functions in the current compilation. The register will not be saved and restored by these functions. Stores into this register are never deleted even if they would appear to be dead, but references may be deleted or moved or simplified.

It is not safe to access the global register variables from signal handlers, or from more than one thread of control, because the system library routines may temporarily use the register for other things (unless you recompile them specially for the task at hand).

It is not safe for one function that uses a global register variable to call another such function `foo` by way of a third function `lose` that was compiled without knowledge of this variable (i.e. in a different source file in which the variable wasn't declared). This is because `lose` might save the register and put some other value there. For example, you can't expect a global register variable to be available in the comparison-function that you pass to `qsort`, since `qsort` might have put something else in that register. (If you are prepared to recompile `qsort` with the same global register variable, you can solve this problem.)

If you want to recompile `qsort` or other source files which do not actually use your global register variable, so that they will not use that register for any other purpose, then it suffices to specify the compiler option '`-ffixed-`*reg*'. You need not actually add a global register declaration to their source code.

A function which can alter the value of a global register variable cannot safely be called from a function compiled without this variable, because it could clobber the value the caller expects to find there on return. Therefore, the function which is the entry point into the part of the program that uses the global register variable must explicitly save and restore the value which belongs to its caller.

On most machines, `longjmp` will restore to each global register variable the value it had at the time of the `setjmp`. On some machines, however, `longjmp` will not change the value of global register variables. To be portable, the function that called `setjmp` should make other arrangements to save the values of the global register variables, and to restore them in a `longjmp`. This way, the same thing will happen regardless of what `longjmp` does.

All global register variable declarations must precede all function definitions. If such a declaration could appear after function definitions, the declaration would be too late to prevent the register from being used for other purposes in the preceding functions.

Global register variables may not have initial values, because an executable file has no means to supply initial contents for a register.

On the SPARC, there are reports that g3 ... g7 are suitable registers, but certain library functions, such as `getwd`, as well as the subroutines for division and remainder, modify g3 and g4. g1 and g2 are local temporaries.

On the 68000, a2 ... a5 should be suitable, as should d2 ... d7. Of course, it will not do to use more than a few of those.

6.44.2 Specifying Registers for Local Variables

You can define a local register variable with a specified register like this:

```
register int *foo asm ("a5");
```

Here `a5` is the name of the register which should be used. Note that this is the same syntax used for defining global register variables, but for a local variable it would appear within a function.

Naturally the register name is cpu-dependent, but this is not a problem, since specific registers are most often useful with explicit assembler instructions (see Section 6.41 [Extended Asm], page 383). Both of these things generally require that you conditionalize your program according to cpu type.

In addition, operating systems on one type of cpu may differ in how they name the registers; then you would need additional conditionals. For example, some 68000 operating systems call this register `%a5`.

Defining such a register variable does not reserve the register; it remains available for other uses in places where flow control determines the variable's value is not live.

This option does not guarantee that GCC will generate code that has this variable in the register you specify at all times. You may not code an explicit reference to this register in the *assembler instruction template* part of an `asm` statement and assume it will always refer to this variable. However, using the variable as an `asm` *operand* guarantees that the specified register is used for the operand.

Stores into local register variables may be deleted when they appear to be dead according to dataflow analysis. References to local register variables may be deleted or moved or simplified.

As for global register variables, it's recommended that you choose a register which is normally saved and restored by function calls on your machine, so that library routines will not clobber it. A common pitfall is to initialize multiple call-clobbered registers with arbitrary expressions, where a function call or library call for an arithmetic operator will overwrite a register value from a previous assignment, for example `r0` below:

```
register int *p1 asm ("r0") = ...;
register int *p2 asm ("r1") = ...;
```

In those cases, a solution is to use a temporary variable for each arbitrary expression. See [Example of asm with clobbered asm reg], page 384.

6.45 Alternate Keywords

'`-ansi`' and the various '`-std`' options disable certain keywords. This causes trouble when you want to use GNU C extensions, or a general-purpose header file that should be usable by all programs, including ISO C programs. The keywords `asm`, `typeof` and `inline` are not available in programs compiled with '`-ansi`' or '`-std`' (although `inline` can be used in a program compiled with '`-std=c99`' or '`-std=c11`'). The ISO C99 keyword `restrict` is only available when '`-std=gnu99`' (which will eventually be the default) or '`-std=c99`' (or the equivalent '`-std=iso9899:1999`'), or an option for a later standard version, is used.

The way to solve these problems is to put '`__`' at the beginning and end of each problematical keyword. For example, use `__asm__` instead of `asm`, and `__inline__` instead of `inline`.

Other C compilers won't accept these alternative keywords; if you want to compile with another compiler, you can define the alternate keywords as macros to replace them with the customary keywords. It looks like this:

```
#ifndef __GNUC__
#define __asm__ asm
#endif
```

'`-pedantic`' and other options cause warnings for many GNU C extensions. You can prevent such warnings within one expression by writing `__extension__` before the expression. `__extension__` has no effect aside from this.

6.46 Incomplete `enum` Types

You can define an `enum` tag without specifying its possible values. This results in an incomplete type, much like what you get if you write `struct foo` without describing the elements. A later declaration which does specify the possible values completes the type.

You can't allocate variables or storage using the type while it is incomplete. However, you can work with pointers to that type.

This extension may not be very useful, but it makes the handling of `enum` more consistent with the way `struct` and `union` are handled.

This extension is not supported by GNU C++.

6.47 Function Names as Strings

GCC provides three magic variables which hold the name of the current function, as a string. The first of these is `__func__`, which is part of the C99 standard:

The identifier `__func__` is implicitly declared by the translator as if, immediately following the opening brace of each function definition, the declaration

```
static const char __func__[] = "function-name";
```

appeared, where function-name is the name of the lexically-enclosing function. This name is the unadorned name of the function.

`__FUNCTION__` is another name for `__func__`. Older versions of GCC recognize only this name. However, it is not standardized. For maximum portability, we recommend you use `__func__`, but provide a fallback definition with the preprocessor:

```
#if __STDC_VERSION__ < 199901L
# if __GNUC__ >= 2
#  define __func__ __FUNCTION__
# else
#  define __func__ "<unknown>"
# endif
#endif
```

In C, `__PRETTY_FUNCTION__` is yet another name for `__func__`. However, in C++, `__PRETTY_FUNCTION__` contains the type signature of the function as well as its bare name. For example, this program:

```
extern "C" {
extern int printf (char *, ...);
}

class a {
```

```
 public:
  void sub (int i)
    {
      printf ("__FUNCTION__ = %s\n", __FUNCTION__);
      printf ("__PRETTY_FUNCTION__ = %s\n", __PRETTY_FUNCTION__);
    }
};

int
main (void)
{
  a ax;
  ax.sub (0);
  return 0;
}
```

gives this output:

```
__FUNCTION__ = sub
__PRETTY_FUNCTION__ = void a::sub(int)
```

These identifiers are not preprocessor macros. In GCC 3.3 and earlier, in C only, `__FUNCTION__` and `__PRETTY_FUNCTION__` were treated as string literals; they could be used to initialize `char` arrays, and they could be concatenated with other string literals. GCC 3.4 and later treat them as variables, like `__func__`. In C++, `__FUNCTION__` and `__PRETTY_FUNCTION__` have always been variables.

6.48 Getting the Return or Frame Address of a Function

These functions may be used to get information about the callers of a function.

`void * __builtin_return_address (`*unsigned int* `level)` [Built-in Function]

This function returns the return address of the current function, or of one of its callers. The *level* argument is number of frames to scan up the call stack. A value of `0` yields the return address of the current function, a value of `1` yields the return address of the caller of the current function, and so forth. When inlining the expected behavior is that the function will return the address of the function that will be returned to. To work around this behavior use the `noinline` function attribute.

The *level* argument must be a constant integer.

On some machines it may be impossible to determine the return address of any function other than the current one; in such cases, or when the top of the stack has been reached, this function will return `0` or a random value. In addition, `__builtin_frame_address` may be used to determine if the top of the stack has been reached.

Additional post-processing of the returned value may be needed, see `__builtin_extract_return_address`.

This function should only be used with a nonzero argument for debugging purposes.

`void * __builtin_extract_return_address (`*void *addr*`)` [Built-in Function]

The address as returned by `__builtin_return_address` may have to be fed through this function to get the actual encoded address. For example, on the 31-bit S/390 platform the highest bit has to be masked out, or on SPARC platforms an offset has to be added for the true next instruction to be executed.

If no fixup is needed, this function simply passes through *addr*.

`void * __builtin_frob_return_address (void *addr)` [Built-in Function]
This function does the reverse of `__builtin_extract_return_address`.

`void * __builtin_frame_address (unsigned int level)` [Built-in Function]
This function is similar to `__builtin_return_address`, but it returns the address of the function frame rather than the return address of the function. Calling `__builtin_frame_address` with a value of `0` yields the frame address of the current function, a value of `1` yields the frame address of the caller of the current function, and so forth.

The frame is the area on the stack which holds local variables and saved registers. The frame address is normally the address of the first word pushed on to the stack by the function. However, the exact definition depends upon the processor and the calling convention. If the processor has a dedicated frame pointer register, and the function has a frame, then `__builtin_frame_address` will return the value of the frame pointer register.

On some machines it may be impossible to determine the frame address of any function other than the current one; in such cases, or when the top of the stack has been reached, this function will return `0` if the first frame pointer is properly initialized by the startup code.

This function should only be used with a nonzero argument for debugging purposes.

6.49 Using vector instructions through built-in functions

On some targets, the instruction set contains SIMD vector instructions that operate on multiple values contained in one large register at the same time. For example, on the i386 the MMX, 3DNow! and SSE extensions can be used this way.

The first step in using these extensions is to provide the necessary data types. This should be done using an appropriate `typedef`:

```
typedef int v4si __attribute__ ((vector_size (16)));
```

The `int` type specifies the base type, while the attribute specifies the vector size for the variable, measured in bytes. For example, the declaration above causes the compiler to set the mode for the `v4si` type to be 16 bytes wide and divided into `int` sized units. For a 32-bit `int` this means a vector of 4 units of 4 bytes, and the corresponding mode of `foo` will be V4SI.

The `vector_size` attribute is only applicable to integral and float scalars, although arrays, pointers, and function return values are allowed in conjunction with this construct.

All the basic integer types can be used as base types, both as signed and as unsigned: `char`, `short`, `int`, `long`, `long long`. In addition, `float` and `double` can be used to build floating-point vector types.

Specifying a combination that is not valid for the current architecture will cause GCC to synthesize the instructions using a narrower mode. For example, if you specify a variable of type `V4SI` and your architecture does not allow for this specific SIMD type, GCC will produce code that uses 4 `SIs`.

The types defined in this manner can be used with a subset of normal C operations. Currently, GCC will allow using the following operators on these types: `+, -, *, /, unary minus, ^, |, &, ~, %`.

The operations behave like C++ `valarrays`. Addition is defined as the addition of the corresponding elements of the operands. For example, in the code below, each of the 4 elements in *a* will be added to the corresponding 4 elements in *b* and the resulting vector will be stored in *c*.

```
typedef int v4si __attribute__ ((vector_size (16)));

v4si a, b, c;

c = a + b;
```

Subtraction, multiplication, division, and the logical operations operate in a similar manner. Likewise, the result of using the unary minus or complement operators on a vector type is a vector whose elements are the negative or complemented values of the corresponding elements in the operand.

In C it is possible to use shifting operators `<<`, `>>` on integer-type vectors. The operation is defined as following: `{a0, a1, ..., an} >> {b0, b1, ..., bn} == {a0 >> b0, a1 >> b1, ..., an >> bn}`. Vector operands must have the same number of elements.

For the convenience in C it is allowed to use a binary vector operation where one operand is a scalar. In that case the compiler will transform the scalar operand into a vector where each element is the scalar from the operation. The transformation will happen only if the scalar could be safely converted to the vector-element type. Consider the following code.

```
typedef int v4si __attribute__ ((vector_size (16)));

v4si a, b, c;
long l;

a = b + 1;    /* a = b + {1,1,1,1}; */
a = 2 * b;    /* a = {2,2,2,2} * b; */

a = l + a;    /* Error, cannot convert long to int. */
```

In C vectors can be subscripted as if the vector were an array with the same number of elements and base type. Out of bound accesses invoke undefined behavior at runtime. Warnings for out of bound accesses for vector subscription can be enabled with '`-Warray-bounds`'.

In GNU C vector comparison is supported within standard comparison operators: `==`, `!=`, `<`, `<=`, `>`, `>=`. Comparison operands can be vector expressions of integer-type or real-type. Comparison between integer-type vectors and real-type vectors are not supported. The result of the comparison is a vector of the same width and number of elements as the comparison operands with a signed integral element type.

Vectors are compared element-wise producing 0 when comparison is false and -1 (constant of the appropriate type where all bits are set) otherwise. Consider the following example.

```
typedef int v4si __attribute__ ((vector_size (16)));

v4si a = {1,2,3,4};
v4si b = {3,2,1,4};
v4si c;

c = a >  b;     /* The result would be {0, 0,-1, 0}  */
c = a == b;     /* The result would be {0,-1, 0,-1}  */
```

Vector shuffling is available using functions `__builtin_shuffle (vec, mask)` and `__builtin_shuffle (vec0, vec1, mask)`. Both functions construct a permutation of ele-

ments from one or two vectors and return a vector of the same type as the input vector(s). The *mask* is an integral vector with the same width (W) and element count (N) as the output vector.

The elements of the input vectors are numbered in memory ordering of *vec0* beginning at 0 and *vec1* beginning at N. The elements of *mask* are considered modulo N in the single-operand case and modulo $2 * N$ in the two-operand case.

Consider the following example,

```
typedef int v4si __attribute__ ((vector_size (16)));

v4si a = {1,2,3,4};
v4si b = {5,6,7,8};
v4si mask1 = {0,1,1,3};
v4si mask2 = {0,4,2,5};
v4si res;

res = __builtin_shuffle (a, mask1);       /* res is {1,2,2,4}  */
res = __builtin_shuffle (a, b, mask2);    /* res is {1,5,3,6}  */
```

Note that `__builtin_shuffle` is intentionally semantically compatible with the OpenCL `shuffle` and `shuffle2` functions.

You can declare variables and use them in function calls and returns, as well as in assignments and some casts. You can specify a vector type as a return type for a function. Vector types can also be used as function arguments. It is possible to cast from one vector type to another, provided they are of the same size (in fact, you can also cast vectors to and from other datatypes of the same size).

You cannot operate between vectors of different lengths or different signedness without a cast.

6.50 Offsetof

GCC implements for both C and C++ a syntactic extension to implement the `offsetof` macro.

```
primary:
        "__builtin_offsetof" "(" typename "," offsetof_member_designator ")"

offsetof_member_designator:
          identifier
        | offsetof_member_designator "." identifier
        | offsetof_member_designator "[" expr "]"
```

This extension is sufficient such that

```
#define offsetof(type, member)  __builtin_offsetof (type, member)
```

is a suitable definition of the `offsetof` macro. In C++, *type* may be dependent. In either case, *member* may consist of a single identifier, or a sequence of member accesses and array references.

6.51 Legacy __sync built-in functions for atomic memory access

The following builtins are intended to be compatible with those described in the *Intel Itanium Processor-specific Application Binary Interface*, section 7.4. As such, they depart

from the normal GCC practice of using the "__builtin_" prefix, and further that they are overloaded such that they work on multiple types.

The definition given in the Intel documentation allows only for the use of the types `int`, `long`, `long long` as well as their unsigned counterparts. GCC will allow any integral scalar or pointer type that is 1, 2, 4 or 8 bytes in length.

Not all operations are supported by all target processors. If a particular operation cannot be implemented on the target processor, a warning will be generated and a call an external function will be generated. The external function will carry the same name as the builtin, with an additional suffix '*_n*' where *n* is the size of the data type.

In most cases, these builtins are considered a *full barrier*. That is, no memory operand will be moved across the operation, either forward or backward. Further, instructions will be issued as necessary to prevent the processor from speculating loads across the operation and from queuing stores after the operation.

All of the routines are described in the Intel documentation to take "an optional list of variables protected by the memory barrier". It's not clear what is meant by that; it could mean that *only* the following variables are protected, or it could mean that these variables should in addition be protected. At present GCC ignores this list and protects all variables which are globally accessible. If in the future we make some use of this list, an empty list will continue to mean all globally accessible variables.

type __sync_fetch_and_add (*type* *ptr, *type* value, ...)
type __sync_fetch_and_sub (*type* *ptr, *type* value, ...)
type __sync_fetch_and_or (*type* *ptr, *type* value, ...)
type __sync_fetch_and_and (*type* *ptr, *type* value, ...)
type __sync_fetch_and_xor (*type* *ptr, *type* value, ...)
type __sync_fetch_and_nand (*type* *ptr, *type* value, ...)

These builtins perform the operation suggested by the name, and returns the value that had previously been in memory. That is,

```
{ tmp = *ptr; *ptr op= value; return tmp; }
{ tmp = *ptr; *ptr = ~(tmp & value); return tmp; }   // nand
```

Note: GCC 4.4 and later implement `__sync_fetch_and_nand` builtin as `*ptr = ~(tmp & value)` instead of `*ptr = ~tmp & value`.

type __sync_add_and_fetch (*type* *ptr, *type* value, ...)
type __sync_sub_and_fetch (*type* *ptr, *type* value, ...)
type __sync_or_and_fetch (*type* *ptr, *type* value, ...)
type __sync_and_and_fetch (*type* *ptr, *type* value, ...)
type __sync_xor_and_fetch (*type* *ptr, *type* value, ...)
type __sync_nand_and_fetch (*type* *ptr, *type* value, ...)

These builtins perform the operation suggested by the name, and return the new value. That is,

```
{ *ptr op= value; return *ptr; }
{ *ptr = ~(*ptr & value); return *ptr; }   // nand
```

Note: GCC 4.4 and later implement `__sync_nand_and_fetch` builtin as `*ptr = ~(*ptr & value)` instead of `*ptr = ~*ptr & value`.

`bool __sync_bool_compare_and_swap (type *ptr, type oldval, type newval, ...)`
`type __sync_val_compare_and_swap (type *ptr, type oldval, type newval, ...)`

These builtins perform an atomic compare and swap. That is, if the current value of `*ptr` is *oldval*, then write *newval* into `*ptr`.

The "bool" version returns true if the comparison is successful and *newval* was written. The "val" version returns the contents of `*ptr` before the operation.

`__sync_synchronize (...)`

This builtin issues a full memory barrier.

`type __sync_lock_test_and_set (type *ptr, type value, ...)`

This builtin, as described by Intel, is not a traditional test-and-set operation, but rather an atomic exchange operation. It writes *value* into `*ptr`, and returns the previous contents of `*ptr`.

Many targets have only minimal support for such locks, and do not support a full exchange operation. In this case, a target may support reduced functionality here by which the *only* valid value to store is the immediate constant 1. The exact value actually stored in `*ptr` is implementation defined.

This builtin is not a full barrier, but rather an *acquire barrier*. This means that references after the builtin cannot move to (or be speculated to) before the builtin, but previous memory stores may not be globally visible yet, and previous memory loads may not yet be satisfied.

`void __sync_lock_release (type *ptr, ...)`

This builtin releases the lock acquired by `__sync_lock_test_and_set`. Normally this means writing the constant 0 to `*ptr`.

This builtin is not a full barrier, but rather a *release barrier*. This means that all previous memory stores are globally visible, and all previous memory loads have been satisfied, but following memory reads are not prevented from being speculated to before the barrier.

6.52 Built-in functions for memory model aware atomic operations

The following built-in functions approximately match the requirements for C++11 memory model. Many are similar to the '`__sync`' prefixed built-in functions, but all also have a memory model parameter. These are all identified by being prefixed with '`__atomic`', and most are overloaded such that they work with multiple types.

GCC will allow any integral scalar or pointer type that is 1, 2, 4, or 8 bytes in length. 16-byte integral types are also allowed if '`__int128`' (see Section 6.8 [__int128], page 318) is supported by the architecture.

Target architectures are encouraged to provide their own patterns for each of these built-in functions. If no target is provided, the original non-memory model set of '`__sync`' atomic built-in functions will be utilized, along with any required synchronization fences surrounding it in order to achieve the proper behaviour. Execution in this case is subject to the same restrictions as those built-in functions.

If there is no pattern or mechanism to provide a lock free instruction sequence, a call is made to an external routine with the same parameters to be resolved at runtime.

The four non-arithmetic functions (load, store, exchange, and compare_exchange) all have a generic version as well. This generic version will work on any data type. If the data type size maps to one of the integral sizes which may have lock free support, the generic version will utilize the lock free built-in function. Otherwise an external call is left to be resolved at runtime. This external call will be the same format with the addition of a '`size_t`' parameter inserted as the first parameter indicating the size of the object being pointed to. All objects must be the same size.

There are 6 different memory models which can be specified. These map to the same names in the C++11 standard. Refer there or to the GCC wiki on atomic synchronization (`http://gcc.gnu.org/wiki/Atomic/GCCMM/AtomicSync`) for more detailed definitions. These memory models integrate both barriers to code motion as well as synchronization requirements with other threads. These are listed in approximately ascending order of strength.

`__ATOMIC_RELAXED`
: No barriers or synchronization.

`__ATOMIC_CONSUME`
: Data dependency only for both barrier and synchronization with another thread.

`__ATOMIC_ACQUIRE`
: Barrier to hoisting of code and synchronizes with release (or stronger) semantic stores from another thread.

`__ATOMIC_RELEASE`
: Barrier to sinking of code and synchronizes with acquire (or stronger) semantic loads from another thread.

`__ATOMIC_ACQ_REL`
: Full barrier in both directions and synchronizes with acquire loads and release stores in another thread.

`__ATOMIC_SEQ_CST`
: Full barrier in both directions and synchronizes with acquire loads and release stores in all threads.

When implementing patterns for these built-in functions , the memory model parameter can be ignored as long as the pattern implements the most restrictive `__ATOMIC_SEQ_CST` model. Any of the other memory models will execute correctly with this memory model but they may not execute as efficiently as they could with a more appropriate implemention of the relaxed requirements.

Note that the C++11 standard allows for the memory model parameter to be determined at runtime rather than at compile time. These built-in functions will map any runtime value to `__ATOMIC_SEQ_CST` rather than invoke a runtime library call or inline a switch statement. This is standard compliant, safe, and the simplest approach for now.

The memory model parameter is a signed int, but only the lower 8 bits are reserved for the memory model. The remainder of the signed int is reserved for future use and should be 0. Use of the predefined atomic values will ensure proper usage.

type __atomic_load_n (*type* *ptr, int memmodel) [Built-in Function]
This built-in function implements an atomic load operation. It returns the contents of **ptr*.

The valid memory model variants are __ATOMIC_RELAXED, __ATOMIC_SEQ_CST, __ATOMIC_ACQUIRE, and __ATOMIC_CONSUME.

void __atomic_load (*type* *ptr, *type* *ret, int memmodel) [Built-in Function]
This is the generic version of an atomic load. It will return the contents of **ptr* in **ret*.

void __atomic_store_n (*type* *ptr, *type* val, int memmodel) [Built-in Function]
This built-in function implements an atomic store operation. It writes *val* into **ptr*.

The valid memory model variants are __ATOMIC_RELAXED, __ATOMIC_SEQ_CST, and __ATOMIC_RELEASE.

void __atomic_store (*type* *ptr, *type* *val, int memmodel) [Built-in Function]
This is the generic version of an atomic store. It will store the value of **val* into **ptr*.

type __atomic_exchange_n (*type* *ptr, *type* val, int memmodel) [Built-in Function]
This built-in function implements an atomic exchange operation. It writes *val* into **ptr*, and returns the previous contents of **ptr*.

The valid memory model variants are __ATOMIC_RELAXED, __ATOMIC_SEQ_CST, __ATOMIC_ACQUIRE, __ATOMIC_RELEASE, and __ATOMIC_ACQ_REL.

void __atomic_exchange (*type* *ptr, *type* *val, *type* *ret, int memmodel) [Built-in Function]
This is the generic version of an atomic exchange. It will store the contents of **val* into **ptr*. The original value of **ptr* will be copied into **ret*.

bool __atomic_compare_exchange_n (*type* *ptr, *type* *expected, *type* desired, bool weak, int success_memmodel, int failure_memmodel) [Built-in Function]
This built-in function implements an atomic compare and exchange operation. This compares the contents of **ptr* with the contents of **expected* and if equal, writes *desired* into **ptr*. If they are not equal, the current contents of **ptr* is written into **expected*. *weak* is true for weak compare_exchange, and false for the strong variation. Many targets only offer the strong variation and ignore the parameter. When in doubt, use the strong variation.

True is returned if *desired* is written into **ptr* and the execution is considered to conform to the memory model specified by *success_memmodel*. There are no restrictions on what memory model can be used here.

False is returned otherwise, and the execution is considered to conform to *failure_memmodel*. This memory model cannot be __ATOMIC_RELEASE nor __ATOMIC_ACQ_REL. It also cannot be a stronger model than that specified by *success_memmodel*.

`bool __atomic_compare_exchange (type *ptr, type *expected, type *desired, bool weak, int success_memmodel, int failure_memmodel)` [Built-in Function]

This built-in function implements the generic version of `__atomic_compare_exchange`. The function is virtually identical to `__atomic_compare_exchange_n`, except the desired value is also a pointer.

`type __atomic_add_fetch (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_sub_fetch (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_and_fetch (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_xor_fetch (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_or_fetch (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_nand_fetch (type *ptr, type val, int memmodel)` [Built-in Function]

These built-in functions perform the operation suggested by the name, and return the result of the operation. That is,

```
{ *ptr op= val; return *ptr; }
```

All memory models are valid.

`type __atomic_fetch_add (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_fetch_sub (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_fetch_and (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_fetch_xor (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_fetch_or (type *ptr, type val, int memmodel)` [Built-in Function]
`type __atomic_fetch_nand (type *ptr, type val, int memmodel)` [Built-in Function]

These built-in functions perform the operation suggested by the name, and return the value that had previously been in `*ptr`. That is,

```
{ tmp = *ptr; *ptr op= val; return tmp; }
```

All memory models are valid.

`bool __atomic_test_and_set (void *ptr, int memmodel)` [Built-in Function]

This built-in function performs an atomic test-and-set operation on the byte at `*ptr`. The byte is set to some implementation defined non-zero "set" value and the return value is `true` if and only if the previous contents were "set".

All memory models are valid.

`void __atomic_clear` (*bool *ptr, int memmodel*) [Built-in Function]
This built-in function performs an atomic clear operation on `*ptr`. After the operation, `*ptr` will contain 0.

The valid memory model variants are `__ATOMIC_RELAXED`, `__ATOMIC_SEQ_CST`, and `__ATOMIC_RELEASE`.

`void __atomic_thread_fence` (*int memmodel*) [Built-in Function]
This built-in function acts as a synchronization fence between threads based on the specified memory model.

All memory orders are valid.

`void __atomic_signal_fence` (*int memmodel*) [Built-in Function]
This built-in function acts as a synchronization fence between a thread and signal handlers based in the same thread.

All memory orders are valid.

`bool __atomic_always_lock_free` (*size_t size, void *ptr*) [Built-in Function]
This built-in function returns true if objects of *size* bytes will always generate lock free atomic instructions for the target architecture. *size* must resolve to a compile time constant and the result also resolves to compile time constant.

ptr is an optional pointer to the object which may be used to determine alignment. A value of 0 indicates typical alignment should be used. The compiler may also ignore this parameter.

```
if (_atomic_always_lock_free (sizeof (long long), 0))
```

`bool __atomic_is_lock_free` (*size_t size, void *ptr*) [Built-in Function]
This built-in function returns true if objects of *size* bytes will always generate lock free atomic instructions for the target architecture. If it is not known to be lock free a call is made to a runtime routine named `__atomic_is_lock_free`.

ptr is an optional pointer to the object which may be used to determine alignment. A value of 0 indicates typical alignment should be used. The compiler may also ignore this parameter.

6.53 Object Size Checking Builtins

GCC implements a limited buffer overflow protection mechanism that can prevent some buffer overflow attacks.

`size_t __builtin_object_size` (*void * ptr, int type*) [Built-in Function]
is a built-in construct that returns a constant number of bytes from *ptr* to the end of the object *ptr* pointer points to (if known at compile time). `__builtin_object_size` never evaluates its arguments for side-effects. If there are any side-effects in them, it returns `(size_t) -1` for *type* 0 or 1 and `(size_t) 0` for *type* 2 or 3. If there are multiple objects *ptr* can point to and all of them are known at compile time, the returned number is the maximum of remaining byte counts in those objects if *type* & 2 is 0 and minimum if nonzero. If it is not possible to determine which objects *ptr* points to at compile time, `__builtin_object_size` should return `(size_t) -1` for *type* 0 or 1 and `(size_t) 0` for *type* 2 or 3.

type is an integer constant from 0 to 3. If the least significant bit is clear, objects are whole variables, if it is set, a closest surrounding subobject is considered the object a pointer points to. The second bit determines if maximum or minimum of remaining bytes is computed.

```
struct V { char buf1[10]; int b; char buf2[10]; } var;
char *p = &var.buf1[1], *q = &var.b;

/* Here the object p points to is var.  */
assert (__builtin_object_size (p, 0) == sizeof (var) - 1);
/* The subobject p points to is var.buf1.  */
assert (__builtin_object_size (p, 1) == sizeof (var.buf1) - 1);
/* The object q points to is var.  */
assert (__builtin_object_size (q, 0)
        == (char *) (&var + 1) - (char *) &var.b);
/* The subobject q points to is var.b.  */
assert (__builtin_object_size (q, 1) == sizeof (var.b));
```

There are built-in functions added for many common string operation functions, e.g., for `memcpy` `__builtin___memcpy_chk` built-in is provided. This built-in has an additional last argument, which is the number of bytes remaining in object the *dest* argument points to or `(size_t) -1` if the size is not known.

The built-in functions are optimized into the normal string functions like `memcpy` if the last argument is `(size_t) -1` or if it is known at compile time that the destination object will not be overflown. If the compiler can determine at compile time the object will be always overflown, it issues a warning.

The intended use can be e.g.

```
#undef memcpy
#define bos0(dest) __builtin_object_size (dest, 0)
#define memcpy(dest, src, n) \
  __builtin___memcpy_chk (dest, src, n, bos0 (dest))

char *volatile p;
char buf[10];
/* It is unknown what object p points to, so this is optimized
   into plain memcpy - no checking is possible.  */
memcpy (p, "abcde", n);
/* Destination is known and length too.  It is known at compile
   time there will be no overflow.  */
memcpy (&buf[5], "abcde", 5);
/* Destination is known, but the length is not known at compile time.
   This will result in __memcpy_chk call that can check for overflow
   at runtime.  */
memcpy (&buf[5], "abcde", n);
/* Destination is known and it is known at compile time there will
   be overflow.  There will be a warning and __memcpy_chk call that
   will abort the program at runtime.  */
memcpy (&buf[6], "abcde", 5);
```

Such built-in functions are provided for `memcpy`, `mempcpy`, `memmove`, `memset`, `strcpy`, `stpcpy`, `strncpy`, `strcat` and `strncat`.

There are also checking built-in functions for formatted output functions.

```
int __builtin___sprintf_chk (char *s, int flag, size_t os, const char *fmt, ...);
int __builtin___snprintf_chk (char *s, size_t maxlen, int flag, size_t os,
                              const char *fmt, ...);
```

```
int __builtin___vsprintf_chk (char *s, int flag, size_t os, const char *fmt,
                              va_list ap);
int __builtin___vsnprintf_chk (char *s, size_t maxlen, int flag, size_t os,
                               const char *fmt, va_list ap);
```

The added *flag* argument is passed unchanged to `__sprintf_chk` etc. functions and can contain implementation specific flags on what additional security measures the checking function might take, such as handling `%n` differently.

The *os* argument is the object size *s* points to, like in the other built-in functions. There is a small difference in the behavior though, if *os* is `(size_t) -1`, the built-in functions are optimized into the non-checking functions only if *flag* is 0, otherwise the checking function is called with *os* argument set to `(size_t) -1`.

In addition to this, there are checking built-in functions `__builtin___printf_chk`, `__builtin___vprintf_chk`, `__builtin___fprintf_chk` and `__builtin___vfprintf_chk`. These have just one additional argument, *flag*, right before format string *fmt*. If the compiler is able to optimize them to `fputc` etc. functions, it will, otherwise the checking function should be called and the *flag* argument passed to it.

6.54 Other built-in functions provided by GCC

GCC provides a large number of built-in functions other than the ones mentioned above. Some of these are for internal use in the processing of exceptions or variable-length argument lists and will not be documented here because they may change from time to time; we do not recommend general use of these functions.

The remaining functions are provided for optimization purposes.

GCC includes built-in versions of many of the functions in the standard C library. The versions prefixed with `__builtin_` will always be treated as having the same meaning as the C library function even if you specify the '`-fno-builtin`' option. (see Section 3.4 [C Dialect Options], page 29) Many of these functions are only optimized in certain cases; if they are not optimized in a particular case, a call to the library function will be emitted.

Outside strict ISO C mode ('`-ansi`', '`-std=c90`', '`-std=c99`' or '`-std=c11`'), the functions `_exit`, `alloca`, `bcmp`, `bzero`, `dcgettext`, `dgettext`, `dremf`, `dreml`, `drem`, `exp10f`, `exp10l`, `exp10`, `ffsll`, `ffsl`, `ffs`, `fprintf_unlocked`, `fputs_unlocked`, `gammaf`, `gammal`, `gamma`, `gammaf_r`, `gammal_r`, `gamma_r`, `gettext`, `index`, `isascii`, `j0f`, `j0l`, `j0`, `j1f`, `j1l`, `j1`, `jnf`, `jnl`, `jn`, `lgammaf_r`, `lgammal_r`, `lgamma_r`, `mempcpy`, `pow10f`, `pow10l`, `pow10`, `printf_unlocked`, `rindex`, `scalbf`, `scalbl`, `scalb`, `signbit`, `signbitf`, `signbitl`, `signbitd32`, `signbitd64`, `signbitd128`, `significandf`, `significandl`, `significand`, `sincosf`, `sincosl`, `sincos`, `stpcpy`, `stpncpy`, `strcasecmp`, `strdup`, `strfmon`, `strncasecmp`, `strndup`, `toascii`, `y0f`, `y0l`, `y0`, `y1f`, `y1l`, `y1`, `ynf`, `ynl` and `yn` may be handled as built-in functions. All these functions have corresponding versions prefixed with `__builtin_`, which may be used even in strict C90 mode.

The ISO C99 functions `_Exit`, `acoshf`, `acoshl`, `acosh`, `asinhf`, `asinhl`, `asinh`, `atanhf`, `atanhl`, `atanh`, `cabsf`, `cabsl`, `cabs`, `cacosf`, `cacoshf`, `cacoshl`, `cacosh`, `cacosl`, `cacos`, `cargf`, `cargl`, `carg`, `casinf`, `casinhf`, `casinhl`, `casinh`, `casinl`, `casin`, `catanf`, `catanhf`, `catanhl`, `catanh`, `catanl`, `catan`, `cbrtf`, `cbrtl`, `cbrt`, `ccosf`, `ccoshf`, `ccoshl`, `ccosh`, `ccosl`, `ccos`, `cexpf`, `cexpl`, `cexp`, `cimagf`, `cimagl`, `cimag`, `clogf`, `clogl`, `clog`, `conjf`, `conjl`, `conj`, `copysignf`, `copysignl`, `copysign`, `cpowf`, `cpowl`, `cpow`, `cprojf`,

`cprojl`, `cproj`, `crealf`, `creall`, `creal`, `csinf`, `csinhf`, `csinhl`, `csinh`, `csinl`, `csin`, `csqrtf`, `csqrtl`, `csqrt`, `ctanf`, `ctanhf`, `ctanhl`, `ctanh`, `ctanl`, `ctan`, `erfcf`, `erfcl`, `erfc`, `erff`, `erfl`, `erf`, `exp2f`, `exp2l`, `exp2`, `expm1f`, `expm1l`, `expm1`, `fdimf`, `fdiml`, `fdim`, `fmaf`, `fmal`, `fmaxf`, `fmaxl`, `fmax`, `fma`, `fminf`, `fminl`, `fmin`, `hypotf`, `hypotl`, `hypot`, `ilogbf`, `ilogbl`, `ilogb`, `imaxabs`, `isblank`, `iswblank`, `lgammaf`, `lgammal`, `lgamma`, `llabs`, `llrintf`, `llrintl`, `llrint`, `llroundf`, `llroundl`, `llround`, `log1pf`, `log1pl`, `log1p`, `log2f`, `log2l`, `log2`, `logbf`, `logbl`, `logb`, `lrintf`, `lrintl`, `lrint`, `lroundf`, `lroundl`, `lround`, `nearbyintf`, `nearbyintl`, `nearbyint`, `nextafterf`, `nextafterl`, `nextafter`, `nexttowardf`, `nexttowardl`, `nexttoward`, `remainderf`, `remainderl`, `remainder`, `remquof`, `remquol`, `remquo`, `rintf`, `rintl`, `rint`, `roundf`, `roundl`, `round`, `scalblnf`, `scalblnl`, `scalbln`, `scalbnf`, `scalbnl`, `scalbn`, `snprintf`, `tgammaf`, `tgammal`, `tgamma`, `truncf`, `truncl`, `trunc`, `vfscanf`, `vscanf`, `vsnprintf` and `vsscanf` are handled as built-in functions except in strict ISO C90 mode (‘`-ansi`’ or ‘`-std=c90`’).

There are also built-in versions of the ISO C99 functions `acosf`, `acosl`, `asinf`, `asinl`, `atan2f`, `atan2l`, `atanf`, `atanl`, `ceilf`, `ceill`, `cosf`, `coshf`, `coshl`, `cosl`, `expf`, `expl`, `fabsf`, `fabsl`, `floorf`, `floorl`, `fmodf`, `fmodl`, `frexpf`, `frexpl`, `ldexpf`, `ldexpl`, `log10f`, `log10l`, `logf`, `logl`, `modfl`, `modf`, `powf`, `powl`, `sinf`, `sinhf`, `sinhl`, `sinl`, `sqrtf`, `sqrtl`, `tanf`, `tanhf`, `tanhl` and `tanl` that are recognized in any mode since ISO C90 reserves these names for the purpose to which ISO C99 puts them. All these functions have corresponding versions prefixed with `__builtin_`.

The ISO C94 functions `iswalnum`, `iswalpha`, `iswcntrl`, `iswdigit`, `iswgraph`, `iswlower`, `iswprint`, `iswpunct`, `iswspace`, `iswupper`, `iswxdigit`, `towlower` and `towupper` are handled as built-in functions except in strict ISO C90 mode (‘`-ansi`’ or ‘`-std=c90`’).

The ISO C90 functions `abort`, `abs`, `acos`, `asin`, `atan2`, `atan`, `calloc`, `ceil`, `cosh`, `cos`, `exit`, `exp`, `fabs`, `floor`, `fmod`, `fprintf`, `fputs`, `frexp`, `fscanf`, `isalnum`, `isalpha`, `iscntrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `tolower`, `toupper`, `labs`, `ldexp`, `log10`, `log`, `malloc`, `memchr`, `memcmp`, `memcpy`, `memset`, `modf`, `pow`, `printf`, `putchar`, `puts`, `scanf`, `sinh`, `sin`, `snprintf`, `sprintf`, `sqrt`, `sscanf`, `strcat`, `strchr`, `strcmp`, `strcpy`, `strcspn`, `strlen`, `strncat`, `strncmp`, `strncpy`, `strpbrk`, `strrchr`, `strspn`, `strstr`, `tanh`, `tan`, `vfprintf`, `vprintf` and `vsprintf` are all recognized as built-in functions unless ‘`-fno-builtin`’ is specified (or ‘`-fno-builtin-`*`function`*’ is specified for an individual function). All of these functions have corresponding versions prefixed with `__builtin_`.

GCC provides built-in versions of the ISO C99 floating point comparison macros that avoid raising exceptions for unordered operands. They have the same names as the standard macros (`isgreater`, `isgreaterequal`, `isless`, `islessequal`, `islessgreater`, and `isunordered`) , with `__builtin_` prefixed. We intend for a library implementor to be able to simply `#define` each standard macro to its built-in equivalent. In the same fashion, GCC provides `fpclassify`, `isfinite`, `isinf_sign` and `isnormal` built-ins used with `__builtin_` prefixed. The `isinf` and `isnan` builtins appear both with and without the `__builtin_` prefix.

`int __builtin_types_compatible_p (`*`type1`*`, `*`type2`*`)` [Built-in Function]

You can use the built-in function `__builtin_types_compatible_p` to determine whether two types are the same.

This built-in function returns 1 if the unqualified versions of the types *type1* and *type2* (which are types, not expressions) are compatible, 0 otherwise. The result of this built-in function can be used in integer constant expressions.

This built-in function ignores top level qualifiers (e.g., `const`, `volatile`). For example, `int` is equivalent to `const int`.

The type `int[]` and `int[5]` are compatible. On the other hand, `int` and `char *` are not compatible, even if the size of their types, on the particular architecture are the same. Also, the amount of pointer indirection is taken into account when determining similarity. Consequently, `short *` is not similar to `short **`. Furthermore, two types that are typedefed are considered compatible if their underlying types are compatible.

An `enum` type is not considered to be compatible with another `enum` type even if both are compatible with the same integer type; this is what the C standard specifies. For example, `enum {foo, bar}` is not similar to `enum {hot, dog}`.

You would typically use this function in code whose execution varies depending on the arguments' types. For example:

```
#define foo(x)                                                  \
  ({                                                           \
    typeof (x) tmp = (x);                                       \
    if (__builtin_types_compatible_p (typeof (x), long double)) \
      tmp = foo_long_double (tmp);                              \
    else if (__builtin_types_compatible_p (typeof (x), double)) \
      tmp = foo_double (tmp);                                   \
    else if (__builtin_types_compatible_p (typeof (x), float))  \
      tmp = foo_float (tmp);                                    \
    else                                                        \
      abort ();                                                 \
    tmp;                                                        \
  })
```

Note: This construct is only available for C.

type `__builtin_choose_expr` (*const_exp*, *exp1*, *exp2*) [Built-in Function]

You can use the built-in function `__builtin_choose_expr` to evaluate code depending on the value of a constant expression. This built-in function returns *exp1* if *const_exp*, which is an integer constant expression, is nonzero. Otherwise it returns *exp2*.

This built-in function is analogous to the '`? :`' operator in C, except that the expression returned has its type unaltered by promotion rules. Also, the built-in function does not evaluate the expression that was not chosen. For example, if *const_exp* evaluates to true, *exp2* is not evaluated even if it has side-effects.

This built-in function can return an lvalue if the chosen argument is an lvalue.

If *exp1* is returned, the return type is the same as *exp1*'s type. Similarly, if *exp2* is returned, its return type is the same as *exp2*.

Example:

```
#define foo(x)                                                  \
  __builtin_choose_expr (                                       \
    __builtin_types_compatible_p (typeof (x), double),          \
    foo_double (x),                                             \
    __builtin_choose_expr (                                     \
      __builtin_types_compatible_p (typeof (x), float),         \
```

```
foo_float (x),                                                  \
/* The void expression results in a compile-time error  \
   when assigning the result to something.  */          \
(void)0))
```

Note: This construct is only available for C. Furthermore, the unused expression (*exp1* or *exp2* depending on the value of *const_exp*) may still generate syntax errors. This may change in future revisions.

type `__builtin_complex` (*real*, *imag*) [Built-in Function]

The built-in function `__builtin_complex` is provided for use in implementing the ISO C11 macros `CMPLXF`, `CMPLX` and `CMPLXL`. *real* and *imag* must have the same type, a real binary floating-point type, and the result has the corresponding complex type with real and imaginary parts *real* and *imag*. Unlike '*real* + I * *imag*', this works even when infinities, NaNs and negative zeros are involved.

`int __builtin_constant_p (`*exp*`)` [Built-in Function]

You can use the built-in function `__builtin_constant_p` to determine if a value is known to be constant at compile-time and hence that GCC can perform constant-folding on expressions involving that value. The argument of the function is the value to test. The function returns the integer 1 if the argument is known to be a compile-time constant and 0 if it is not known to be a compile-time constant. A return of 0 does not indicate that the value is *not* a constant, but merely that GCC cannot prove it is a constant with the specified value of the '`-O`' option.

You would typically use this function in an embedded application where memory was a critical resource. If you have some complex calculation, you may want it to be folded if it involves constants, but need to call a function if it does not. For example:

```
#define Scale_Value(X)      \
  (__builtin_constant_p (X) \
  ? ((X) * SCALE + OFFSET) : Scale (X))
```

You may use this built-in function in either a macro or an inline function. However, if you use it in an inlined function and pass an argument of the function as the argument to the built-in, GCC will never return 1 when you call the inline function with a string constant or compound literal (see Section 6.25 [Compound Literals], page 328) and will not return 1 when you pass a constant numeric value to the inline function unless you specify the '`-O`' option.

You may also use `__builtin_constant_p` in initializers for static data. For instance, you can write

```
static const int table[] = {
   __builtin_constant_p (EXPRESSION) ? (EXPRESSION) : -1,
   /* ... */
};
```

This is an acceptable initializer even if *EXPRESSION* is not a constant expression, including the case where `__builtin_constant_p` returns 1 because *EXPRESSION* can be folded to a constant but *EXPRESSION* contains operands that would not otherwise be permitted in a static initializer (for example, `0 && foo ()`). GCC must be more conservative about evaluating the built-in in this case, because it has no opportunity to perform optimization.

Previous versions of GCC did not accept this built-in in data initializers. The earliest version where it is completely safe is 3.0.1.

`long __builtin_expect (long exp, long c)` [Built-in Function]

You may use `__builtin_expect` to provide the compiler with branch prediction information. In general, you should prefer to use actual profile feedback for this (‘`-fprofile-arcs`’), as programmers are notoriously bad at predicting how their programs actually perform. However, there are applications in which this data is hard to collect.

The return value is the value of *exp*, which should be an integral expression. The semantics of the built-in are that it is expected that *exp* == *c*. For example:

```
if (__builtin_expect (x, 0))
  foo ();
```

would indicate that we do not expect to call `foo`, since we expect `x` to be zero. Since you are limited to integral expressions for *exp*, you should use constructions such as

```
if (__builtin_expect (ptr != NULL, 1))
  foo (*ptr);
```

when testing pointer or floating-point values.

`void __builtin_trap (void)` [Built-in Function]

This function causes the program to exit abnormally. GCC implements this function by using a target-dependent mechanism (such as intentionally executing an illegal instruction) or by calling `abort`. The mechanism used may vary from release to release so you should not rely on any particular implementation.

`void __builtin_unreachable (void)` [Built-in Function]

If control flow reaches the point of the `__builtin_unreachable`, the program is undefined. It is useful in situations where the compiler cannot deduce the unreachability of the code.

One such case is immediately following an `asm` statement that will either never terminate, or one that transfers control elsewhere and never returns. In this example, without the `__builtin_unreachable`, GCC would issue a warning that control reaches the end of a non-void function. It would also generate code to return after the `asm`.

```
int f (int c, int v)
{
  if (c)
    {
      return v;
    }
  else
    {
      asm("jmp error_handler");
      __builtin_unreachable ();
    }
}
```

Because the `asm` statement unconditionally transfers control out of the function, control will never reach the end of the function body. The `__builtin_unreachable` is in fact unreachable and communicates this fact to the compiler.

Another use for `__builtin_unreachable` is following a call a function that never returns but that is not declared `__attribute__((noreturn))`, as in this example:

```
void function_that_never_returns (void);

int g (int c)
{
  if (c)
    {
      return 1;
    }
  else
    {
      function_that_never_returns ();
      __builtin_unreachable ();
    }
}
```

`void *__builtin_assume_aligned (`*const void *`**exp**, *size_t* **align**, ...`)` [Built-in Function]

This function returns its first argument, and allows the compiler to assume that the returned pointer is at least *align* bytes aligned. This built-in can have either two or three arguments, if it has three, the third argument should have integer type, and if it is non-zero means misalignment offset. For example:

```
void *x = __builtin_assume_aligned (arg, 16);
```

means that the compiler can assume x, set to arg, is at least 16 byte aligned, while:

```
void *x = __builtin_assume_aligned (arg, 32, 8);
```

means that the compiler can assume for x, set to arg, that (char *) x - 8 is 32 byte aligned.

`void __builtin___clear_cache (`*char *`**begin**, *char *`**end**`)` [Built-in Function]

This function is used to flush the processor's instruction cache for the region of memory between *begin* inclusive and *end* exclusive. Some targets require that the instruction cache be flushed, after modifying memory containing code, in order to obtain deterministic behavior.

If the target does not require instruction cache flushes, `__builtin___clear_cache` has no effect. Otherwise either instructions are emitted in-line to clear the instruction cache or a call to the `__clear_cache` function in libgcc is made.

`void __builtin_prefetch (`*const void *`**addr**, ...`)` [Built-in Function]

This function is used to minimize cache-miss latency by moving data into a cache before it is accessed. You can insert calls to `__builtin_prefetch` into code for which you know addresses of data in memory that is likely to be accessed soon. If the target supports them, data prefetch instructions will be generated. If the prefetch is done early enough before the access then the data will be in the cache by the time it is accessed.

The value of *addr* is the address of the memory to prefetch. There are two optional arguments, *rw* and *locality*. The value of *rw* is a compile-time constant one or zero; one means that the prefetch is preparing for a write to the memory address and zero, the default, means that the prefetch is preparing for a read. The value *locality* must be a compile-time constant integer between zero and three. A value of zero means that the data has no temporal locality, so it need not be left in the cache after the access. A value of three means that the data has a high degree of temporal locality and

should be left in all levels of cache possible. Values of one and two mean, respectively, a low or moderate degree of temporal locality. The default is three.

```
for (i = 0; i < n; i++)
  {
    a[i] = a[i] + b[i];
    __builtin_prefetch (&a[i+j], 1, 1);
    __builtin_prefetch (&b[i+j], 0, 1);
    /* ... */
  }
```

Data prefetch does not generate faults if *addr* is invalid, but the address expression itself must be valid. For example, a prefetch of `p->next` will not fault if `p->next` is not a valid address, but evaluation will fault if `p` is not a valid address.

If the target does not support data prefetch, the address expression is evaluated if it includes side effects but no other code is generated and GCC does not issue a warning.

`double __builtin_huge_val (`*void*`)` [Built-in Function]
Returns a positive infinity, if supported by the floating-point format, else `DBL_MAX`. This function is suitable for implementing the ISO C macro `HUGE_VAL`.

`float __builtin_huge_valf (`*void*`)` [Built-in Function]
Similar to `__builtin_huge_val`, except the return type is `float`.

`long double __builtin_huge_vall (`*void*`)` [Built-in Function]
Similar to `__builtin_huge_val`, except the return type is `long double`.

`int __builtin_fpclassify (`*int, int, int, int, int, ...*`)` [Built-in Function]
This built-in implements the C99 fpclassify functionality. The first five int arguments should be the target library's notion of the possible FP classes and are used for return values. They must be constant values and they must appear in this order: `FP_NAN`, `FP_INFINITE`, `FP_NORMAL`, `FP_SUBNORMAL` and `FP_ZERO`. The ellipsis is for exactly one floating point value to classify. GCC treats the last argument as type-generic, which means it does not do default promotion from float to double.

`double __builtin_inf (`*void*`)` [Built-in Function]
Similar to `__builtin_huge_val`, except a warning is generated if the target floating-point format does not support infinities.

`_Decimal32 __builtin_infd32 (`*void*`)` [Built-in Function]
Similar to `__builtin_inf`, except the return type is `_Decimal32`.

`_Decimal64 __builtin_infd64 (`*void*`)` [Built-in Function]
Similar to `__builtin_inf`, except the return type is `_Decimal64`.

`_Decimal128 __builtin_infd128 (`*void*`)` [Built-in Function]
Similar to `__builtin_inf`, except the return type is `_Decimal128`.

`float __builtin_inff (`*void*`)` [Built-in Function]
Similar to `__builtin_inf`, except the return type is `float`. This function is suitable for implementing the ISO C99 macro `INFINITY`.

`long double __builtin_infl (`*void*`)` [Built-in Function]
Similar to `__builtin_inf`, except the return type is `long double`.

`int __builtin_isinf_sign (...)` [Built-in Function]
Similar to `isinf`, except the return value will be negative for an argument of `-Inf`. Note while the parameter list is an ellipsis, this function only accepts exactly one floating point argument. GCC treats this parameter as type-generic, which means it does not do default promotion from float to double.

`double __builtin_nan (const char *str)` [Built-in Function]
This is an implementation of the ISO C99 function `nan`.

Since ISO C99 defines this function in terms of `strtod`, which we do not implement, a description of the parsing is in order. The string is parsed as by `strtol`; that is, the base is recognized by leading '`0`' or '`0x`' prefixes. The number parsed is placed in the significand such that the least significant bit of the number is at the least significant bit of the significand. The number is truncated to fit the significand field provided. The significand is forced to be a quiet NaN.

This function, if given a string literal all of which would have been consumed by strtol, is evaluated early enough that it is considered a compile-time constant.

`_Decimal32 __builtin_nand32 (const char *str)` [Built-in Function]
Similar to `__builtin_nan`, except the return type is `_Decimal32`.

`_Decimal64 __builtin_nand64 (const char *str)` [Built-in Function]
Similar to `__builtin_nan`, except the return type is `_Decimal64`.

`_Decimal128 __builtin_nand128 (const char *str)` [Built-in Function]
Similar to `__builtin_nan`, except the return type is `_Decimal128`.

`float __builtin_nanf (const char *str)` [Built-in Function]
Similar to `__builtin_nan`, except the return type is `float`.

`long double __builtin_nanl (const char *str)` [Built-in Function]
Similar to `__builtin_nan`, except the return type is `long double`.

`double __builtin_nans (const char *str)` [Built-in Function]
Similar to `__builtin_nan`, except the significand is forced to be a signaling NaN. The `nans` function is proposed by WG14 N965.

`float __builtin_nansf (const char *str)` [Built-in Function]
Similar to `__builtin_nans`, except the return type is `float`.

`long double __builtin_nansl (const char *str)` [Built-in Function]
Similar to `__builtin_nans`, except the return type is `long double`.

`int __builtin_ffs (unsigned int x)` [Built-in Function]
Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.

`int __builtin_clz (unsigned int x)` [Built-in Function]
Returns the number of leading 0-bits in x, starting at the most significant bit position. If x is 0, the result is undefined.

`int __builtin_ctz (`*unsigned int x*`)` [Built-in Function]
Returns the number of trailing 0-bits in *x*, starting at the least significant bit position. If *x* is 0, the result is undefined.

`int __builtin_clrsb (`*int x*`)` [Built-in Function]
Returns the number of leading redundant sign bits in *x*, i.e. the number of bits following the most significant bit which are identical to it. There are no special cases for 0 or other values.

`int __builtin_popcount (`*unsigned int x*`)` [Built-in Function]
Returns the number of 1-bits in *x*.

`int __builtin_parity (`*unsigned int x*`)` [Built-in Function]
Returns the parity of *x*, i.e. the number of 1-bits in *x* modulo 2.

`int __builtin_ffsl (`*unsigned long*`)` [Built-in Function]
Similar to `__builtin_ffs`, except the argument type is `unsigned long`.

`int __builtin_clzl (`*unsigned long*`)` [Built-in Function]
Similar to `__builtin_clz`, except the argument type is `unsigned long`.

`int __builtin_ctzl (`*unsigned long*`)` [Built-in Function]
Similar to `__builtin_ctz`, except the argument type is `unsigned long`.

`int __builtin_clrsbl (`*long*`)` [Built-in Function]
Similar to `__builtin_clrsb`, except the argument type is `long`.

`int __builtin_popcountl (`*unsigned long*`)` [Built-in Function]
Similar to `__builtin_popcount`, except the argument type is `unsigned long`.

`int __builtin_parityl (`*unsigned long*`)` [Built-in Function]
Similar to `__builtin_parity`, except the argument type is `unsigned long`.

`int __builtin_ffsll (`*unsigned long long*`)` [Built-in Function]
Similar to `__builtin_ffs`, except the argument type is `unsigned long long`.

`int __builtin_clzll (`*unsigned long long*`)` [Built-in Function]
Similar to `__builtin_clz`, except the argument type is `unsigned long long`.

`int __builtin_ctzll (`*unsigned long long*`)` [Built-in Function]
Similar to `__builtin_ctz`, except the argument type is `unsigned long long`.

`int __builtin_clrsbll (`*long long*`)` [Built-in Function]
Similar to `__builtin_clrsb`, except the argument type is `long long`.

`int __builtin_popcountll (`*unsigned long long*`)` [Built-in Function]
Similar to `__builtin_popcount`, except the argument type is `unsigned long long`.

`int __builtin_parityll (`*unsigned long long*`)` [Built-in Function]
Similar to `__builtin_parity`, except the argument type is `unsigned long long`.

`double __builtin_powi (`*double, int*`)` [Built-in Function]
Returns the first argument raised to the power of the second. Unlike the `pow` function no guarantees about precision and rounding are made.

`float __builtin_powif (`*float, int*`)` [Built-in Function]
Similar to `__builtin_powi`, except the argument and return types are `float`.

`long double __builtin_powil (`*long double, int*`)` [Built-in Function]
Similar to `__builtin_powi`, except the argument and return types are `long double`.

`int32_t __builtin_bswap32 (`*int32_t x*`)` [Built-in Function]
Returns *x* with the order of the bytes reversed; for example, `0xaabbccdd` becomes `0xddccbbaa`. Byte here always means exactly 8 bits.

`int64_t __builtin_bswap64 (`*int64_t x*`)` [Built-in Function]
Similar to `__builtin_bswap32`, except the argument and return types are 64-bit.

6.55 Built-in Functions Specific to Particular Target Machines

On some target machines, GCC supports many built-in functions specific to those machines. Generally these generate calls to specific machine instructions, but allow the compiler to schedule those calls.

6.55.1 Alpha Built-in Functions

These built-in functions are available for the Alpha family of processors, depending on the command-line switches used.

The following built-in functions are always available. They all generate the machine instruction that is part of the name.

```
long __builtin_alpha_implver (void)
long __builtin_alpha_rpcc (void)
long __builtin_alpha_amask (long)
long __builtin_alpha_cmpbge (long, long)
long __builtin_alpha_extbl (long, long)
long __builtin_alpha_extwl (long, long)
long __builtin_alpha_extll (long, long)
long __builtin_alpha_extql (long, long)
long __builtin_alpha_extwh (long, long)
long __builtin_alpha_extlh (long, long)
long __builtin_alpha_extqh (long, long)
long __builtin_alpha_insbl (long, long)
long __builtin_alpha_inswl (long, long)
long __builtin_alpha_insll (long, long)
long __builtin_alpha_insql (long, long)
long __builtin_alpha_inswh (long, long)
long __builtin_alpha_inslh (long, long)
long __builtin_alpha_insqh (long, long)
long __builtin_alpha_mskbl (long, long)
long __builtin_alpha_mskwl (long, long)
long __builtin_alpha_mskll (long, long)
long __builtin_alpha_mskql (long, long)
long __builtin_alpha_mskwh (long, long)
long __builtin_alpha_msklh (long, long)
```

```
long __builtin_alpha_mskqh (long, long)
long __builtin_alpha_umulh (long, long)
long __builtin_alpha_zap (long, long)
long __builtin_alpha_zapnot (long, long)
```

The following built-in functions are always with '`-mmax`' or '`-mcpu=`*cpu*' where *cpu* is `pca56` or later. They all generate the machine instruction that is part of the name.

```
long __builtin_alpha_pklb (long)
long __builtin_alpha_pkwb (long)
long __builtin_alpha_unpkbl (long)
long __builtin_alpha_unpkbw (long)
long __builtin_alpha_minub8 (long, long)
long __builtin_alpha_minsb8 (long, long)
long __builtin_alpha_minuw4 (long, long)
long __builtin_alpha_minsw4 (long, long)
long __builtin_alpha_maxub8 (long, long)
long __builtin_alpha_maxsb8 (long, long)
long __builtin_alpha_maxuw4 (long, long)
long __builtin_alpha_maxsw4 (long, long)
long __builtin_alpha_perr (long, long)
```

The following built-in functions are always with '`-mcix`' or '`-mcpu=`*cpu*' where *cpu* is `ev67` or later. They all generate the machine instruction that is part of the name.

```
long __builtin_alpha_cttz (long)
long __builtin_alpha_ctlz (long)
long __builtin_alpha_ctpop (long)
```

The following builtins are available on systems that use the OSF/1 PALcode. Normally they invoke the `rduniq` and `wruniq` PAL calls, but when invoked with '`-mtls-kernel`', they invoke `rdval` and `wrval`.

```
void *__builtin_thread_pointer (void)
void __builtin_set_thread_pointer (void *)
```

6.55.2 ARM iWMMXt Built-in Functions

These built-in functions are available for the ARM family of processors when the '`-mcpu=iwmmxt`' switch is used:

```
typedef int v2si __attribute__ ((vector_size (8)));
typedef short v4hi __attribute__ ((vector_size (8)));
typedef char v8qi __attribute__ ((vector_size (8)));

int __builtin_arm_getwcx (int)
void __builtin_arm_setwcx (int, int)
int __builtin_arm_textrmsb (v8qi, int)
int __builtin_arm_textrmsh (v4hi, int)
int __builtin_arm_textrmsw (v2si, int)
int __builtin_arm_textrmub (v8qi, int)
int __builtin_arm_textrmuh (v4hi, int)
int __builtin_arm_textrmuw (v2si, int)
v8qi __builtin_arm_tinsrb (v8qi, int)
v4hi __builtin_arm_tinsrh (v4hi, int)
v2si __builtin_arm_tinsrw (v2si, int)
long long __builtin_arm_tmia (long long, int, int)
long long __builtin_arm_tmiabb (long long, int, int)
long long __builtin_arm_tmiabt (long long, int, int)
long long __builtin_arm_tmiaph (long long, int, int)
long long __builtin_arm_tmiatb (long long, int, int)
long long __builtin_arm_tmiatt (long long, int, int)
```

```
int __builtin_arm_tmovmskb (v8qi)
int __builtin_arm_tmovmskh (v4hi)
int __builtin_arm_tmovmskw (v2si)
long long __builtin_arm_waccb (v8qi)
long long __builtin_arm_wacch (v4hi)
long long __builtin_arm_waccw (v2si)
v8qi __builtin_arm_waddb (v8qi, v8qi)
v8qi __builtin_arm_waddbss (v8qi, v8qi)
v8qi __builtin_arm_waddbus (v8qi, v8qi)
v4hi __builtin_arm_waddh (v4hi, v4hi)
v4hi __builtin_arm_waddhss (v4hi, v4hi)
v4hi __builtin_arm_waddhus (v4hi, v4hi)
v2si __builtin_arm_waddw (v2si, v2si)
v2si __builtin_arm_waddwss (v2si, v2si)
v2si __builtin_arm_waddwus (v2si, v2si)
v8qi __builtin_arm_walign (v8qi, v8qi, int)
long long __builtin_arm_wand(long long, long long)
long long __builtin_arm_wandn (long long, long long)
v8qi __builtin_arm_wavg2b (v8qi, v8qi)
v8qi __builtin_arm_wavg2br (v8qi, v8qi)
v4hi __builtin_arm_wavg2h (v4hi, v4hi)
v4hi __builtin_arm_wavg2hr (v4hi, v4hi)
v8qi __builtin_arm_wcmpeqb (v8qi, v8qi)
v4hi __builtin_arm_wcmpeqh (v4hi, v4hi)
v2si __builtin_arm_wcmpeqw (v2si, v2si)
v8qi __builtin_arm_wcmpgtsb (v8qi, v8qi)
v4hi __builtin_arm_wcmpgtsh (v4hi, v4hi)
v2si __builtin_arm_wcmpgtsw (v2si, v2si)
v8qi __builtin_arm_wcmpgtub (v8qi, v8qi)
v4hi __builtin_arm_wcmpgtuh (v4hi, v4hi)
v2si __builtin_arm_wcmpgtuw (v2si, v2si)
long long __builtin_arm_wmacs (long long, v4hi, v4hi)
long long __builtin_arm_wmacsz (v4hi, v4hi)
long long __builtin_arm_wmacu (long long, v4hi, v4hi)
long long __builtin_arm_wmacuz (v4hi, v4hi)
v4hi __builtin_arm_wmadds (v4hi, v4hi)
v4hi __builtin_arm_wmaddu (v4hi, v4hi)
v8qi __builtin_arm_wmaxsb (v8qi, v8qi)
v4hi __builtin_arm_wmaxsh (v4hi, v4hi)
v2si __builtin_arm_wmaxsw (v2si, v2si)
v8qi __builtin_arm_wmaxub (v8qi, v8qi)
v4hi __builtin_arm_wmaxuh (v4hi, v4hi)
v2si __builtin_arm_wmaxuw (v2si, v2si)
v8qi __builtin_arm_wminsb (v8qi, v8qi)
v4hi __builtin_arm_wminsh (v4hi, v4hi)
v2si __builtin_arm_wminsw (v2si, v2si)
v8qi __builtin_arm_wminub (v8qi, v8qi)
v4hi __builtin_arm_wminuh (v4hi, v4hi)
v2si __builtin_arm_wminuw (v2si, v2si)
v4hi __builtin_arm_wmulsm (v4hi, v4hi)
v4hi __builtin_arm_wmulul (v4hi, v4hi)
v4hi __builtin_arm_wmulum (v4hi, v4hi)
long long __builtin_arm_wor (long long, long long)
v2si __builtin_arm_wpackdss (long long, long long)
v2si __builtin_arm_wpackdus (long long, long long)
v8qi __builtin_arm_wpackhss (v4hi, v4hi)
v8qi __builtin_arm_wpackhus (v4hi, v4hi)
v4hi __builtin_arm_wpackwss (v2si, v2si)
```

```
v4hi __builtin_arm_wpackwus (v2si, v2si)
long long __builtin_arm_wrord (long long, long long)
long long __builtin_arm_wrordi (long long, int)
v4hi __builtin_arm_wrorh (v4hi, long long)
v4hi __builtin_arm_wrorhi (v4hi, int)
v2si __builtin_arm_wrorw (v2si, long long)
v2si __builtin_arm_wrorwi (v2si, int)
v2si __builtin_arm_wsadb (v8qi, v8qi)
v2si __builtin_arm_wsadbz (v8qi, v8qi)
v2si __builtin_arm_wsadh (v4hi, v4hi)
v2si __builtin_arm_wsadhz (v4hi, v4hi)
v4hi __builtin_arm_wshufh (v4hi, int)
long long __builtin_arm_wslld (long long, long long)
long long __builtin_arm_wslldi (long long, int)
v4hi __builtin_arm_wsllh (v4hi, long long)
v4hi __builtin_arm_wsllhi (v4hi, int)
v2si __builtin_arm_wsllw (v2si, long long)
v2si __builtin_arm_wsllwi (v2si, int)
long long __builtin_arm_wsrad (long long, long long)
long long __builtin_arm_wsradi (long long, int)
v4hi __builtin_arm_wsrah (v4hi, long long)
v4hi __builtin_arm_wsrahi (v4hi, int)
v2si __builtin_arm_wsraw (v2si, long long)
v2si __builtin_arm_wsrawi (v2si, int)
long long __builtin_arm_wsrld (long long, long long)
long long __builtin_arm_wsrldi (long long, int)
v4hi __builtin_arm_wsrlh (v4hi, long long)
v4hi __builtin_arm_wsrlhi (v4hi, int)
v2si __builtin_arm_wsrlw (v2si, long long)
v2si __builtin_arm_wsrlwi (v2si, int)
v8qi __builtin_arm_wsubb (v8qi, v8qi)
v8qi __builtin_arm_wsubbss (v8qi, v8qi)
v8qi __builtin_arm_wsubbus (v8qi, v8qi)
v4hi __builtin_arm_wsubh (v4hi, v4hi)
v4hi __builtin_arm_wsubhss (v4hi, v4hi)
v4hi __builtin_arm_wsubhus (v4hi, v4hi)
v2si __builtin_arm_wsubw (v2si, v2si)
v2si __builtin_arm_wsubwss (v2si, v2si)
v2si __builtin_arm_wsubwus (v2si, v2si)
v4hi __builtin_arm_wunpckehsb (v8qi)
v2si __builtin_arm_wunpckehsh (v4hi)
long long __builtin_arm_wunpckehsw (v2si)
v4hi __builtin_arm_wunpckehub (v8qi)
v2si __builtin_arm_wunpckehuh (v4hi)
long long __builtin_arm_wunpckehuw (v2si)
v4hi __builtin_arm_wunpckelsb (v8qi)
v2si __builtin_arm_wunpckelsh (v4hi)
long long __builtin_arm_wunpckelsw (v2si)
v4hi __builtin_arm_wunpckelub (v8qi)
v2si __builtin_arm_wunpckeluh (v4hi)
long long __builtin_arm_wunpckeluw (v2si)
v8qi __builtin_arm_wunpckihb (v8qi, v8qi)
v4hi __builtin_arm_wunpckihh (v4hi, v4hi)
v2si __builtin_arm_wunpckihw (v2si, v2si)
v8qi __builtin_arm_wunpckilb (v8qi, v8qi)
v4hi __builtin_arm_wunpckilh (v4hi, v4hi)
v2si __builtin_arm_wunpckilw (v2si, v2si)
long long __builtin_arm_wxor (long long, long long)
```

```
long long __builtin_arm_wzero ()
```

6.55.3 ARM NEON Intrinsics

These built-in intrinsics for the ARM Advanced SIMD extension are available when the '`-mfpu=neon`' switch is used:

6.55.3.1 Addition

- uint32x2_t vadd_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vadd.i32 d0, d0, d0`
- uint16x4_t vadd_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vadd.i16 d0, d0, d0`
- uint8x8_t vadd_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vadd.i8 d0, d0, d0`
- int32x2_t vadd_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vadd.i32 d0, d0, d0`
- int16x4_t vadd_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vadd.i16 d0, d0, d0`
- int8x8_t vadd_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vadd.i8 d0, d0, d0`
- float32x2_t vadd_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vadd.f32 d0, d0, d0`
- uint64x1_t vadd_u64 (uint64x1_t, uint64x1_t)
- int64x1_t vadd_s64 (int64x1_t, int64x1_t)
- uint32x4_t vaddq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vadd.i32 q0, q0, q0`
- uint16x8_t vaddq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vadd.i16 q0, q0, q0`
- uint8x16_t vaddq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vadd.i8 q0, q0, q0`
- int32x4_t vaddq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vadd.i32 q0, q0, q0`
- int16x8_t vaddq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vadd.i16 q0, q0, q0`
- int8x16_t vaddq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vadd.i8 q0, q0, q0`
- uint64x2_t vaddq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vadd.i64 q0, q0, q0`
- int64x2_t vaddq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vadd.i64 q0, q0, q0`
- float32x4_t vaddq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vadd.f32 q0, q0, q0`
- uint64x2_t vaddl_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vaddl.u32 q0, d0, d0`

- uint32x4_t vaddl_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vaddl.u16 q0, d0, d0`
- uint16x8_t vaddl_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vaddl.u8 q0, d0, d0`
- int64x2_t vaddl_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vaddl.s32 q0, d0, d0`
- int32x4_t vaddl_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vaddl.s16 q0, d0, d0`
- int16x8_t vaddl_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vaddl.s8 q0, d0, d0`
- uint64x2_t vaddw_u32 (uint64x2_t, uint32x2_t)
  Form of expected instruction(s): `vaddw.u32 q0, q0, d0`
- uint32x4_t vaddw_u16 (uint32x4_t, uint16x4_t)
  Form of expected instruction(s): `vaddw.u16 q0, q0, d0`
- uint16x8_t vaddw_u8 (uint16x8_t, uint8x8_t)
  Form of expected instruction(s): `vaddw.u8 q0, q0, d0`
- int64x2_t vaddw_s32 (int64x2_t, int32x2_t)
  Form of expected instruction(s): `vaddw.s32 q0, q0, d0`
- int32x4_t vaddw_s16 (int32x4_t, int16x4_t)
  Form of expected instruction(s): `vaddw.s16 q0, q0, d0`
- int16x8_t vaddw_s8 (int16x8_t, int8x8_t)
  Form of expected instruction(s): `vaddw.s8 q0, q0, d0`
- uint32x2_t vhadd_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vhadd.u32 d0, d0, d0`
- uint16x4_t vhadd_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vhadd.u16 d0, d0, d0`
- uint8x8_t vhadd_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vhadd.u8 d0, d0, d0`
- int32x2_t vhadd_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vhadd.s32 d0, d0, d0`
- int16x4_t vhadd_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vhadd.s16 d0, d0, d0`
- int8x8_t vhadd_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vhadd.s8 d0, d0, d0`
- uint32x4_t vhaddq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vhadd.u32 q0, q0, q0`
- uint16x8_t vhaddq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vhadd.u16 q0, q0, q0`
- uint8x16_t vhaddq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vhadd.u8 q0, q0, q0`
- int32x4_t vhaddq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vhadd.s32 q0, q0, q0`

- int16x8_t vhaddq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vhadd.s16 q0, q0, q0`
- int8x16_t vhaddq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vhadd.s8 q0, q0, q0`
- uint32x2_t vrhadd_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vrhadd.u32 d0, d0, d0`
- uint16x4_t vrhadd_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vrhadd.u16 d0, d0, d0`
- uint8x8_t vrhadd_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vrhadd.u8 d0, d0, d0`
- int32x2_t vrhadd_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vrhadd.s32 d0, d0, d0`
- int16x4_t vrhadd_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vrhadd.s16 d0, d0, d0`
- int8x8_t vrhadd_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vrhadd.s8 d0, d0, d0`
- uint32x4_t vrhaddq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vrhadd.u32 q0, q0, q0`
- uint16x8_t vrhaddq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vrhadd.u16 q0, q0, q0`
- uint8x16_t vrhaddq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vrhadd.u8 q0, q0, q0`
- int32x4_t vrhaddq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vrhadd.s32 q0, q0, q0`
- int16x8_t vrhaddq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vrhadd.s16 q0, q0, q0`
- int8x16_t vrhaddq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vrhadd.s8 q0, q0, q0`
- uint32x2_t vqadd_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vqadd.u32 d0, d0, d0`
- uint16x4_t vqadd_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vqadd.u16 d0, d0, d0`
- uint8x8_t vqadd_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vqadd.u8 d0, d0, d0`
- int32x2_t vqadd_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqadd.s32 d0, d0, d0`
- int16x4_t vqadd_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqadd.s16 d0, d0, d0`
- int8x8_t vqadd_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vqadd.s8 d0, d0, d0`
- uint64x1_t vqadd_u64 (uint64x1_t, uint64x1_t)
  Form of expected instruction(s): `vqadd.u64 d0, d0, d0`

- int64x1_t vqadd_s64 (int64x1_t, int64x1_t)
  Form of expected instruction(s): `vqadd.s64 d0, d0, d0`
- uint32x4_t vqaddq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vqadd.u32 q0, q0, q0`
- uint16x8_t vqaddq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vqadd.u16 q0, q0, q0`
- uint8x16_t vqaddq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vqadd.u8 q0, q0, q0`
- int32x4_t vqaddq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vqadd.s32 q0, q0, q0`
- int16x8_t vqaddq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vqadd.s16 q0, q0, q0`
- int8x16_t vqaddq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vqadd.s8 q0, q0, q0`
- uint64x2_t vqaddq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vqadd.u64 q0, q0, q0`
- int64x2_t vqaddq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vqadd.s64 q0, q0, q0`
- uint32x2_t vaddhn_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vaddhn.i64 d0, q0, q0`
- uint16x4_t vaddhn_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vaddhn.i32 d0, q0, q0`
- uint8x8_t vaddhn_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vaddhn.i16 d0, q0, q0`
- int32x2_t vaddhn_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vaddhn.i64 d0, q0, q0`
- int16x4_t vaddhn_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vaddhn.i32 d0, q0, q0`
- int8x8_t vaddhn_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vaddhn.i16 d0, q0, q0`
- uint32x2_t vraddhn_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vraddhn.i64 d0, q0, q0`
- uint16x4_t vraddhn_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vraddhn.i32 d0, q0, q0`
- uint8x8_t vraddhn_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vraddhn.i16 d0, q0, q0`
- int32x2_t vraddhn_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vraddhn.i64 d0, q0, q0`
- int16x4_t vraddhn_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vraddhn.i32 d0, q0, q0`
- int8x8_t vraddhn_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vraddhn.i16 d0, q0, q0`

6.55.3.2 Multiplication

- uint32x2_t vmul_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmul.i32 d0, d0, d0`
- uint16x4_t vmul_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmul.i16 d0, d0, d0`
- uint8x8_t vmul_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmul.i8 d0, d0, d0`
- int32x2_t vmul_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmul.i32 d0, d0, d0`
- int16x4_t vmul_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmul.i16 d0, d0, d0`
- int8x8_t vmul_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmul.i8 d0, d0, d0`
- float32x2_t vmul_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vmul.f32 d0, d0, d0`
- poly8x8_t vmul_p8 (poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vmul.p8 d0, d0, d0`
- uint32x4_t vmulq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vmul.i32 q0, q0, q0`
- uint16x8_t vmulq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vmul.i16 q0, q0, q0`
- uint8x16_t vmulq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vmul.i8 q0, q0, q0`
- int32x4_t vmulq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vmul.i32 q0, q0, q0`
- int16x8_t vmulq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vmul.i16 q0, q0, q0`
- int8x16_t vmulq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vmul.i8 q0, q0, q0`
- float32x4_t vmulq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vmul.f32 q0, q0, q0`
- poly8x16_t vmulq_p8 (poly8x16_t, poly8x16_t)
  Form of expected instruction(s): `vmul.p8 q0, q0, q0`
- int32x2_t vqdmulh_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqdmulh.s32 d0, d0, d0`
- int16x4_t vqdmulh_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqdmulh.s16 d0, d0, d0`
- int32x4_t vqdmulhq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vqdmulh.s32 q0, q0, q0`
- int16x8_t vqdmulhq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vqdmulh.s16 q0, q0, q0`
- int32x2_t vqrdmulh_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqrdmulh.s32 d0, d0, d0`

- int16x4_t vqrdmulh_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqrdmulh.s16 d0, d0, d0`
- int32x4_t vqrdmulhq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vqrdmulh.s32 q0, q0, q0`
- int16x8_t vqrdmulhq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vqrdmulh.s16 q0, q0, q0`
- uint64x2_t vmull_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmull.u32 q0, d0, d0`
- uint32x4_t vmull_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmull.u16 q0, d0, d0`
- uint16x8_t vmull_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmull.u8 q0, d0, d0`
- int64x2_t vmull_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmull.s32 q0, d0, d0`
- int32x4_t vmull_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmull.s16 q0, d0, d0`
- int16x8_t vmull_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmull.s8 q0, d0, d0`
- poly16x8_t vmull_p8 (poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vmull.p8 q0, d0, d0`
- int64x2_t vqdmull_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqdmull.s32 q0, d0, d0`
- int32x4_t vqdmull_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqdmull.s16 q0, d0, d0`

6.55.3.3 Multiply-accumulate

- uint32x2_t vmla_u32 (uint32x2_t, uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmla.i32 d0, d0, d0`
- uint16x4_t vmla_u16 (uint16x4_t, uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmla.i16 d0, d0, d0`
- uint8x8_t vmla_u8 (uint8x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmla.i8 d0, d0, d0`
- int32x2_t vmla_s32 (int32x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmla.i32 d0, d0, d0`
- int16x4_t vmla_s16 (int16x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmla.i16 d0, d0, d0`
- int8x8_t vmla_s8 (int8x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmla.i8 d0, d0, d0`
- float32x2_t vmla_f32 (float32x2_t, float32x2_t, float32x2_t)
  Form of expected instruction(s): `vmla.f32 d0, d0, d0`
- uint32x4_t vmlaq_u32 (uint32x4_t, uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vmla.i32 q0, q0, q0`

- uint16x8_t vmlaq_u16 (uint16x8_t, uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vmla.i16 q0, q0, q0`
- uint8x16_t vmlaq_u8 (uint8x16_t, uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vmla.i8 q0, q0, q0`
- int32x4_t vmlaq_s32 (int32x4_t, int32x4_t, int32x4_t)
  Form of expected instruction(s): `vmla.i32 q0, q0, q0`
- int16x8_t vmlaq_s16 (int16x8_t, int16x8_t, int16x8_t)
  Form of expected instruction(s): `vmla.i16 q0, q0, q0`
- int8x16_t vmlaq_s8 (int8x16_t, int8x16_t, int8x16_t)
  Form of expected instruction(s): `vmla.i8 q0, q0, q0`
- float32x4_t vmlaq_f32 (float32x4_t, float32x4_t, float32x4_t)
  Form of expected instruction(s): `vmla.f32 q0, q0, q0`
- uint64x2_t vmlal_u32 (uint64x2_t, uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmlal.u32 q0, d0, d0`
- uint32x4_t vmlal_u16 (uint32x4_t, uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmlal.u16 q0, d0, d0`
- uint16x8_t vmlal_u8 (uint16x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmlal.u8 q0, d0, d0`
- int64x2_t vmlal_s32 (int64x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmlal.s32 q0, d0, d0`
- int32x4_t vmlal_s16 (int32x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmlal.s16 q0, d0, d0`
- int16x8_t vmlal_s8 (int16x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmlal.s8 q0, d0, d0`
- int64x2_t vqdmlal_s32 (int64x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqdmlal.s32 q0, d0, d0`
- int32x4_t vqdmlal_s16 (int32x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqdmlal.s16 q0, d0, d0`

6.55.3.4 Multiply-subtract

- uint32x2_t vmls_u32 (uint32x2_t, uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmls.i32 d0, d0, d0`
- uint16x4_t vmls_u16 (uint16x4_t, uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmls.i16 d0, d0, d0`
- uint8x8_t vmls_u8 (uint8x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmls.i8 d0, d0, d0`
- int32x2_t vmls_s32 (int32x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmls.i32 d0, d0, d0`
- int16x4_t vmls_s16 (int16x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmls.i16 d0, d0, d0`
- int8x8_t vmls_s8 (int8x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmls.i8 d0, d0, d0`

- float32x2_t vmls_f32 (float32x2_t, float32x2_t, float32x2_t)
  Form of expected instruction(s): `vmls.f32 d0, d0, d0`
- uint32x4_t vmlsq_u32 (uint32x4_t, uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vmls.i32 q0, q0, q0`
- uint16x8_t vmlsq_u16 (uint16x8_t, uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vmls.i16 q0, q0, q0`
- uint8x16_t vmlsq_u8 (uint8x16_t, uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vmls.i8 q0, q0, q0`
- int32x4_t vmlsq_s32 (int32x4_t, int32x4_t, int32x4_t)
  Form of expected instruction(s): `vmls.i32 q0, q0, q0`
- int16x8_t vmlsq_s16 (int16x8_t, int16x8_t, int16x8_t)
  Form of expected instruction(s): `vmls.i16 q0, q0, q0`
- int8x16_t vmlsq_s8 (int8x16_t, int8x16_t, int8x16_t)
  Form of expected instruction(s): `vmls.i8 q0, q0, q0`
- float32x4_t vmlsq_f32 (float32x4_t, float32x4_t, float32x4_t)
  Form of expected instruction(s): `vmls.f32 q0, q0, q0`
- uint64x2_t vmlsl_u32 (uint64x2_t, uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmlsl.u32 q0, d0, d0`
- uint32x4_t vmlsl_u16 (uint32x4_t, uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmlsl.u16 q0, d0, d0`
- uint16x8_t vmlsl_u8 (uint16x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmlsl.u8 q0, d0, d0`
- int64x2_t vmlsl_s32 (int64x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmlsl.s32 q0, d0, d0`
- int32x4_t vmlsl_s16 (int32x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmlsl.s16 q0, d0, d0`
- int16x8_t vmlsl_s8 (int16x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmlsl.s8 q0, d0, d0`
- int64x2_t vqdmlsl_s32 (int64x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqdmlsl.s32 q0, d0, d0`
- int32x4_t vqdmlsl_s16 (int32x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqdmlsl.s16 q0, d0, d0`

6.55.3.5 Subtraction

- uint32x2_t vsub_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vsub.i32 d0, d0, d0`
- uint16x4_t vsub_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vsub.i16 d0, d0, d0`
- uint8x8_t vsub_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vsub.i8 d0, d0, d0`
- int32x2_t vsub_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vsub.i32 d0, d0, d0`

- int16x4_t vsub_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vsub.i16 d0, d0, d0`
- int8x8_t vsub_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vsub.i8 d0, d0, d0`
- float32x2_t vsub_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vsub.f32 d0, d0, d0`
- uint64x1_t vsub_u64 (uint64x1_t, uint64x1_t)
- int64x1_t vsub_s64 (int64x1_t, int64x1_t)
- uint32x4_t vsubq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vsub.i32 q0, q0, q0`
- uint16x8_t vsubq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vsub.i16 q0, q0, q0`
- uint8x16_t vsubq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vsub.i8 q0, q0, q0`
- int32x4_t vsubq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vsub.i32 q0, q0, q0`
- int16x8_t vsubq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vsub.i16 q0, q0, q0`
- int8x16_t vsubq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vsub.i8 q0, q0, q0`
- uint64x2_t vsubq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vsub.i64 q0, q0, q0`
- int64x2_t vsubq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vsub.i64 q0, q0, q0`
- float32x4_t vsubq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vsub.f32 q0, q0, q0`
- uint64x2_t vsubl_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vsubl.u32 q0, d0, d0`
- uint32x4_t vsubl_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vsubl.u16 q0, d0, d0`
- uint16x8_t vsubl_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vsubl.u8 q0, d0, d0`
- int64x2_t vsubl_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vsubl.s32 q0, d0, d0`
- int32x4_t vsubl_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vsubl.s16 q0, d0, d0`
- int16x8_t vsubl_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vsubl.s8 q0, d0, d0`
- uint64x2_t vsubw_u32 (uint64x2_t, uint32x2_t)
  Form of expected instruction(s): `vsubw.u32 q0, q0, d0`
- uint32x4_t vsubw_u16 (uint32x4_t, uint16x4_t)
  Form of expected instruction(s): `vsubw.u16 q0, q0, d0`

- uint16x8_t vsubw_u8 (uint16x8_t, uint8x8_t)
  Form of expected instruction(s): `vsubw.u8 q0, q0, d0`
- int64x2_t vsubw_s32 (int64x2_t, int32x2_t)
  Form of expected instruction(s): `vsubw.s32 q0, q0, d0`
- int32x4_t vsubw_s16 (int32x4_t, int16x4_t)
  Form of expected instruction(s): `vsubw.s16 q0, q0, d0`
- int16x8_t vsubw_s8 (int16x8_t, int8x8_t)
  Form of expected instruction(s): `vsubw.s8 q0, q0, d0`
- uint32x2_t vhsub_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vhsub.u32 d0, d0, d0`
- uint16x4_t vhsub_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vhsub.u16 d0, d0, d0`
- uint8x8_t vhsub_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vhsub.u8 d0, d0, d0`
- int32x2_t vhsub_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vhsub.s32 d0, d0, d0`
- int16x4_t vhsub_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vhsub.s16 d0, d0, d0`
- int8x8_t vhsub_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vhsub.s8 d0, d0, d0`
- uint32x4_t vhsubq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vhsub.u32 q0, q0, q0`
- uint16x8_t vhsubq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vhsub.u16 q0, q0, q0`
- uint8x16_t vhsubq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vhsub.u8 q0, q0, q0`
- int32x4_t vhsubq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vhsub.s32 q0, q0, q0`
- int16x8_t vhsubq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vhsub.s16 q0, q0, q0`
- int8x16_t vhsubq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vhsub.s8 q0, q0, q0`
- uint32x2_t vqsub_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vqsub.u32 d0, d0, d0`
- uint16x4_t vqsub_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vqsub.u16 d0, d0, d0`
- uint8x8_t vqsub_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vqsub.u8 d0, d0, d0`
- int32x2_t vqsub_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqsub.s32 d0, d0, d0`
- int16x4_t vqsub_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqsub.s16 d0, d0, d0`

- int8x8_t vqsub_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vqsub.s8 d0, d0, d0`
- uint64x1_t vqsub_u64 (uint64x1_t, uint64x1_t)
  Form of expected instruction(s): `vqsub.u64 d0, d0, d0`
- int64x1_t vqsub_s64 (int64x1_t, int64x1_t)
  Form of expected instruction(s): `vqsub.s64 d0, d0, d0`
- uint32x4_t vqsubq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vqsub.u32 q0, q0, q0`
- uint16x8_t vqsubq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vqsub.u16 q0, q0, q0`
- uint8x16_t vqsubq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vqsub.u8 q0, q0, q0`
- int32x4_t vqsubq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vqsub.s32 q0, q0, q0`
- int16x8_t vqsubq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vqsub.s16 q0, q0, q0`
- int8x16_t vqsubq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vqsub.s8 q0, q0, q0`
- uint64x2_t vqsubq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vqsub.u64 q0, q0, q0`
- int64x2_t vqsubq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vqsub.s64 q0, q0, q0`
- uint32x2_t vsubhn_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vsubhn.i64 d0, q0, q0`
- uint16x4_t vsubhn_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vsubhn.i32 d0, q0, q0`
- uint8x8_t vsubhn_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vsubhn.i16 d0, q0, q0`
- int32x2_t vsubhn_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vsubhn.i64 d0, q0, q0`
- int16x4_t vsubhn_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vsubhn.i32 d0, q0, q0`
- int8x8_t vsubhn_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vsubhn.i16 d0, q0, q0`
- uint32x2_t vrsubhn_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vrsubhn.i64 d0, q0, q0`
- uint16x4_t vrsubhn_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vrsubhn.i32 d0, q0, q0`
- uint8x8_t vrsubhn_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vrsubhn.i16 d0, q0, q0`
- int32x2_t vrsubhn_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vrsubhn.i64 d0, q0, q0`

- int16x4_t vrsubhn_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vrsubhn.i32 d0, q0, q0`
- int8x8_t vrsubhn_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vrsubhn.i16 d0, q0, q0`

6.55.3.6 Comparison (equal-to)

- uint32x2_t vceq_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vceq.i32 d0, d0, d0`
- uint16x4_t vceq_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vceq.i16 d0, d0, d0`
- uint8x8_t vceq_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vceq.i8 d0, d0, d0`
- uint32x2_t vceq_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vceq.i32 d0, d0, d0`
- uint16x4_t vceq_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vceq.i16 d0, d0, d0`
- uint8x8_t vceq_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vceq.i8 d0, d0, d0`
- uint32x2_t vceq_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vceq.f32 d0, d0, d0`
- uint8x8_t vceq_p8 (poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vceq.i8 d0, d0, d0`
- uint32x4_t vceqq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vceq.i32 q0, q0, q0`
- uint16x8_t vceqq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vceq.i16 q0, q0, q0`
- uint8x16_t vceqq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vceq.i8 q0, q0, q0`
- uint32x4_t vceqq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vceq.i32 q0, q0, q0`
- uint16x8_t vceqq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vceq.i16 q0, q0, q0`
- uint8x16_t vceqq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vceq.i8 q0, q0, q0`
- uint32x4_t vceqq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vceq.f32 q0, q0, q0`
- uint8x16_t vceqq_p8 (poly8x16_t, poly8x16_t)
  Form of expected instruction(s): `vceq.i8 q0, q0, q0`

6.55.3.7 Comparison (greater-than-or-equal-to)

- uint32x2_t vcge_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vcge.u32 d0, d0, d0`

- uint16x4_t vcge_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vcge.u16 d0, d0, d0`
- uint8x8_t vcge_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vcge.u8 d0, d0, d0`
- uint32x2_t vcge_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vcge.s32 d0, d0, d0`
- uint16x4_t vcge_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vcge.s16 d0, d0, d0`
- uint8x8_t vcge_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vcge.s8 d0, d0, d0`
- uint32x2_t vcge_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vcge.f32 d0, d0, d0`
- uint32x4_t vcgeq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vcge.u32 q0, q0, q0`
- uint16x8_t vcgeq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vcge.u16 q0, q0, q0`
- uint8x16_t vcgeq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vcge.u8 q0, q0, q0`
- uint32x4_t vcgeq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vcge.s32 q0, q0, q0`
- uint16x8_t vcgeq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vcge.s16 q0, q0, q0`
- uint8x16_t vcgeq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vcge.s8 q0, q0, q0`
- uint32x4_t vcgeq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vcge.f32 q0, q0, q0`

6.55.3.8 Comparison (less-than-or-equal-to)

- uint32x2_t vcle_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vcge.u32 d0, d0, d0`
- uint16x4_t vcle_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vcge.u16 d0, d0, d0`
- uint8x8_t vcle_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vcge.u8 d0, d0, d0`
- uint32x2_t vcle_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vcge.s32 d0, d0, d0`
- uint16x4_t vcle_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vcge.s16 d0, d0, d0`
- uint8x8_t vcle_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vcge.s8 d0, d0, d0`
- uint32x2_t vcle_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vcge.f32 d0, d0, d0`

- uint32x4_t vcleq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vcge.u32 q0, q0, q0`
- uint16x8_t vcleq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vcge.u16 q0, q0, q0`
- uint8x16_t vcleq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vcge.u8 q0, q0, q0`
- uint32x4_t vcleq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vcge.s32 q0, q0, q0`
- uint16x8_t vcleq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vcge.s16 q0, q0, q0`
- uint8x16_t vcleq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vcge.s8 q0, q0, q0`
- uint32x4_t vcleq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vcge.f32 q0, q0, q0`

6.55.3.9 Comparison (greater-than)

- uint32x2_t vcgt_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vcgt.u32 d0, d0, d0`
- uint16x4_t vcgt_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vcgt.u16 d0, d0, d0`
- uint8x8_t vcgt_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vcgt.u8 d0, d0, d0`
- uint32x2_t vcgt_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vcgt.s32 d0, d0, d0`
- uint16x4_t vcgt_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vcgt.s16 d0, d0, d0`
- uint8x8_t vcgt_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vcgt.s8 d0, d0, d0`
- uint32x2_t vcgt_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vcgt.f32 d0, d0, d0`
- uint32x4_t vcgtq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vcgt.u32 q0, q0, q0`
- uint16x8_t vcgtq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vcgt.u16 q0, q0, q0`
- uint8x16_t vcgtq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vcgt.u8 q0, q0, q0`
- uint32x4_t vcgtq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vcgt.s32 q0, q0, q0`
- uint16x8_t vcgtq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vcgt.s16 q0, q0, q0`
- uint8x16_t vcgtq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vcgt.s8 q0, q0, q0`

- uint32x4_t vcgtq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vcgt.f32 q0, q0, q0`

6.55.3.10 Comparison (less-than)

- uint32x2_t vclt_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vcgt.u32 d0, d0, d0`
- uint16x4_t vclt_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vcgt.u16 d0, d0, d0`
- uint8x8_t vclt_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vcgt.u8 d0, d0, d0`
- uint32x2_t vclt_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vcgt.s32 d0, d0, d0`
- uint16x4_t vclt_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vcgt.s16 d0, d0, d0`
- uint8x8_t vclt_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vcgt.s8 d0, d0, d0`
- uint32x2_t vclt_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vcgt.f32 d0, d0, d0`
- uint32x4_t vcltq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vcgt.u32 q0, q0, q0`
- uint16x8_t vcltq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vcgt.u16 q0, q0, q0`
- uint8x16_t vcltq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vcgt.u8 q0, q0, q0`
- uint32x4_t vcltq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vcgt.s32 q0, q0, q0`
- uint16x8_t vcltq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vcgt.s16 q0, q0, q0`
- uint8x16_t vcltq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vcgt.s8 q0, q0, q0`
- uint32x4_t vcltq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vcgt.f32 q0, q0, q0`

6.55.3.11 Comparison (absolute greater-than-or-equal-to)

- uint32x2_t vcage_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vacge.f32 d0, d0, d0`
- uint32x4_t vcageq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vacge.f32 q0, q0, q0`

6.55.3.12 Comparison (absolute less-than-or-equal-to)

- uint32x2_t vcale_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vacge.f32 d0, d0, d0`
- uint32x4_t vcaleq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vacge.f32 q0, q0, q0`

6.55.3.13 Comparison (absolute greater-than)

- uint32x2_t vcagt_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vacgt.f32 d0, d0, d0`
- uint32x4_t vcagtq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vacgt.f32 q0, q0, q0`

6.55.3.14 Comparison (absolute less-than)

- uint32x2_t vcalt_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vacgt.f32 d0, d0, d0`
- uint32x4_t vcaltq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vacgt.f32 q0, q0, q0`

6.55.3.15 Test bits

- uint32x2_t vtst_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vtst.32 d0, d0, d0`
- uint16x4_t vtst_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vtst.16 d0, d0, d0`
- uint8x8_t vtst_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vtst.8 d0, d0, d0`
- uint32x2_t vtst_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vtst.32 d0, d0, d0`
- uint16x4_t vtst_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vtst.16 d0, d0, d0`
- uint8x8_t vtst_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vtst.8 d0, d0, d0`
- uint8x8_t vtst_p8 (poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vtst.8 d0, d0, d0`
- uint32x4_t vtstq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vtst.32 q0, q0, q0`
- uint16x8_t vtstq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vtst.16 q0, q0, q0`
- uint8x16_t vtstq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vtst.8 q0, q0, q0`
- uint32x4_t vtstq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vtst.32 q0, q0, q0`
- uint16x8_t vtstq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vtst.16 q0, q0, q0`
- uint8x16_t vtstq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vtst.8 q0, q0, q0`
- uint8x16_t vtstq_p8 (poly8x16_t, poly8x16_t)
  Form of expected instruction(s): `vtst.8 q0, q0, q0`

6.55.3.16 Absolute difference

- uint32x2_t vabd_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vabd.u32 d0, d0, d0`
- uint16x4_t vabd_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vabd.u16 d0, d0, d0`
- uint8x8_t vabd_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vabd.u8 d0, d0, d0`
- int32x2_t vabd_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vabd.s32 d0, d0, d0`
- int16x4_t vabd_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vabd.s16 d0, d0, d0`
- int8x8_t vabd_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vabd.s8 d0, d0, d0`
- float32x2_t vabd_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vabd.f32 d0, d0, d0`
- uint32x4_t vabdq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vabd.u32 q0, q0, q0`
- uint16x8_t vabdq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vabd.u16 q0, q0, q0`
- uint8x16_t vabdq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vabd.u8 q0, q0, q0`
- int32x4_t vabdq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vabd.s32 q0, q0, q0`
- int16x8_t vabdq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vabd.s16 q0, q0, q0`
- int8x16_t vabdq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vabd.s8 q0, q0, q0`
- float32x4_t vabdq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vabd.f32 q0, q0, q0`
- uint64x2_t vabdl_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vabdl.u32 q0, d0, d0`
- uint32x4_t vabdl_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vabdl.u16 q0, d0, d0`
- uint16x8_t vabdl_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vabdl.u8 q0, d0, d0`
- int64x2_t vabdl_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vabdl.s32 q0, d0, d0`
- int32x4_t vabdl_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vabdl.s16 q0, d0, d0`
- int16x8_t vabdl_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vabdl.s8 q0, d0, d0`

6.55.3.17 Absolute difference and accumulate

- uint32x2_t vaba_u32 (uint32x2_t, uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vaba.u32 d0, d0, d0`
- uint16x4_t vaba_u16 (uint16x4_t, uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vaba.u16 d0, d0, d0`
- uint8x8_t vaba_u8 (uint8x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vaba.u8 d0, d0, d0`
- int32x2_t vaba_s32 (int32x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vaba.s32 d0, d0, d0`
- int16x4_t vaba_s16 (int16x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vaba.s16 d0, d0, d0`
- int8x8_t vaba_s8 (int8x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vaba.s8 d0, d0, d0`
- uint32x4_t vabaq_u32 (uint32x4_t, uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vaba.u32 q0, q0, q0`
- uint16x8_t vabaq_u16 (uint16x8_t, uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vaba.u16 q0, q0, q0`
- uint8x16_t vabaq_u8 (uint8x16_t, uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vaba.u8 q0, q0, q0`
- int32x4_t vabaq_s32 (int32x4_t, int32x4_t, int32x4_t)
  Form of expected instruction(s): `vaba.s32 q0, q0, q0`
- int16x8_t vabaq_s16 (int16x8_t, int16x8_t, int16x8_t)
  Form of expected instruction(s): `vaba.s16 q0, q0, q0`
- int8x16_t vabaq_s8 (int8x16_t, int8x16_t, int8x16_t)
  Form of expected instruction(s): `vaba.s8 q0, q0, q0`
- uint64x2_t vabal_u32 (uint64x2_t, uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vabal.u32 q0, d0, d0`
- uint32x4_t vabal_u16 (uint32x4_t, uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vabal.u16 q0, d0, d0`
- uint16x8_t vabal_u8 (uint16x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vabal.u8 q0, d0, d0`
- int64x2_t vabal_s32 (int64x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vabal.s32 q0, d0, d0`
- int32x4_t vabal_s16 (int32x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vabal.s16 q0, d0, d0`
- int16x8_t vabal_s8 (int16x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vabal.s8 q0, d0, d0`

6.55.3.18 Maximum

- uint32x2_t vmax_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmax.u32 d0, d0, d0`
- uint16x4_t vmax_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmax.u16 d0, d0, d0`

- uint8x8_t vmax_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmax.u8 d0, d0, d0`
- int32x2_t vmax_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmax.s32 d0, d0, d0`
- int16x4_t vmax_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmax.s16 d0, d0, d0`
- int8x8_t vmax_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmax.s8 d0, d0, d0`
- float32x2_t vmax_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vmax.f32 d0, d0, d0`
- uint32x4_t vmaxq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vmax.u32 q0, q0, q0`
- uint16x8_t vmaxq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vmax.u16 q0, q0, q0`
- uint8x16_t vmaxq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vmax.u8 q0, q0, q0`
- int32x4_t vmaxq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vmax.s32 q0, q0, q0`
- int16x8_t vmaxq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vmax.s16 q0, q0, q0`
- int8x16_t vmaxq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vmax.s8 q0, q0, q0`
- float32x4_t vmaxq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vmax.f32 q0, q0, q0`

6.55.3.19 Minimum

- uint32x2_t vmin_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vmin.u32 d0, d0, d0`
- uint16x4_t vmin_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vmin.u16 d0, d0, d0`
- uint8x8_t vmin_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vmin.u8 d0, d0, d0`
- int32x2_t vmin_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vmin.s32 d0, d0, d0`
- int16x4_t vmin_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vmin.s16 d0, d0, d0`
- int8x8_t vmin_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vmin.s8 d0, d0, d0`
- float32x2_t vmin_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vmin.f32 d0, d0, d0`
- uint32x4_t vminq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vmin.u32 q0, q0, q0`

- uint16x8_t vminq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vmin.u16 q0, q0, q0`
- uint8x16_t vminq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vmin.u8 q0, q0, q0`
- int32x4_t vminq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vmin.s32 q0, q0, q0`
- int16x8_t vminq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vmin.s16 q0, q0, q0`
- int8x16_t vminq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vmin.s8 q0, q0, q0`
- float32x4_t vminq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vmin.f32 q0, q0, q0`

6.55.3.20 Pairwise add

- uint32x2_t vpadd_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vpadd.i32 d0, d0, d0`
- uint16x4_t vpadd_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vpadd.i16 d0, d0, d0`
- uint8x8_t vpadd_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vpadd.i8 d0, d0, d0`
- int32x2_t vpadd_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vpadd.i32 d0, d0, d0`
- int16x4_t vpadd_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vpadd.i16 d0, d0, d0`
- int8x8_t vpadd_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vpadd.i8 d0, d0, d0`
- float32x2_t vpadd_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vpadd.f32 d0, d0, d0`
- uint64x1_t vpaddl_u32 (uint32x2_t)
  Form of expected instruction(s): `vpaddl.u32 d0, d0`
- uint32x2_t vpaddl_u16 (uint16x4_t)
  Form of expected instruction(s): `vpaddl.u16 d0, d0`
- uint16x4_t vpaddl_u8 (uint8x8_t)
  Form of expected instruction(s): `vpaddl.u8 d0, d0`
- int64x1_t vpaddl_s32 (int32x2_t)
  Form of expected instruction(s): `vpaddl.s32 d0, d0`
- int32x2_t vpaddl_s16 (int16x4_t)
  Form of expected instruction(s): `vpaddl.s16 d0, d0`
- int16x4_t vpaddl_s8 (int8x8_t)
  Form of expected instruction(s): `vpaddl.s8 d0, d0`
- uint64x2_t vpaddlq_u32 (uint32x4_t)
  Form of expected instruction(s): `vpaddl.u32 q0, q0`

- uint32x4_t vpaddlq_u16 (uint16x8_t)
  Form of expected instruction(s): `vpaddl.u16 q0, q0`
- uint16x8_t vpaddlq_u8 (uint8x16_t)
  Form of expected instruction(s): `vpaddl.u8 q0, q0`
- int64x2_t vpaddlq_s32 (int32x4_t)
  Form of expected instruction(s): `vpaddl.s32 q0, q0`
- int32x4_t vpaddlq_s16 (int16x8_t)
  Form of expected instruction(s): `vpaddl.s16 q0, q0`
- int16x8_t vpaddlq_s8 (int8x16_t)
  Form of expected instruction(s): `vpaddl.s8 q0, q0`

6.55.3.21 Pairwise add, single_opcode widen and accumulate

- uint64x1_t vpadal_u32 (uint64x1_t, uint32x2_t)
  Form of expected instruction(s): `vpadal.u32 d0, d0`
- uint32x2_t vpadal_u16 (uint32x2_t, uint16x4_t)
  Form of expected instruction(s): `vpadal.u16 d0, d0`
- uint16x4_t vpadal_u8 (uint16x4_t, uint8x8_t)
  Form of expected instruction(s): `vpadal.u8 d0, d0`
- int64x1_t vpadal_s32 (int64x1_t, int32x2_t)
  Form of expected instruction(s): `vpadal.s32 d0, d0`
- int32x2_t vpadal_s16 (int32x2_t, int16x4_t)
  Form of expected instruction(s): `vpadal.s16 d0, d0`
- int16x4_t vpadal_s8 (int16x4_t, int8x8_t)
  Form of expected instruction(s): `vpadal.s8 d0, d0`
- uint64x2_t vpadalq_u32 (uint64x2_t, uint32x4_t)
  Form of expected instruction(s): `vpadal.u32 q0, q0`
- uint32x4_t vpadalq_u16 (uint32x4_t, uint16x8_t)
  Form of expected instruction(s): `vpadal.u16 q0, q0`
- uint16x8_t vpadalq_u8 (uint16x8_t, uint8x16_t)
  Form of expected instruction(s): `vpadal.u8 q0, q0`
- int64x2_t vpadalq_s32 (int64x2_t, int32x4_t)
  Form of expected instruction(s): `vpadal.s32 q0, q0`
- int32x4_t vpadalq_s16 (int32x4_t, int16x8_t)
  Form of expected instruction(s): `vpadal.s16 q0, q0`
- int16x8_t vpadalq_s8 (int16x8_t, int8x16_t)
  Form of expected instruction(s): `vpadal.s8 q0, q0`

6.55.3.22 Folding maximum

- uint32x2_t vpmax_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vpmax.u32 d0, d0, d0`
- uint16x4_t vpmax_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vpmax.u16 d0, d0, d0`

- uint8x8_t vpmax_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vpmax.u8 d0, d0, d0`
- int32x2_t vpmax_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vpmax.s32 d0, d0, d0`
- int16x4_t vpmax_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vpmax.s16 d0, d0, d0`
- int8x8_t vpmax_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vpmax.s8 d0, d0, d0`
- float32x2_t vpmax_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vpmax.f32 d0, d0, d0`

6.55.3.23 Folding minimum

- uint32x2_t vpmin_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vpmin.u32 d0, d0, d0`
- uint16x4_t vpmin_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vpmin.u16 d0, d0, d0`
- uint8x8_t vpmin_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vpmin.u8 d0, d0, d0`
- int32x2_t vpmin_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vpmin.s32 d0, d0, d0`
- int16x4_t vpmin_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vpmin.s16 d0, d0, d0`
- int8x8_t vpmin_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vpmin.s8 d0, d0, d0`
- float32x2_t vpmin_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vpmin.f32 d0, d0, d0`

6.55.3.24 Reciprocal step

- float32x2_t vrecps_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vrecps.f32 d0, d0, d0`
- float32x4_t vrecpsq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vrecps.f32 q0, q0, q0`
- float32x2_t vrsqrts_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vrsqrts.f32 d0, d0, d0`
- float32x4_t vrsqrtsq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vrsqrts.f32 q0, q0, q0`

6.55.3.25 Vector shift left

- uint32x2_t vshl_u32 (uint32x2_t, int32x2_t)
  Form of expected instruction(s): `vshl.u32 d0, d0, d0`
- uint16x4_t vshl_u16 (uint16x4_t, int16x4_t)
  Form of expected instruction(s): `vshl.u16 d0, d0, d0`
- uint8x8_t vshl_u8 (uint8x8_t, int8x8_t)
  Form of expected instruction(s): `vshl.u8 d0, d0, d0`

- int32x2_t vshl_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vshl.s32 d0, d0, d0`
- int16x4_t vshl_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vshl.s16 d0, d0, d0`
- int8x8_t vshl_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vshl.s8 d0, d0, d0`
- uint64x1_t vshl_u64 (uint64x1_t, int64x1_t)
  Form of expected instruction(s): `vshl.u64 d0, d0, d0`
- int64x1_t vshl_s64 (int64x1_t, int64x1_t)
  Form of expected instruction(s): `vshl.s64 d0, d0, d0`
- uint32x4_t vshlq_u32 (uint32x4_t, int32x4_t)
  Form of expected instruction(s): `vshl.u32 q0, q0, q0`
- uint16x8_t vshlq_u16 (uint16x8_t, int16x8_t)
  Form of expected instruction(s): `vshl.u16 q0, q0, q0`
- uint8x16_t vshlq_u8 (uint8x16_t, int8x16_t)
  Form of expected instruction(s): `vshl.u8 q0, q0, q0`
- int32x4_t vshlq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vshl.s32 q0, q0, q0`
- int16x8_t vshlq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vshl.s16 q0, q0, q0`
- int8x16_t vshlq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vshl.s8 q0, q0, q0`
- uint64x2_t vshlq_u64 (uint64x2_t, int64x2_t)
  Form of expected instruction(s): `vshl.u64 q0, q0, q0`
- int64x2_t vshlq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vshl.s64 q0, q0, q0`
- uint32x2_t vrshl_u32 (uint32x2_t, int32x2_t)
  Form of expected instruction(s): `vrshl.u32 d0, d0, d0`
- uint16x4_t vrshl_u16 (uint16x4_t, int16x4_t)
  Form of expected instruction(s): `vrshl.u16 d0, d0, d0`
- uint8x8_t vrshl_u8 (uint8x8_t, int8x8_t)
  Form of expected instruction(s): `vrshl.u8 d0, d0, d0`
- int32x2_t vrshl_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vrshl.s32 d0, d0, d0`
- int16x4_t vrshl_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vrshl.s16 d0, d0, d0`
- int8x8_t vrshl_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vrshl.s8 d0, d0, d0`
- uint64x1_t vrshl_u64 (uint64x1_t, int64x1_t)
  Form of expected instruction(s): `vrshl.u64 d0, d0, d0`
- int64x1_t vrshl_s64 (int64x1_t, int64x1_t)
  Form of expected instruction(s): `vrshl.s64 d0, d0, d0`

- uint32x4_t vrshlq_u32 (uint32x4_t, int32x4_t)
  Form of expected instruction(s): `vrshl.u32 q0, q0, q0`
- uint16x8_t vrshlq_u16 (uint16x8_t, int16x8_t)
  Form of expected instruction(s): `vrshl.u16 q0, q0, q0`
- uint8x16_t vrshlq_u8 (uint8x16_t, int8x16_t)
  Form of expected instruction(s): `vrshl.u8 q0, q0, q0`
- int32x4_t vrshlq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vrshl.s32 q0, q0, q0`
- int16x8_t vrshlq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vrshl.s16 q0, q0, q0`
- int8x16_t vrshlq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vrshl.s8 q0, q0, q0`
- uint64x2_t vrshlq_u64 (uint64x2_t, int64x2_t)
  Form of expected instruction(s): `vrshl.u64 q0, q0, q0`
- int64x2_t vrshlq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vrshl.s64 q0, q0, q0`
- uint32x2_t vqshl_u32 (uint32x2_t, int32x2_t)
  Form of expected instruction(s): `vqshl.u32 d0, d0, d0`
- uint16x4_t vqshl_u16 (uint16x4_t, int16x4_t)
  Form of expected instruction(s): `vqshl.u16 d0, d0, d0`
- uint8x8_t vqshl_u8 (uint8x8_t, int8x8_t)
  Form of expected instruction(s): `vqshl.u8 d0, d0, d0`
- int32x2_t vqshl_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqshl.s32 d0, d0, d0`
- int16x4_t vqshl_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqshl.s16 d0, d0, d0`
- int8x8_t vqshl_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vqshl.s8 d0, d0, d0`
- uint64x1_t vqshl_u64 (uint64x1_t, int64x1_t)
  Form of expected instruction(s): `vqshl.u64 d0, d0, d0`
- int64x1_t vqshl_s64 (int64x1_t, int64x1_t)
  Form of expected instruction(s): `vqshl.s64 d0, d0, d0`
- uint32x4_t vqshlq_u32 (uint32x4_t, int32x4_t)
  Form of expected instruction(s): `vqshl.u32 q0, q0, q0`
- uint16x8_t vqshlq_u16 (uint16x8_t, int16x8_t)
  Form of expected instruction(s): `vqshl.u16 q0, q0, q0`
- uint8x16_t vqshlq_u8 (uint8x16_t, int8x16_t)
  Form of expected instruction(s): `vqshl.u8 q0, q0, q0`
- int32x4_t vqshlq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vqshl.s32 q0, q0, q0`
- int16x8_t vqshlq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vqshl.s16 q0, q0, q0`

- int8x16_t vqshlq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vqshl.s8 q0, q0, q0`
- uint64x2_t vqshlq_u64 (uint64x2_t, int64x2_t)
  Form of expected instruction(s): `vqshl.u64 q0, q0, q0`
- int64x2_t vqshlq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vqshl.s64 q0, q0, q0`
- uint32x2_t vqrshl_u32 (uint32x2_t, int32x2_t)
  Form of expected instruction(s): `vqrshl.u32 d0, d0, d0`
- uint16x4_t vqrshl_u16 (uint16x4_t, int16x4_t)
  Form of expected instruction(s): `vqrshl.u16 d0, d0, d0`
- uint8x8_t vqrshl_u8 (uint8x8_t, int8x8_t)
  Form of expected instruction(s): `vqrshl.u8 d0, d0, d0`
- int32x2_t vqrshl_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vqrshl.s32 d0, d0, d0`
- int16x4_t vqrshl_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vqrshl.s16 d0, d0, d0`
- int8x8_t vqrshl_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vqrshl.s8 d0, d0, d0`
- uint64x1_t vqrshl_u64 (uint64x1_t, int64x1_t)
  Form of expected instruction(s): `vqrshl.u64 d0, d0, d0`
- int64x1_t vqrshl_s64 (int64x1_t, int64x1_t)
  Form of expected instruction(s): `vqrshl.s64 d0, d0, d0`
- uint32x4_t vqrshlq_u32 (uint32x4_t, int32x4_t)
  Form of expected instruction(s): `vqrshl.u32 q0, q0, q0`
- uint16x8_t vqrshlq_u16 (uint16x8_t, int16x8_t)
  Form of expected instruction(s): `vqrshl.u16 q0, q0, q0`
- uint8x16_t vqrshlq_u8 (uint8x16_t, int8x16_t)
  Form of expected instruction(s): `vqrshl.u8 q0, q0, q0`
- int32x4_t vqrshlq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vqrshl.s32 q0, q0, q0`
- int16x8_t vqrshlq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vqrshl.s16 q0, q0, q0`
- int8x16_t vqrshlq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vqrshl.s8 q0, q0, q0`
- uint64x2_t vqrshlq_u64 (uint64x2_t, int64x2_t)
  Form of expected instruction(s): `vqrshl.u64 q0, q0, q0`
- int64x2_t vqrshlq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vqrshl.s64 q0, q0, q0`

6.55.3.26 Vector shift left by constant

- uint32x2_t vshl_n_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vshl.i32 d0, d0, #0`

- uint16x4_t vshl_n_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vshl.i16 d0, d0, #0`
- uint8x8_t vshl_n_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vshl.i8 d0, d0, #0`
- int32x2_t vshl_n_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vshl.i32 d0, d0, #0`
- int16x4_t vshl_n_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vshl.i16 d0, d0, #0`
- int8x8_t vshl_n_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vshl.i8 d0, d0, #0`
- uint64x1_t vshl_n_u64 (uint64x1_t, const int)
  Form of expected instruction(s): `vshl.i64 d0, d0, #0`
- int64x1_t vshl_n_s64 (int64x1_t, const int)
  Form of expected instruction(s): `vshl.i64 d0, d0, #0`
- uint32x4_t vshlq_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vshl.i32 q0, q0, #0`
- uint16x8_t vshlq_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vshl.i16 q0, q0, #0`
- uint8x16_t vshlq_n_u8 (uint8x16_t, const int)
  Form of expected instruction(s): `vshl.i8 q0, q0, #0`
- int32x4_t vshlq_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vshl.i32 q0, q0, #0`
- int16x8_t vshlq_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vshl.i16 q0, q0, #0`
- int8x16_t vshlq_n_s8 (int8x16_t, const int)
  Form of expected instruction(s): `vshl.i8 q0, q0, #0`
- uint64x2_t vshlq_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vshl.i64 q0, q0, #0`
- int64x2_t vshlq_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vshl.i64 q0, q0, #0`
- uint32x2_t vqshl_n_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vqshl.u32 d0, d0, #0`
- uint16x4_t vqshl_n_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vqshl.u16 d0, d0, #0`
- uint8x8_t vqshl_n_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vqshl.u8 d0, d0, #0`
- int32x2_t vqshl_n_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vqshl.s32 d0, d0, #0`
- int16x4_t vqshl_n_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vqshl.s16 d0, d0, #0`
- int8x8_t vqshl_n_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vqshl.s8 d0, d0, #0`

- uint64x1_t vqshl_n_u64 (uint64x1_t, const int)
  Form of expected instruction(s): `vqshl.u64 d0, d0, #0`
- int64x1_t vqshl_n_s64 (int64x1_t, const int)
  Form of expected instruction(s): `vqshl.s64 d0, d0, #0`
- uint32x4_t vqshlq_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vqshl.u32 q0, q0, #0`
- uint16x8_t vqshlq_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vqshl.u16 q0, q0, #0`
- uint8x16_t vqshlq_n_u8 (uint8x16_t, const int)
  Form of expected instruction(s): `vqshl.u8 q0, q0, #0`
- int32x4_t vqshlq_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vqshl.s32 q0, q0, #0`
- int16x8_t vqshlq_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vqshl.s16 q0, q0, #0`
- int8x16_t vqshlq_n_s8 (int8x16_t, const int)
  Form of expected instruction(s): `vqshl.s8 q0, q0, #0`
- uint64x2_t vqshlq_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vqshl.u64 q0, q0, #0`
- int64x2_t vqshlq_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vqshl.s64 q0, q0, #0`
- uint64x1_t vqshlu_n_s64 (int64x1_t, const int)
  Form of expected instruction(s): `vqshlu.s64 d0, d0, #0`
- uint32x2_t vqshlu_n_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vqshlu.s32 d0, d0, #0`
- uint16x4_t vqshlu_n_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vqshlu.s16 d0, d0, #0`
- uint8x8_t vqshlu_n_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vqshlu.s8 d0, d0, #0`
- uint64x2_t vqshluq_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vqshlu.s64 q0, q0, #0`
- uint32x4_t vqshluq_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vqshlu.s32 q0, q0, #0`
- uint16x8_t vqshluq_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vqshlu.s16 q0, q0, #0`
- uint8x16_t vqshluq_n_s8 (int8x16_t, const int)
  Form of expected instruction(s): `vqshlu.s8 q0, q0, #0`
- uint64x2_t vshll_n_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vshll.u32 q0, d0, #0`
- uint32x4_t vshll_n_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vshll.u16 q0, d0, #0`
- uint16x8_t vshll_n_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vshll.u8 q0, d0, #0`

- int64x2_t vshll_n_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vshll.s32 q0, d0, #0`
- int32x4_t vshll_n_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vshll.s16 q0, d0, #0`
- int16x8_t vshll_n_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vshll.s8 q0, d0, #0`

6.55.3.27 Vector shift right by constant

- uint32x2_t vshr_n_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vshr.u32 d0, d0, #0`
- uint16x4_t vshr_n_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vshr.u16 d0, d0, #0`
- uint8x8_t vshr_n_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vshr.u8 d0, d0, #0`
- int32x2_t vshr_n_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vshr.s32 d0, d0, #0`
- int16x4_t vshr_n_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vshr.s16 d0, d0, #0`
- int8x8_t vshr_n_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vshr.s8 d0, d0, #0`
- uint64x1_t vshr_n_u64 (uint64x1_t, const int)
  Form of expected instruction(s): `vshr.u64 d0, d0, #0`
- int64x1_t vshr_n_s64 (int64x1_t, const int)
  Form of expected instruction(s): `vshr.s64 d0, d0, #0`
- uint32x4_t vshrq_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vshr.u32 q0, q0, #0`
- uint16x8_t vshrq_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vshr.u16 q0, q0, #0`
- uint8x16_t vshrq_n_u8 (uint8x16_t, const int)
  Form of expected instruction(s): `vshr.u8 q0, q0, #0`
- int32x4_t vshrq_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vshr.s32 q0, q0, #0`
- int16x8_t vshrq_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vshr.s16 q0, q0, #0`
- int8x16_t vshrq_n_s8 (int8x16_t, const int)
  Form of expected instruction(s): `vshr.s8 q0, q0, #0`
- uint64x2_t vshrq_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vshr.u64 q0, q0, #0`
- int64x2_t vshrq_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vshr.s64 q0, q0, #0`
- uint32x2_t vrshr_n_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vrshr.u32 d0, d0, #0`

- uint16x4_t vrshr_n_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vrshr.u16 d0, d0, #0`
- uint8x8_t vrshr_n_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vrshr.u8 d0, d0, #0`
- int32x2_t vrshr_n_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vrshr.s32 d0, d0, #0`
- int16x4_t vrshr_n_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vrshr.s16 d0, d0, #0`
- int8x8_t vrshr_n_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vrshr.s8 d0, d0, #0`
- uint64x1_t vrshr_n_u64 (uint64x1_t, const int)
  Form of expected instruction(s): `vrshr.u64 d0, d0, #0`
- int64x1_t vrshr_n_s64 (int64x1_t, const int)
  Form of expected instruction(s): `vrshr.s64 d0, d0, #0`
- uint32x4_t vrshrq_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vrshr.u32 q0, q0, #0`
- uint16x8_t vrshrq_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vrshr.u16 q0, q0, #0`
- uint8x16_t vrshrq_n_u8 (uint8x16_t, const int)
  Form of expected instruction(s): `vrshr.u8 q0, q0, #0`
- int32x4_t vrshrq_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vrshr.s32 q0, q0, #0`
- int16x8_t vrshrq_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vrshr.s16 q0, q0, #0`
- int8x16_t vrshrq_n_s8 (int8x16_t, const int)
  Form of expected instruction(s): `vrshr.s8 q0, q0, #0`
- uint64x2_t vrshrq_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vrshr.u64 q0, q0, #0`
- int64x2_t vrshrq_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vrshr.s64 q0, q0, #0`
- uint32x2_t vshrn_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vshrn.i64 d0, q0, #0`
- uint16x4_t vshrn_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vshrn.i32 d0, q0, #0`
- uint8x8_t vshrn_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vshrn.i16 d0, q0, #0`
- int32x2_t vshrn_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vshrn.i64 d0, q0, #0`
- int16x4_t vshrn_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vshrn.i32 d0, q0, #0`
- int8x8_t vshrn_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vshrn.i16 d0, q0, #0`

- uint32x2_t vrshrn_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vrshrn.i64 d0, q0, #0`
- uint16x4_t vrshrn_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vrshrn.i32 d0, q0, #0`
- uint8x8_t vrshrn_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vrshrn.i16 d0, q0, #0`
- int32x2_t vrshrn_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vrshrn.i64 d0, q0, #0`
- int16x4_t vrshrn_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vrshrn.i32 d0, q0, #0`
- int8x8_t vrshrn_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vrshrn.i16 d0, q0, #0`
- uint32x2_t vqshrn_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vqshrn.u64 d0, q0, #0`
- uint16x4_t vqshrn_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vqshrn.u32 d0, q0, #0`
- uint8x8_t vqshrn_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vqshrn.u16 d0, q0, #0`
- int32x2_t vqshrn_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vqshrn.s64 d0, q0, #0`
- int16x4_t vqshrn_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vqshrn.s32 d0, q0, #0`
- int8x8_t vqshrn_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vqshrn.s16 d0, q0, #0`
- uint32x2_t vqrshrn_n_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vqrshrn.u64 d0, q0, #0`
- uint16x4_t vqrshrn_n_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vqrshrn.u32 d0, q0, #0`
- uint8x8_t vqrshrn_n_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vqrshrn.u16 d0, q0, #0`
- int32x2_t vqrshrn_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vqrshrn.s64 d0, q0, #0`
- int16x4_t vqrshrn_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vqrshrn.s32 d0, q0, #0`
- int8x8_t vqrshrn_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vqrshrn.s16 d0, q0, #0`
- uint32x2_t vqshrun_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vqshrun.s64 d0, q0, #0`
- uint16x4_t vqshrun_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vqshrun.s32 d0, q0, #0`
- uint8x8_t vqshrun_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vqshrun.s16 d0, q0, #0`

- uint32x2_t vqrshrun_n_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vqrshrun.s64 d0, q0, #0`
- uint16x4_t vqrshrun_n_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vqrshrun.s32 d0, q0, #0`
- uint8x8_t vqrshrun_n_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vqrshrun.s16 d0, q0, #0`

6.55.3.28 Vector shift right by constant and accumulate

- uint32x2_t vsra_n_u32 (uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vsra.u32 d0, d0, #0`
- uint16x4_t vsra_n_u16 (uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vsra.u16 d0, d0, #0`
- uint8x8_t vsra_n_u8 (uint8x8_t, uint8x8_t, const int)
  Form of expected instruction(s): `vsra.u8 d0, d0, #0`
- int32x2_t vsra_n_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vsra.s32 d0, d0, #0`
- int16x4_t vsra_n_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vsra.s16 d0, d0, #0`
- int8x8_t vsra_n_s8 (int8x8_t, int8x8_t, const int)
  Form of expected instruction(s): `vsra.s8 d0, d0, #0`
- uint64x1_t vsra_n_u64 (uint64x1_t, uint64x1_t, const int)
  Form of expected instruction(s): `vsra.u64 d0, d0, #0`
- int64x1_t vsra_n_s64 (int64x1_t, int64x1_t, const int)
  Form of expected instruction(s): `vsra.s64 d0, d0, #0`
- uint32x4_t vsraq_n_u32 (uint32x4_t, uint32x4_t, const int)
  Form of expected instruction(s): `vsra.u32 q0, q0, #0`
- uint16x8_t vsraq_n_u16 (uint16x8_t, uint16x8_t, const int)
  Form of expected instruction(s): `vsra.u16 q0, q0, #0`
- uint8x16_t vsraq_n_u8 (uint8x16_t, uint8x16_t, const int)
  Form of expected instruction(s): `vsra.u8 q0, q0, #0`
- int32x4_t vsraq_n_s32 (int32x4_t, int32x4_t, const int)
  Form of expected instruction(s): `vsra.s32 q0, q0, #0`
- int16x8_t vsraq_n_s16 (int16x8_t, int16x8_t, const int)
  Form of expected instruction(s): `vsra.s16 q0, q0, #0`
- int8x16_t vsraq_n_s8 (int8x16_t, int8x16_t, const int)
  Form of expected instruction(s): `vsra.s8 q0, q0, #0`
- uint64x2_t vsraq_n_u64 (uint64x2_t, uint64x2_t, const int)
  Form of expected instruction(s): `vsra.u64 q0, q0, #0`
- int64x2_t vsraq_n_s64 (int64x2_t, int64x2_t, const int)
  Form of expected instruction(s): `vsra.s64 q0, q0, #0`
- uint32x2_t vrsra_n_u32 (uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vrsra.u32 d0, d0, #0`

- uint16x4_t vrsra_n_u16 (uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vrsra.u16 d0, d0, #0`
- uint8x8_t vrsra_n_u8 (uint8x8_t, uint8x8_t, const int)
  Form of expected instruction(s): `vrsra.u8 d0, d0, #0`
- int32x2_t vrsra_n_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vrsra.s32 d0, d0, #0`
- int16x4_t vrsra_n_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vrsra.s16 d0, d0, #0`
- int8x8_t vrsra_n_s8 (int8x8_t, int8x8_t, const int)
  Form of expected instruction(s): `vrsra.s8 d0, d0, #0`
- uint64x1_t vrsra_n_u64 (uint64x1_t, uint64x1_t, const int)
  Form of expected instruction(s): `vrsra.u64 d0, d0, #0`
- int64x1_t vrsra_n_s64 (int64x1_t, int64x1_t, const int)
  Form of expected instruction(s): `vrsra.s64 d0, d0, #0`
- uint32x4_t vrsraq_n_u32 (uint32x4_t, uint32x4_t, const int)
  Form of expected instruction(s): `vrsra.u32 q0, q0, #0`
- uint16x8_t vrsraq_n_u16 (uint16x8_t, uint16x8_t, const int)
  Form of expected instruction(s): `vrsra.u16 q0, q0, #0`
- uint8x16_t vrsraq_n_u8 (uint8x16_t, uint8x16_t, const int)
  Form of expected instruction(s): `vrsra.u8 q0, q0, #0`
- int32x4_t vrsraq_n_s32 (int32x4_t, int32x4_t, const int)
  Form of expected instruction(s): `vrsra.s32 q0, q0, #0`
- int16x8_t vrsraq_n_s16 (int16x8_t, int16x8_t, const int)
  Form of expected instruction(s): `vrsra.s16 q0, q0, #0`
- int8x16_t vrsraq_n_s8 (int8x16_t, int8x16_t, const int)
  Form of expected instruction(s): `vrsra.s8 q0, q0, #0`
- uint64x2_t vrsraq_n_u64 (uint64x2_t, uint64x2_t, const int)
  Form of expected instruction(s): `vrsra.u64 q0, q0, #0`
- int64x2_t vrsraq_n_s64 (int64x2_t, int64x2_t, const int)
  Form of expected instruction(s): `vrsra.s64 q0, q0, #0`

6.55.3.29 Vector shift right and insert

- uint32x2_t vsri_n_u32 (uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vsri.32 d0, d0, #0`
- uint16x4_t vsri_n_u16 (uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vsri.16 d0, d0, #0`
- uint8x8_t vsri_n_u8 (uint8x8_t, uint8x8_t, const int)
  Form of expected instruction(s): `vsri.8 d0, d0, #0`
- int32x2_t vsri_n_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vsri.32 d0, d0, #0`
- int16x4_t vsri_n_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vsri.16 d0, d0, #0`

- int8x8_t vsri_n_s8 (int8x8_t, int8x8_t, const int)
  Form of expected instruction(s): `vsri.8 d0, d0, #0`
- uint64x1_t vsri_n_u64 (uint64x1_t, uint64x1_t, const int)
  Form of expected instruction(s): `vsri.64 d0, d0, #0`
- int64x1_t vsri_n_s64 (int64x1_t, int64x1_t, const int)
  Form of expected instruction(s): `vsri.64 d0, d0, #0`
- poly16x4_t vsri_n_p16 (poly16x4_t, poly16x4_t, const int)
  Form of expected instruction(s): `vsri.16 d0, d0, #0`
- poly8x8_t vsri_n_p8 (poly8x8_t, poly8x8_t, const int)
  Form of expected instruction(s): `vsri.8 d0, d0, #0`
- uint32x4_t vsriq_n_u32 (uint32x4_t, uint32x4_t, const int)
  Form of expected instruction(s): `vsri.32 q0, q0, #0`
- uint16x8_t vsriq_n_u16 (uint16x8_t, uint16x8_t, const int)
  Form of expected instruction(s): `vsri.16 q0, q0, #0`
- uint8x16_t vsriq_n_u8 (uint8x16_t, uint8x16_t, const int)
  Form of expected instruction(s): `vsri.8 q0, q0, #0`
- int32x4_t vsriq_n_s32 (int32x4_t, int32x4_t, const int)
  Form of expected instruction(s): `vsri.32 q0, q0, #0`
- int16x8_t vsriq_n_s16 (int16x8_t, int16x8_t, const int)
  Form of expected instruction(s): `vsri.16 q0, q0, #0`
- int8x16_t vsriq_n_s8 (int8x16_t, int8x16_t, const int)
  Form of expected instruction(s): `vsri.8 q0, q0, #0`
- uint64x2_t vsriq_n_u64 (uint64x2_t, uint64x2_t, const int)
  Form of expected instruction(s): `vsri.64 q0, q0, #0`
- int64x2_t vsriq_n_s64 (int64x2_t, int64x2_t, const int)
  Form of expected instruction(s): `vsri.64 q0, q0, #0`
- poly16x8_t vsriq_n_p16 (poly16x8_t, poly16x8_t, const int)
  Form of expected instruction(s): `vsri.16 q0, q0, #0`
- poly8x16_t vsriq_n_p8 (poly8x16_t, poly8x16_t, const int)
  Form of expected instruction(s): `vsri.8 q0, q0, #0`

6.55.3.30 Vector shift left and insert

- uint32x2_t vsli_n_u32 (uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vsli.32 d0, d0, #0`
- uint16x4_t vsli_n_u16 (uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vsli.16 d0, d0, #0`
- uint8x8_t vsli_n_u8 (uint8x8_t, uint8x8_t, const int)
  Form of expected instruction(s): `vsli.8 d0, d0, #0`
- int32x2_t vsli_n_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vsli.32 d0, d0, #0`
- int16x4_t vsli_n_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vsli.16 d0, d0, #0`

- int8x8_t vsli_n_s8 (int8x8_t, int8x8_t, const int)
  Form of expected instruction(s): `vsli.8 d0, d0, #0`
- uint64x1_t vsli_n_u64 (uint64x1_t, uint64x1_t, const int)
  Form of expected instruction(s): `vsli.64 d0, d0, #0`
- int64x1_t vsli_n_s64 (int64x1_t, int64x1_t, const int)
  Form of expected instruction(s): `vsli.64 d0, d0, #0`
- poly16x4_t vsli_n_p16 (poly16x4_t, poly16x4_t, const int)
  Form of expected instruction(s): `vsli.16 d0, d0, #0`
- poly8x8_t vsli_n_p8 (poly8x8_t, poly8x8_t, const int)
  Form of expected instruction(s): `vsli.8 d0, d0, #0`
- uint32x4_t vsliq_n_u32 (uint32x4_t, uint32x4_t, const int)
  Form of expected instruction(s): `vsli.32 q0, q0, #0`
- uint16x8_t vsliq_n_u16 (uint16x8_t, uint16x8_t, const int)
  Form of expected instruction(s): `vsli.16 q0, q0, #0`
- uint8x16_t vsliq_n_u8 (uint8x16_t, uint8x16_t, const int)
  Form of expected instruction(s): `vsli.8 q0, q0, #0`
- int32x4_t vsliq_n_s32 (int32x4_t, int32x4_t, const int)
  Form of expected instruction(s): `vsli.32 q0, q0, #0`
- int16x8_t vsliq_n_s16 (int16x8_t, int16x8_t, const int)
  Form of expected instruction(s): `vsli.16 q0, q0, #0`
- int8x16_t vsliq_n_s8 (int8x16_t, int8x16_t, const int)
  Form of expected instruction(s): `vsli.8 q0, q0, #0`
- uint64x2_t vsliq_n_u64 (uint64x2_t, uint64x2_t, const int)
  Form of expected instruction(s): `vsli.64 q0, q0, #0`
- int64x2_t vsliq_n_s64 (int64x2_t, int64x2_t, const int)
  Form of expected instruction(s): `vsli.64 q0, q0, #0`
- poly16x8_t vsliq_n_p16 (poly16x8_t, poly16x8_t, const int)
  Form of expected instruction(s): `vsli.16 q0, q0, #0`
- poly8x16_t vsliq_n_p8 (poly8x16_t, poly8x16_t, const int)
  Form of expected instruction(s): `vsli.8 q0, q0, #0`

6.55.3.31 Absolute value

- float32x2_t vabs_f32 (float32x2_t)
  Form of expected instruction(s): `vabs.f32 d0, d0`
- int32x2_t vabs_s32 (int32x2_t)
  Form of expected instruction(s): `vabs.s32 d0, d0`
- int16x4_t vabs_s16 (int16x4_t)
  Form of expected instruction(s): `vabs.s16 d0, d0`
- int8x8_t vabs_s8 (int8x8_t)
  Form of expected instruction(s): `vabs.s8 d0, d0`
- float32x4_t vabsq_f32 (float32x4_t)
  Form of expected instruction(s): `vabs.f32 q0, q0`

- int32x4_t vabsq_s32 (int32x4_t)
  Form of expected instruction(s): `vabs.s32` *q0, q0*
- int16x8_t vabsq_s16 (int16x8_t)
  Form of expected instruction(s): `vabs.s16` *q0, q0*
- int8x16_t vabsq_s8 (int8x16_t)
  Form of expected instruction(s): `vabs.s8` *q0, q0*
- int32x2_t vqabs_s32 (int32x2_t)
  Form of expected instruction(s): `vqabs.s32` *d0, d0*
- int16x4_t vqabs_s16 (int16x4_t)
  Form of expected instruction(s): `vqabs.s16` *d0, d0*
- int8x8_t vqabs_s8 (int8x8_t)
  Form of expected instruction(s): `vqabs.s8` *d0, d0*
- int32x4_t vqabsq_s32 (int32x4_t)
  Form of expected instruction(s): `vqabs.s32` *q0, q0*
- int16x8_t vqabsq_s16 (int16x8_t)
  Form of expected instruction(s): `vqabs.s16` *q0, q0*
- int8x16_t vqabsq_s8 (int8x16_t)
  Form of expected instruction(s): `vqabs.s8` *q0, q0*

6.55.3.32 Negation

- float32x2_t vneg_f32 (float32x2_t)
  Form of expected instruction(s): `vneg.f32` *d0, d0*
- int32x2_t vneg_s32 (int32x2_t)
  Form of expected instruction(s): `vneg.s32` *d0, d0*
- int16x4_t vneg_s16 (int16x4_t)
  Form of expected instruction(s): `vneg.s16` *d0, d0*
- int8x8_t vneg_s8 (int8x8_t)
  Form of expected instruction(s): `vneg.s8` *d0, d0*
- float32x4_t vnegq_f32 (float32x4_t)
  Form of expected instruction(s): `vneg.f32` *q0, q0*
- int32x4_t vnegq_s32 (int32x4_t)
  Form of expected instruction(s): `vneg.s32` *q0, q0*
- int16x8_t vnegq_s16 (int16x8_t)
  Form of expected instruction(s): `vneg.s16` *q0, q0*
- int8x16_t vnegq_s8 (int8x16_t)
  Form of expected instruction(s): `vneg.s8` *q0, q0*
- int32x2_t vqneg_s32 (int32x2_t)
  Form of expected instruction(s): `vqneg.s32` *d0, d0*
- int16x4_t vqneg_s16 (int16x4_t)
  Form of expected instruction(s): `vqneg.s16` *d0, d0*
- int8x8_t vqneg_s8 (int8x8_t)
  Form of expected instruction(s): `vqneg.s8` *d0, d0*

- int32x4_t vqnegq_s32 (int32x4_t)
  Form of expected instruction(s): `vqneg.s32 q0, q0`
- int16x8_t vqnegq_s16 (int16x8_t)
  Form of expected instruction(s): `vqneg.s16 q0, q0`
- int8x16_t vqnegq_s8 (int8x16_t)
  Form of expected instruction(s): `vqneg.s8 q0, q0`

6.55.3.33 Bitwise not

- uint32x2_t vmvn_u32 (uint32x2_t)
  Form of expected instruction(s): `vmvn d0, d0`
- uint16x4_t vmvn_u16 (uint16x4_t)
  Form of expected instruction(s): `vmvn d0, d0`
- uint8x8_t vmvn_u8 (uint8x8_t)
  Form of expected instruction(s): `vmvn d0, d0`
- int32x2_t vmvn_s32 (int32x2_t)
  Form of expected instruction(s): `vmvn d0, d0`
- int16x4_t vmvn_s16 (int16x4_t)
  Form of expected instruction(s): `vmvn d0, d0`
- int8x8_t vmvn_s8 (int8x8_t)
  Form of expected instruction(s): `vmvn d0, d0`
- poly8x8_t vmvn_p8 (poly8x8_t)
  Form of expected instruction(s): `vmvn d0, d0`
- uint32x4_t vmvnq_u32 (uint32x4_t)
  Form of expected instruction(s): `vmvn q0, q0`
- uint16x8_t vmvnq_u16 (uint16x8_t)
  Form of expected instruction(s): `vmvn q0, q0`
- uint8x16_t vmvnq_u8 (uint8x16_t)
  Form of expected instruction(s): `vmvn q0, q0`
- int32x4_t vmvnq_s32 (int32x4_t)
  Form of expected instruction(s): `vmvn q0, q0`
- int16x8_t vmvnq_s16 (int16x8_t)
  Form of expected instruction(s): `vmvn q0, q0`
- int8x16_t vmvnq_s8 (int8x16_t)
  Form of expected instruction(s): `vmvn q0, q0`
- poly8x16_t vmvnq_p8 (poly8x16_t)
  Form of expected instruction(s): `vmvn q0, q0`

6.55.3.34 Count leading sign bits

- int32x2_t vcls_s32 (int32x2_t)
  Form of expected instruction(s): `vcls.s32 d0, d0`
- int16x4_t vcls_s16 (int16x4_t)
  Form of expected instruction(s): `vcls.s16 d0, d0`

- int8x8_t vcls_s8 (int8x8_t)
  Form of expected instruction(s): `vcls.s8 d0, d0`
- int32x4_t vclsq_s32 (int32x4_t)
  Form of expected instruction(s): `vcls.s32 q0, q0`
- int16x8_t vclsq_s16 (int16x8_t)
  Form of expected instruction(s): `vcls.s16 q0, q0`
- int8x16_t vclsq_s8 (int8x16_t)
  Form of expected instruction(s): `vcls.s8 q0, q0`

6.55.3.35 Count leading zeros

- uint32x2_t vclz_u32 (uint32x2_t)
  Form of expected instruction(s): `vclz.i32 d0, d0`
- uint16x4_t vclz_u16 (uint16x4_t)
  Form of expected instruction(s): `vclz.i16 d0, d0`
- uint8x8_t vclz_u8 (uint8x8_t)
  Form of expected instruction(s): `vclz.i8 d0, d0`
- int32x2_t vclz_s32 (int32x2_t)
  Form of expected instruction(s): `vclz.i32 d0, d0`
- int16x4_t vclz_s16 (int16x4_t)
  Form of expected instruction(s): `vclz.i16 d0, d0`
- int8x8_t vclz_s8 (int8x8_t)
  Form of expected instruction(s): `vclz.i8 d0, d0`
- uint32x4_t vclzq_u32 (uint32x4_t)
  Form of expected instruction(s): `vclz.i32 q0, q0`
- uint16x8_t vclzq_u16 (uint16x8_t)
  Form of expected instruction(s): `vclz.i16 q0, q0`
- uint8x16_t vclzq_u8 (uint8x16_t)
  Form of expected instruction(s): `vclz.i8 q0, q0`
- int32x4_t vclzq_s32 (int32x4_t)
  Form of expected instruction(s): `vclz.i32 q0, q0`
- int16x8_t vclzq_s16 (int16x8_t)
  Form of expected instruction(s): `vclz.i16 q0, q0`
- int8x16_t vclzq_s8 (int8x16_t)
  Form of expected instruction(s): `vclz.i8 q0, q0`

6.55.3.36 Count number of set bits

- uint8x8_t vcnt_u8 (uint8x8_t)
  Form of expected instruction(s): `vcnt.8 d0, d0`
- int8x8_t vcnt_s8 (int8x8_t)
  Form of expected instruction(s): `vcnt.8 d0, d0`
- poly8x8_t vcnt_p8 (poly8x8_t)
  Form of expected instruction(s): `vcnt.8 d0, d0`

- uint8x16_t vcntq_u8 (uint8x16_t)
  Form of expected instruction(s): `vcnt.8 q0, q0`
- int8x16_t vcntq_s8 (int8x16_t)
  Form of expected instruction(s): `vcnt.8 q0, q0`
- poly8x16_t vcntq_p8 (poly8x16_t)
  Form of expected instruction(s): `vcnt.8 q0, q0`

6.55.3.37 Reciprocal estimate

- float32x2_t vrecpe_f32 (float32x2_t)
  Form of expected instruction(s): `vrecpe.f32 d0, d0`
- uint32x2_t vrecpe_u32 (uint32x2_t)
  Form of expected instruction(s): `vrecpe.u32 d0, d0`
- float32x4_t vrecpeq_f32 (float32x4_t)
  Form of expected instruction(s): `vrecpe.f32 q0, q0`
- uint32x4_t vrecpeq_u32 (uint32x4_t)
  Form of expected instruction(s): `vrecpe.u32 q0, q0`

6.55.3.38 Reciprocal square-root estimate

- float32x2_t vrsqrte_f32 (float32x2_t)
  Form of expected instruction(s): `vrsqrte.f32 d0, d0`
- uint32x2_t vrsqrte_u32 (uint32x2_t)
  Form of expected instruction(s): `vrsqrte.u32 d0, d0`
- float32x4_t vrsqrteq_f32 (float32x4_t)
  Form of expected instruction(s): `vrsqrte.f32 q0, q0`
- uint32x4_t vrsqrteq_u32 (uint32x4_t)
  Form of expected instruction(s): `vrsqrte.u32 q0, q0`

6.55.3.39 Get lanes from a vector

- uint32_t vget_lane_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vmov.32 r0, d0[0]`
- uint16_t vget_lane_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vmov.u16 r0, d0[0]`
- uint8_t vget_lane_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vmov.u8 r0, d0[0]`
- int32_t vget_lane_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vmov.32 r0, d0[0]`
- int16_t vget_lane_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vmov.s16 r0, d0[0]`
- int8_t vget_lane_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vmov.s8 r0, d0[0]`
- float32_t vget_lane_f32 (float32x2_t, const int)
  Form of expected instruction(s): `vmov.32 r0, d0[0]`
- poly16_t vget_lane_p16 (poly16x4_t, const int)
  Form of expected instruction(s): `vmov.u16 r0, d0[0]`

- poly8_t vget_lane_p8 (poly8x8_t, const int)
  Form of expected instruction(s): `vmov.u8 r0, d0[0]`
- uint64_t vget_lane_u64 (uint64x1_t, const int)
- int64_t vget_lane_s64 (int64x1_t, const int)
- uint32_t vgetq_lane_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vmov.32 r0, d0[0]`
- uint16_t vgetq_lane_u16 (uint16x8_t, const int)
  Form of expected instruction(s): `vmov.u16 r0, d0[0]`
- uint8_t vgetq_lane_u8 (uint8x16_t, const int)
  Form of expected instruction(s): `vmov.u8 r0, d0[0]`
- int32_t vgetq_lane_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vmov.32 r0, d0[0]`
- int16_t vgetq_lane_s16 (int16x8_t, const int)
  Form of expected instruction(s): `vmov.s16 r0, d0[0]`
- int8_t vgetq_lane_s8 (int8x16_t, const int)
  Form of expected instruction(s): `vmov.s8 r0, d0[0]`
- float32_t vgetq_lane_f32 (float32x4_t, const int)
  Form of expected instruction(s): `vmov.32 r0, d0[0]`
- poly16_t vgetq_lane_p16 (poly16x8_t, const int)
  Form of expected instruction(s): `vmov.u16 r0, d0[0]`
- poly8_t vgetq_lane_p8 (poly8x16_t, const int)
  Form of expected instruction(s): `vmov.u8 r0, d0[0]`
- uint64_t vgetq_lane_u64 (uint64x2_t, const int)
  Form of expected instruction(s): `vmov r0, r0, d0`
- int64_t vgetq_lane_s64 (int64x2_t, const int)
  Form of expected instruction(s): `vmov r0, r0, d0`

6.55.3.40 Set lanes in a vector

- uint32x2_t vset_lane_u32 (uint32_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmov.32 d0[0], r0`
- uint16x4_t vset_lane_u16 (uint16_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmov.16 d0[0], r0`
- uint8x8_t vset_lane_u8 (uint8_t, uint8x8_t, const int)
  Form of expected instruction(s): `vmov.8 d0[0], r0`
- int32x2_t vset_lane_s32 (int32_t, int32x2_t, const int)
  Form of expected instruction(s): `vmov.32 d0[0], r0`
- int16x4_t vset_lane_s16 (int16_t, int16x4_t, const int)
  Form of expected instruction(s): `vmov.16 d0[0], r0`
- int8x8_t vset_lane_s8 (int8_t, int8x8_t, const int)
  Form of expected instruction(s): `vmov.8 d0[0], r0`
- float32x2_t vset_lane_f32 (float32_t, float32x2_t, const int)
  Form of expected instruction(s): `vmov.32 d0[0], r0`

- poly16x4_t vset_lane_p16 (poly16_t, poly16x4_t, const int)
  Form of expected instruction(s): `vmov.16 d0[0], r0`
- poly8x8_t vset_lane_p8 (poly8_t, poly8x8_t, const int)
  Form of expected instruction(s): `vmov.8 d0[0], r0`
- uint64x1_t vset_lane_u64 (uint64_t, uint64x1_t, const int)
- int64x1_t vset_lane_s64 (int64_t, int64x1_t, const int)
- uint32x4_t vsetq_lane_u32 (uint32_t, uint32x4_t, const int)
  Form of expected instruction(s): `vmov.32 d0[0], r0`
- uint16x8_t vsetq_lane_u16 (uint16_t, uint16x8_t, const int)
  Form of expected instruction(s): `vmov.16 d0[0], r0`
- uint8x16_t vsetq_lane_u8 (uint8_t, uint8x16_t, const int)
  Form of expected instruction(s): `vmov.8 d0[0], r0`
- int32x4_t vsetq_lane_s32 (int32_t, int32x4_t, const int)
  Form of expected instruction(s): `vmov.32 d0[0], r0`
- int16x8_t vsetq_lane_s16 (int16_t, int16x8_t, const int)
  Form of expected instruction(s): `vmov.16 d0[0], r0`
- int8x16_t vsetq_lane_s8 (int8_t, int8x16_t, const int)
  Form of expected instruction(s): `vmov.8 d0[0], r0`
- float32x4_t vsetq_lane_f32 (float32_t, float32x4_t, const int)
  Form of expected instruction(s): `vmov.32 d0[0], r0`
- poly16x8_t vsetq_lane_p16 (poly16_t, poly16x8_t, const int)
  Form of expected instruction(s): `vmov.16 d0[0], r0`
- poly8x16_t vsetq_lane_p8 (poly8_t, poly8x16_t, const int)
  Form of expected instruction(s): `vmov.8 d0[0], r0`
- uint64x2_t vsetq_lane_u64 (uint64_t, uint64x2_t, const int)
  Form of expected instruction(s): `vmov d0, r0, r0`
- int64x2_t vsetq_lane_s64 (int64_t, int64x2_t, const int)
  Form of expected instruction(s): `vmov d0, r0, r0`

6.55.3.41 Create vector from literal bit pattern

- uint32x2_t vcreate_u32 (uint64_t)
- uint16x4_t vcreate_u16 (uint64_t)
- uint8x8_t vcreate_u8 (uint64_t)
- int32x2_t vcreate_s32 (uint64_t)
- int16x4_t vcreate_s16 (uint64_t)
- int8x8_t vcreate_s8 (uint64_t)
- uint64x1_t vcreate_u64 (uint64_t)
- int64x1_t vcreate_s64 (uint64_t)
- float32x2_t vcreate_f32 (uint64_t)
- poly16x4_t vcreate_p16 (uint64_t)
- poly8x8_t vcreate_p8 (uint64_t)

6.55.3.42 Set all lanes to the same value

- uint32x2_t vdup_n_u32 (uint32_t)
  Form of expected instruction(s): `vdup.32 d0, r0`
- uint16x4_t vdup_n_u16 (uint16_t)
  Form of expected instruction(s): `vdup.16 d0, r0`
- uint8x8_t vdup_n_u8 (uint8_t)
  Form of expected instruction(s): `vdup.8 d0, r0`
- int32x2_t vdup_n_s32 (int32_t)
  Form of expected instruction(s): `vdup.32 d0, r0`
- int16x4_t vdup_n_s16 (int16_t)
  Form of expected instruction(s): `vdup.16 d0, r0`
- int8x8_t vdup_n_s8 (int8_t)
  Form of expected instruction(s): `vdup.8 d0, r0`
- float32x2_t vdup_n_f32 (float32_t)
  Form of expected instruction(s): `vdup.32 d0, r0`
- poly16x4_t vdup_n_p16 (poly16_t)
  Form of expected instruction(s): `vdup.16 d0, r0`
- poly8x8_t vdup_n_p8 (poly8_t)
  Form of expected instruction(s): `vdup.8 d0, r0`
- uint64x1_t vdup_n_u64 (uint64_t)
- int64x1_t vdup_n_s64 (int64_t)
- uint32x4_t vdupq_n_u32 (uint32_t)
  Form of expected instruction(s): `vdup.32 q0, r0`
- uint16x8_t vdupq_n_u16 (uint16_t)
  Form of expected instruction(s): `vdup.16 q0, r0`
- uint8x16_t vdupq_n_u8 (uint8_t)
  Form of expected instruction(s): `vdup.8 q0, r0`
- int32x4_t vdupq_n_s32 (int32_t)
  Form of expected instruction(s): `vdup.32 q0, r0`
- int16x8_t vdupq_n_s16 (int16_t)
  Form of expected instruction(s): `vdup.16 q0, r0`
- int8x16_t vdupq_n_s8 (int8_t)
  Form of expected instruction(s): `vdup.8 q0, r0`
- float32x4_t vdupq_n_f32 (float32_t)
  Form of expected instruction(s): `vdup.32 q0, r0`
- poly16x8_t vdupq_n_p16 (poly16_t)
  Form of expected instruction(s): `vdup.16 q0, r0`
- poly8x16_t vdupq_n_p8 (poly8_t)
  Form of expected instruction(s): `vdup.8 q0, r0`
- uint64x2_t vdupq_n_u64 (uint64_t)
- int64x2_t vdupq_n_s64 (int64_t)

- uint32x2_t vmov_n_u32 (uint32_t)
  Form of expected instruction(s): `vdup.32 d0, r0`
- uint16x4_t vmov_n_u16 (uint16_t)
  Form of expected instruction(s): `vdup.16 d0, r0`
- uint8x8_t vmov_n_u8 (uint8_t)
  Form of expected instruction(s): `vdup.8 d0, r0`
- int32x2_t vmov_n_s32 (int32_t)
  Form of expected instruction(s): `vdup.32 d0, r0`
- int16x4_t vmov_n_s16 (int16_t)
  Form of expected instruction(s): `vdup.16 d0, r0`
- int8x8_t vmov_n_s8 (int8_t)
  Form of expected instruction(s): `vdup.8 d0, r0`
- float32x2_t vmov_n_f32 (float32_t)
  Form of expected instruction(s): `vdup.32 d0, r0`
- poly16x4_t vmov_n_p16 (poly16_t)
  Form of expected instruction(s): `vdup.16 d0, r0`
- poly8x8_t vmov_n_p8 (poly8_t)
  Form of expected instruction(s): `vdup.8 d0, r0`
- uint64x1_t vmov_n_u64 (uint64_t)
- int64x1_t vmov_n_s64 (int64_t)
- uint32x4_t vmovq_n_u32 (uint32_t)
  Form of expected instruction(s): `vdup.32 q0, r0`
- uint16x8_t vmovq_n_u16 (uint16_t)
  Form of expected instruction(s): `vdup.16 q0, r0`
- uint8x16_t vmovq_n_u8 (uint8_t)
  Form of expected instruction(s): `vdup.8 q0, r0`
- int32x4_t vmovq_n_s32 (int32_t)
  Form of expected instruction(s): `vdup.32 q0, r0`
- int16x8_t vmovq_n_s16 (int16_t)
  Form of expected instruction(s): `vdup.16 q0, r0`
- int8x16_t vmovq_n_s8 (int8_t)
  Form of expected instruction(s): `vdup.8 q0, r0`
- float32x4_t vmovq_n_f32 (float32_t)
  Form of expected instruction(s): `vdup.32 q0, r0`
- poly16x8_t vmovq_n_p16 (poly16_t)
  Form of expected instruction(s): `vdup.16 q0, r0`
- poly8x16_t vmovq_n_p8 (poly8_t)
  Form of expected instruction(s): `vdup.8 q0, r0`
- uint64x2_t vmovq_n_u64 (uint64_t)
- int64x2_t vmovq_n_s64 (int64_t)
- uint32x2_t vdup_lane_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vdup.32 d0, d0[0]`

- uint16x4_t vdup_lane_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vdup.16 d0, d0[0]`
- uint8x8_t vdup_lane_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vdup.8 d0, d0[0]`
- int32x2_t vdup_lane_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vdup.32 d0, d0[0]`
- int16x4_t vdup_lane_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vdup.16 d0, d0[0]`
- int8x8_t vdup_lane_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vdup.8 d0, d0[0]`
- float32x2_t vdup_lane_f32 (float32x2_t, const int)
  Form of expected instruction(s): `vdup.32 d0, d0[0]`
- poly16x4_t vdup_lane_p16 (poly16x4_t, const int)
  Form of expected instruction(s): `vdup.16 d0, d0[0]`
- poly8x8_t vdup_lane_p8 (poly8x8_t, const int)
  Form of expected instruction(s): `vdup.8 d0, d0[0]`
- uint64x1_t vdup_lane_u64 (uint64x1_t, const int)
- int64x1_t vdup_lane_s64 (int64x1_t, const int)
- uint32x4_t vdupq_lane_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vdup.32 q0, d0[0]`
- uint16x8_t vdupq_lane_u16 (uint16x4_t, const int)
  Form of expected instruction(s): `vdup.16 q0, d0[0]`
- uint8x16_t vdupq_lane_u8 (uint8x8_t, const int)
  Form of expected instruction(s): `vdup.8 q0, d0[0]`
- int32x4_t vdupq_lane_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vdup.32 q0, d0[0]`
- int16x8_t vdupq_lane_s16 (int16x4_t, const int)
  Form of expected instruction(s): `vdup.16 q0, d0[0]`
- int8x16_t vdupq_lane_s8 (int8x8_t, const int)
  Form of expected instruction(s): `vdup.8 q0, d0[0]`
- float32x4_t vdupq_lane_f32 (float32x2_t, const int)
  Form of expected instruction(s): `vdup.32 q0, d0[0]`
- poly16x8_t vdupq_lane_p16 (poly16x4_t, const int)
  Form of expected instruction(s): `vdup.16 q0, d0[0]`
- poly8x16_t vdupq_lane_p8 (poly8x8_t, const int)
  Form of expected instruction(s): `vdup.8 q0, d0[0]`
- uint64x2_t vdupq_lane_u64 (uint64x1_t, const int)
- int64x2_t vdupq_lane_s64 (int64x1_t, const int)

6.55.3.43 Combining vectors

- uint32x4_t vcombine_u32 (uint32x2_t, uint32x2_t)
- uint16x8_t vcombine_u16 (uint16x4_t, uint16x4_t)

- uint8x16_t vcombine_u8 (uint8x8_t, uint8x8_t)
- int32x4_t vcombine_s32 (int32x2_t, int32x2_t)
- int16x8_t vcombine_s16 (int16x4_t, int16x4_t)
- int8x16_t vcombine_s8 (int8x8_t, int8x8_t)
- uint64x2_t vcombine_u64 (uint64x1_t, uint64x1_t)
- int64x2_t vcombine_s64 (int64x1_t, int64x1_t)
- float32x4_t vcombine_f32 (float32x2_t, float32x2_t)
- poly16x8_t vcombine_p16 (poly16x4_t, poly16x4_t)
- poly8x16_t vcombine_p8 (poly8x8_t, poly8x8_t)

6.55.3.44 Splitting vectors

- uint32x2_t vget_high_u32 (uint32x4_t)
- uint16x4_t vget_high_u16 (uint16x8_t)
- uint8x8_t vget_high_u8 (uint8x16_t)
- int32x2_t vget_high_s32 (int32x4_t)
- int16x4_t vget_high_s16 (int16x8_t)
- int8x8_t vget_high_s8 (int8x16_t)
- uint64x1_t vget_high_u64 (uint64x2_t)
- int64x1_t vget_high_s64 (int64x2_t)
- float32x2_t vget_high_f32 (float32x4_t)
- poly16x4_t vget_high_p16 (poly16x8_t)
- poly8x8_t vget_high_p8 (poly8x16_t)
- uint32x2_t vget_low_u32 (uint32x4_t)
  Form of expected instruction(s): `vmov d0, d0`
- uint16x4_t vget_low_u16 (uint16x8_t)
  Form of expected instruction(s): `vmov d0, d0`
- uint8x8_t vget_low_u8 (uint8x16_t)
  Form of expected instruction(s): `vmov d0, d0`
- int32x2_t vget_low_s32 (int32x4_t)
  Form of expected instruction(s): `vmov d0, d0`
- int16x4_t vget_low_s16 (int16x8_t)
  Form of expected instruction(s): `vmov d0, d0`
- int8x8_t vget_low_s8 (int8x16_t)
  Form of expected instruction(s): `vmov d0, d0`
- float32x2_t vget_low_f32 (float32x4_t)
  Form of expected instruction(s): `vmov d0, d0`
- poly16x4_t vget_low_p16 (poly16x8_t)
  Form of expected instruction(s): `vmov d0, d0`
- poly8x8_t vget_low_p8 (poly8x16_t)
  Form of expected instruction(s): `vmov d0, d0`
- uint64x1_t vget_low_u64 (uint64x2_t)
- int64x1_t vget_low_s64 (int64x2_t)

6.55.3.45 Conversions

- float32x2_t vcvt_f32_u32 (uint32x2_t)
  Form of expected instruction(s): `vcvt.f32.u32 d0, d0`
- float32x2_t vcvt_f32_s32 (int32x2_t)
  Form of expected instruction(s): `vcvt.f32.s32 d0, d0`
- uint32x2_t vcvt_u32_f32 (float32x2_t)
  Form of expected instruction(s): `vcvt.u32.f32 d0, d0`
- int32x2_t vcvt_s32_f32 (float32x2_t)
  Form of expected instruction(s): `vcvt.s32.f32 d0, d0`
- float32x4_t vcvtq_f32_u32 (uint32x4_t)
  Form of expected instruction(s): `vcvt.f32.u32 q0, q0`
- float32x4_t vcvtq_f32_s32 (int32x4_t)
  Form of expected instruction(s): `vcvt.f32.s32 q0, q0`
- uint32x4_t vcvtq_u32_f32 (float32x4_t)
  Form of expected instruction(s): `vcvt.u32.f32 q0, q0`
- int32x4_t vcvtq_s32_f32 (float32x4_t)
  Form of expected instruction(s): `vcvt.s32.f32 q0, q0`
- float32x2_t vcvt_n_f32_u32 (uint32x2_t, const int)
  Form of expected instruction(s): `vcvt.f32.u32 d0, d0, #0`
- float32x2_t vcvt_n_f32_s32 (int32x2_t, const int)
  Form of expected instruction(s): `vcvt.f32.s32 d0, d0, #0`
- uint32x2_t vcvt_n_u32_f32 (float32x2_t, const int)
  Form of expected instruction(s): `vcvt.u32.f32 d0, d0, #0`
- int32x2_t vcvt_n_s32_f32 (float32x2_t, const int)
  Form of expected instruction(s): `vcvt.s32.f32 d0, d0, #0`
- float32x4_t vcvtq_n_f32_u32 (uint32x4_t, const int)
  Form of expected instruction(s): `vcvt.f32.u32 q0, q0, #0`
- float32x4_t vcvtq_n_f32_s32 (int32x4_t, const int)
  Form of expected instruction(s): `vcvt.f32.s32 q0, q0, #0`
- uint32x4_t vcvtq_n_u32_f32 (float32x4_t, const int)
  Form of expected instruction(s): `vcvt.u32.f32 q0, q0, #0`
- int32x4_t vcvtq_n_s32_f32 (float32x4_t, const int)
  Form of expected instruction(s): `vcvt.s32.f32 q0, q0, #0`

6.55.3.46 Move, single_opcode narrowing

- uint32x2_t vmovn_u64 (uint64x2_t)
  Form of expected instruction(s): `vmovn.i64 d0, q0`
- uint16x4_t vmovn_u32 (uint32x4_t)
  Form of expected instruction(s): `vmovn.i32 d0, q0`
- uint8x8_t vmovn_u16 (uint16x8_t)
  Form of expected instruction(s): `vmovn.i16 d0, q0`
- int32x2_t vmovn_s64 (int64x2_t)
  Form of expected instruction(s): `vmovn.i64 d0, q0`

- int16x4_t vmovn_s32 (int32x4_t)
  Form of expected instruction(s): `vmovn.i32 d0, q0`
- int8x8_t vmovn_s16 (int16x8_t)
  Form of expected instruction(s): `vmovn.i16 d0, q0`
- uint32x2_t vqmovn_u64 (uint64x2_t)
  Form of expected instruction(s): `vqmovn.u64 d0, q0`
- uint16x4_t vqmovn_u32 (uint32x4_t)
  Form of expected instruction(s): `vqmovn.u32 d0, q0`
- uint8x8_t vqmovn_u16 (uint16x8_t)
  Form of expected instruction(s): `vqmovn.u16 d0, q0`
- int32x2_t vqmovn_s64 (int64x2_t)
  Form of expected instruction(s): `vqmovn.s64 d0, q0`
- int16x4_t vqmovn_s32 (int32x4_t)
  Form of expected instruction(s): `vqmovn.s32 d0, q0`
- int8x8_t vqmovn_s16 (int16x8_t)
  Form of expected instruction(s): `vqmovn.s16 d0, q0`
- uint32x2_t vqmovun_s64 (int64x2_t)
  Form of expected instruction(s): `vqmovun.s64 d0, q0`
- uint16x4_t vqmovun_s32 (int32x4_t)
  Form of expected instruction(s): `vqmovun.s32 d0, q0`
- uint8x8_t vqmovun_s16 (int16x8_t)
  Form of expected instruction(s): `vqmovun.s16 d0, q0`

6.55.3.47 Move, single_opcode long

- uint64x2_t vmovl_u32 (uint32x2_t)
  Form of expected instruction(s): `vmovl.u32 q0, d0`
- uint32x4_t vmovl_u16 (uint16x4_t)
  Form of expected instruction(s): `vmovl.u16 q0, d0`
- uint16x8_t vmovl_u8 (uint8x8_t)
  Form of expected instruction(s): `vmovl.u8 q0, d0`
- int64x2_t vmovl_s32 (int32x2_t)
  Form of expected instruction(s): `vmovl.s32 q0, d0`
- int32x4_t vmovl_s16 (int16x4_t)
  Form of expected instruction(s): `vmovl.s16 q0, d0`
- int16x8_t vmovl_s8 (int8x8_t)
  Form of expected instruction(s): `vmovl.s8 q0, d0`

6.55.3.48 Table lookup

- poly8x8_t vtbl1_p8 (poly8x8_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0}, d0`
- int8x8_t vtbl1_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0}, d0`

- uint8x8_t vtbl1_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0}, d0`
- poly8x8_t vtbl2_p8 (poly8x8x2_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1}, d0`
- int8x8_t vtbl2_s8 (int8x8x2_t, int8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1}, d0`
- uint8x8_t vtbl2_u8 (uint8x8x2_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1}, d0`
- poly8x8_t vtbl3_p8 (poly8x8x3_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1, d2}, d0`
- int8x8_t vtbl3_s8 (int8x8x3_t, int8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1, d2}, d0`
- uint8x8_t vtbl3_u8 (uint8x8x3_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1, d2}, d0`
- poly8x8_t vtbl4_p8 (poly8x8x4_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1, d2, d3}, d0`
- int8x8_t vtbl4_s8 (int8x8x4_t, int8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1, d2, d3}, d0`
- uint8x8_t vtbl4_u8 (uint8x8x4_t, uint8x8_t)
  Form of expected instruction(s): `vtbl.8 d0, {d0, d1, d2, d3}, d0`

6.55.3.49 Extended table lookup

- poly8x8_t vtbx1_p8 (poly8x8_t, poly8x8_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0}, d0`
- int8x8_t vtbx1_s8 (int8x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0}, d0`
- uint8x8_t vtbx1_u8 (uint8x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0}, d0`
- poly8x8_t vtbx2_p8 (poly8x8_t, poly8x8x2_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1}, d0`
- int8x8_t vtbx2_s8 (int8x8_t, int8x8x2_t, int8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1}, d0`
- uint8x8_t vtbx2_u8 (uint8x8_t, uint8x8x2_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1}, d0`
- poly8x8_t vtbx3_p8 (poly8x8_t, poly8x8x3_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1, d2}, d0`
- int8x8_t vtbx3_s8 (int8x8_t, int8x8x3_t, int8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1, d2}, d0`
- uint8x8_t vtbx3_u8 (uint8x8_t, uint8x8x3_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1, d2}, d0`
- poly8x8_t vtbx4_p8 (poly8x8_t, poly8x8x4_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1, d2, d3}, d0`

- int8x8_t vtbx4_s8 (int8x8_t, int8x8x4_t, int8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1, d2, d3}, d0`
- uint8x8_t vtbx4_u8 (uint8x8_t, uint8x8x4_t, uint8x8_t)
  Form of expected instruction(s): `vtbx.8 d0, {d0, d1, d2, d3}, d0`

6.55.3.50 Multiply, lane

- float32x2_t vmul_lane_f32 (float32x2_t, float32x2_t, const int)
  Form of expected instruction(s): `vmul.f32 d0, d0, d0[0]`
- uint32x2_t vmul_lane_u32 (uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmul.i32 d0, d0, d0[0]`
- uint16x4_t vmul_lane_u16 (uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmul.i16 d0, d0, d0[0]`
- int32x2_t vmul_lane_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vmul.i32 d0, d0, d0[0]`
- int16x4_t vmul_lane_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vmul.i16 d0, d0, d0[0]`
- float32x4_t vmulq_lane_f32 (float32x4_t, float32x2_t, const int)
  Form of expected instruction(s): `vmul.f32 q0, q0, d0[0]`
- uint32x4_t vmulq_lane_u32 (uint32x4_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmul.i32 q0, q0, d0[0]`
- uint16x8_t vmulq_lane_u16 (uint16x8_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmul.i16 q0, q0, d0[0]`
- int32x4_t vmulq_lane_s32 (int32x4_t, int32x2_t, const int)
  Form of expected instruction(s): `vmul.i32 q0, q0, d0[0]`
- int16x8_t vmulq_lane_s16 (int16x8_t, int16x4_t, const int)
  Form of expected instruction(s): `vmul.i16 q0, q0, d0[0]`

6.55.3.51 Long multiply, lane

- uint64x2_t vmull_lane_u32 (uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmull.u32 q0, d0, d0[0]`
- uint32x4_t vmull_lane_u16 (uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmull.u16 q0, d0, d0[0]`
- int64x2_t vmull_lane_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vmull.s32 q0, d0, d0[0]`
- int32x4_t vmull_lane_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vmull.s16 q0, d0, d0[0]`

6.55.3.52 Saturating doubling long multiply, lane

- int64x2_t vqdmull_lane_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vqdmull.s32 q0, d0, d0[0]`
- int32x4_t vqdmull_lane_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vqdmull.s16 q0, d0, d0[0]`

6.55.3.53 Saturating doubling multiply high, lane

- int32x4_t vqdmulhq_lane_s32 (int32x4_t, int32x2_t, const int)
  Form of expected instruction(s): `vqdmulh.s32 q0, q0, d0[0]`
- int16x8_t vqdmulhq_lane_s16 (int16x8_t, int16x4_t, const int)
  Form of expected instruction(s): `vqdmulh.s16 q0, q0, d0[0]`
- int32x2_t vqdmulh_lane_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vqdmulh.s32 d0, d0, d0[0]`
- int16x4_t vqdmulh_lane_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vqdmulh.s16 d0, d0, d0[0]`
- int32x4_t vqrdmulhq_lane_s32 (int32x4_t, int32x2_t, const int)
  Form of expected instruction(s): `vqrdmulh.s32 q0, q0, d0[0]`
- int16x8_t vqrdmulhq_lane_s16 (int16x8_t, int16x4_t, const int)
  Form of expected instruction(s): `vqrdmulh.s16 q0, q0, d0[0]`
- int32x2_t vqrdmulh_lane_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vqrdmulh.s32 d0, d0, d0[0]`
- int16x4_t vqrdmulh_lane_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vqrdmulh.s16 d0, d0, d0[0]`

6.55.3.54 Multiply-accumulate, lane

- float32x2_t vmla_lane_f32 (float32x2_t, float32x2_t, float32x2_t, const int)
  Form of expected instruction(s): `vmla.f32 d0, d0, d0[0]`
- uint32x2_t vmla_lane_u32 (uint32x2_t, uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmla.i32 d0, d0, d0[0]`
- uint16x4_t vmla_lane_u16 (uint16x4_t, uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmla.i16 d0, d0, d0[0]`
- int32x2_t vmla_lane_s32 (int32x2_t, int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vmla.i32 d0, d0, d0[0]`
- int16x4_t vmla_lane_s16 (int16x4_t, int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vmla.i16 d0, d0, d0[0]`
- float32x4_t vmlaq_lane_f32 (float32x4_t, float32x4_t, float32x2_t, const int)
  Form of expected instruction(s): `vmla.f32 q0, q0, d0[0]`
- uint32x4_t vmlaq_lane_u32 (uint32x4_t, uint32x4_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmla.i32 q0, q0, d0[0]`
- uint16x8_t vmlaq_lane_u16 (uint16x8_t, uint16x8_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmla.i16 q0, q0, d0[0]`
- int32x4_t vmlaq_lane_s32 (int32x4_t, int32x4_t, int32x2_t, const int)
  Form of expected instruction(s): `vmla.i32 q0, q0, d0[0]`
- int16x8_t vmlaq_lane_s16 (int16x8_t, int16x8_t, int16x4_t, const int)
  Form of expected instruction(s): `vmla.i16 q0, q0, d0[0]`
- uint64x2_t vmlal_lane_u32 (uint64x2_t, uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmlal.u32 q0, d0, d0[0]`
- uint32x4_t vmlal_lane_u16 (uint32x4_t, uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmlal.u16 q0, d0, d0[0]`

- int64x2_t vmlal_lane_s32 (int64x2_t, int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vmlal.s32 q0, d0, d0[0]`
- int32x4_t vmlal_lane_s16 (int32x4_t, int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vmlal.s16 q0, d0, d0[0]`
- int64x2_t vqdmlal_lane_s32 (int64x2_t, int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vqdmlal.s32 q0, d0, d0[0]`
- int32x4_t vqdmlal_lane_s16 (int32x4_t, int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vqdmlal.s16 q0, d0, d0[0]`

6.55.3.55 Multiply-subtract, lane

- float32x2_t vmls_lane_f32 (float32x2_t, float32x2_t, float32x2_t, const int)
  Form of expected instruction(s): `vmls.f32 d0, d0, d0[0]`
- uint32x2_t vmls_lane_u32 (uint32x2_t, uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmls.i32 d0, d0, d0[0]`
- uint16x4_t vmls_lane_u16 (uint16x4_t, uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmls.i16 d0, d0, d0[0]`
- int32x2_t vmls_lane_s32 (int32x2_t, int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vmls.i32 d0, d0, d0[0]`
- int16x4_t vmls_lane_s16 (int16x4_t, int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vmls.i16 d0, d0, d0[0]`
- float32x4_t vmlsq_lane_f32 (float32x4_t, float32x4_t, float32x2_t, const int)
  Form of expected instruction(s): `vmls.f32 q0, q0, d0[0]`
- uint32x4_t vmlsq_lane_u32 (uint32x4_t, uint32x4_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmls.i32 q0, q0, d0[0]`
- uint16x8_t vmlsq_lane_u16 (uint16x8_t, uint16x8_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmls.i16 q0, q0, d0[0]`
- int32x4_t vmlsq_lane_s32 (int32x4_t, int32x4_t, int32x2_t, const int)
  Form of expected instruction(s): `vmls.i32 q0, q0, d0[0]`
- int16x8_t vmlsq_lane_s16 (int16x8_t, int16x8_t, int16x4_t, const int)
  Form of expected instruction(s): `vmls.i16 q0, q0, d0[0]`
- uint64x2_t vmlsl_lane_u32 (uint64x2_t, uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vmlsl.u32 q0, d0, d0[0]`
- uint32x4_t vmlsl_lane_u16 (uint32x4_t, uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vmlsl.u16 q0, d0, d0[0]`
- int64x2_t vmlsl_lane_s32 (int64x2_t, int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vmlsl.s32 q0, d0, d0[0]`
- int32x4_t vmlsl_lane_s16 (int32x4_t, int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vmlsl.s16 q0, d0, d0[0]`
- int64x2_t vqdmlsl_lane_s32 (int64x2_t, int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vqdmlsl.s32 q0, d0, d0[0]`
- int32x4_t vqdmlsl_lane_s16 (int32x4_t, int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vqdmlsl.s16 q0, d0, d0[0]`

6.55.3.56 Vector multiply by scalar

- float32x2_t vmul_n_f32 (float32x2_t, float32_t)
  Form of expected instruction(s): `vmul.f32 d0, d0, d0[0]`
- uint32x2_t vmul_n_u32 (uint32x2_t, uint32_t)
  Form of expected instruction(s): `vmul.i32 d0, d0, d0[0]`
- uint16x4_t vmul_n_u16 (uint16x4_t, uint16_t)
  Form of expected instruction(s): `vmul.i16 d0, d0, d0[0]`
- int32x2_t vmul_n_s32 (int32x2_t, int32_t)
  Form of expected instruction(s): `vmul.i32 d0, d0, d0[0]`
- int16x4_t vmul_n_s16 (int16x4_t, int16_t)
  Form of expected instruction(s): `vmul.i16 d0, d0, d0[0]`
- float32x4_t vmulq_n_f32 (float32x4_t, float32_t)
  Form of expected instruction(s): `vmul.f32 q0, q0, d0[0]`
- uint32x4_t vmulq_n_u32 (uint32x4_t, uint32_t)
  Form of expected instruction(s): `vmul.i32 q0, q0, d0[0]`
- uint16x8_t vmulq_n_u16 (uint16x8_t, uint16_t)
  Form of expected instruction(s): `vmul.i16 q0, q0, d0[0]`
- int32x4_t vmulq_n_s32 (int32x4_t, int32_t)
  Form of expected instruction(s): `vmul.i32 q0, q0, d0[0]`
- int16x8_t vmulq_n_s16 (int16x8_t, int16_t)
  Form of expected instruction(s): `vmul.i16 q0, q0, d0[0]`

6.55.3.57 Vector long multiply by scalar

- uint64x2_t vmull_n_u32 (uint32x2_t, uint32_t)
  Form of expected instruction(s): `vmull.u32 q0, d0, d0[0]`
- uint32x4_t vmull_n_u16 (uint16x4_t, uint16_t)
  Form of expected instruction(s): `vmull.u16 q0, d0, d0[0]`
- int64x2_t vmull_n_s32 (int32x2_t, int32_t)
  Form of expected instruction(s): `vmull.s32 q0, d0, d0[0]`
- int32x4_t vmull_n_s16 (int16x4_t, int16_t)
  Form of expected instruction(s): `vmull.s16 q0, d0, d0[0]`

6.55.3.58 Vector saturating doubling long multiply by scalar

- int64x2_t vqdmull_n_s32 (int32x2_t, int32_t)
  Form of expected instruction(s): `vqdmull.s32 q0, d0, d0[0]`
- int32x4_t vqdmull_n_s16 (int16x4_t, int16_t)
  Form of expected instruction(s): `vqdmull.s16 q0, d0, d0[0]`

6.55.3.59 Vector saturating doubling multiply high by scalar

- int32x4_t vqdmulhq_n_s32 (int32x4_t, int32_t)
  Form of expected instruction(s): `vqdmulh.s32 q0, q0, d0[0]`
- int16x8_t vqdmulhq_n_s16 (int16x8_t, int16_t)
  Form of expected instruction(s): `vqdmulh.s16 q0, q0, d0[0]`

- int32x2_t vqdmulh_n_s32 (int32x2_t, int32_t)
  Form of expected instruction(s): `vqdmulh.s32 d0, d0, d0[0]`
- int16x4_t vqdmulh_n_s16 (int16x4_t, int16_t)
  Form of expected instruction(s): `vqdmulh.s16 d0, d0, d0[0]`
- int32x4_t vqrdmulhq_n_s32 (int32x4_t, int32_t)
  Form of expected instruction(s): `vqrdmulh.s32 q0, q0, d0[0]`
- int16x8_t vqrdmulhq_n_s16 (int16x8_t, int16_t)
  Form of expected instruction(s): `vqrdmulh.s16 q0, q0, d0[0]`
- int32x2_t vqrdmulh_n_s32 (int32x2_t, int32_t)
  Form of expected instruction(s): `vqrdmulh.s32 d0, d0, d0[0]`
- int16x4_t vqrdmulh_n_s16 (int16x4_t, int16_t)
  Form of expected instruction(s): `vqrdmulh.s16 d0, d0, d0[0]`

6.55.3.60 Vector multiply-accumulate by scalar

- float32x2_t vmla_n_f32 (float32x2_t, float32x2_t, float32_t)
  Form of expected instruction(s): `vmla.f32 d0, d0, d0[0]`
- uint32x2_t vmla_n_u32 (uint32x2_t, uint32x2_t, uint32_t)
  Form of expected instruction(s): `vmla.i32 d0, d0, d0[0]`
- uint16x4_t vmla_n_u16 (uint16x4_t, uint16x4_t, uint16_t)
  Form of expected instruction(s): `vmla.i16 d0, d0, d0[0]`
- int32x2_t vmla_n_s32 (int32x2_t, int32x2_t, int32_t)
  Form of expected instruction(s): `vmla.i32 d0, d0, d0[0]`
- int16x4_t vmla_n_s16 (int16x4_t, int16x4_t, int16_t)
  Form of expected instruction(s): `vmla.i16 d0, d0, d0[0]`
- float32x4_t vmlaq_n_f32 (float32x4_t, float32x4_t, float32_t)
  Form of expected instruction(s): `vmla.f32 q0, q0, d0[0]`
- uint32x4_t vmlaq_n_u32 (uint32x4_t, uint32x4_t, uint32_t)
  Form of expected instruction(s): `vmla.i32 q0, q0, d0[0]`
- uint16x8_t vmlaq_n_u16 (uint16x8_t, uint16x8_t, uint16_t)
  Form of expected instruction(s): `vmla.i16 q0, q0, d0[0]`
- int32x4_t vmlaq_n_s32 (int32x4_t, int32x4_t, int32_t)
  Form of expected instruction(s): `vmla.i32 q0, q0, d0[0]`
- int16x8_t vmlaq_n_s16 (int16x8_t, int16x8_t, int16_t)
  Form of expected instruction(s): `vmla.i16 q0, q0, d0[0]`
- uint64x2_t vmlal_n_u32 (uint64x2_t, uint32x2_t, uint32_t)
  Form of expected instruction(s): `vmlal.u32 q0, d0, d0[0]`
- uint32x4_t vmlal_n_u16 (uint32x4_t, uint16x4_t, uint16_t)
  Form of expected instruction(s): `vmlal.u16 q0, d0, d0[0]`
- int64x2_t vmlal_n_s32 (int64x2_t, int32x2_t, int32_t)
  Form of expected instruction(s): `vmlal.s32 q0, d0, d0[0]`
- int32x4_t vmlal_n_s16 (int32x4_t, int16x4_t, int16_t)
  Form of expected instruction(s): `vmlal.s16 q0, d0, d0[0]`

- int64x2_t vqdmlal_n_s32 (int64x2_t, int32x2_t, int32_t)
  Form of expected instruction(s): `vqdmlal.s32 q0, d0, d0[0]`
- int32x4_t vqdmlal_n_s16 (int32x4_t, int16x4_t, int16_t)
  Form of expected instruction(s): `vqdmlal.s16 q0, d0, d0[0]`

6.55.3.61 Vector multiply-subtract by scalar

- float32x2_t vmls_n_f32 (float32x2_t, float32x2_t, float32_t)
  Form of expected instruction(s): `vmls.f32 d0, d0, d0[0]`
- uint32x2_t vmls_n_u32 (uint32x2_t, uint32x2_t, uint32_t)
  Form of expected instruction(s): `vmls.i32 d0, d0, d0[0]`
- uint16x4_t vmls_n_u16 (uint16x4_t, uint16x4_t, uint16_t)
  Form of expected instruction(s): `vmls.i16 d0, d0, d0[0]`
- int32x2_t vmls_n_s32 (int32x2_t, int32x2_t, int32_t)
  Form of expected instruction(s): `vmls.i32 d0, d0, d0[0]`
- int16x4_t vmls_n_s16 (int16x4_t, int16x4_t, int16_t)
  Form of expected instruction(s): `vmls.i16 d0, d0, d0[0]`
- float32x4_t vmlsq_n_f32 (float32x4_t, float32x4_t, float32_t)
  Form of expected instruction(s): `vmls.f32 q0, q0, d0[0]`
- uint32x4_t vmlsq_n_u32 (uint32x4_t, uint32x4_t, uint32_t)
  Form of expected instruction(s): `vmls.i32 q0, q0, d0[0]`
- uint16x8_t vmlsq_n_u16 (uint16x8_t, uint16x8_t, uint16_t)
  Form of expected instruction(s): `vmls.i16 q0, q0, d0[0]`
- int32x4_t vmlsq_n_s32 (int32x4_t, int32x4_t, int32_t)
  Form of expected instruction(s): `vmls.i32 q0, q0, d0[0]`
- int16x8_t vmlsq_n_s16 (int16x8_t, int16x8_t, int16_t)
  Form of expected instruction(s): `vmls.i16 q0, q0, d0[0]`
- uint64x2_t vmlsl_n_u32 (uint64x2_t, uint32x2_t, uint32_t)
  Form of expected instruction(s): `vmlsl.u32 q0, d0, d0[0]`
- uint32x4_t vmlsl_n_u16 (uint32x4_t, uint16x4_t, uint16_t)
  Form of expected instruction(s): `vmlsl.u16 q0, d0, d0[0]`
- int64x2_t vmlsl_n_s32 (int64x2_t, int32x2_t, int32_t)
  Form of expected instruction(s): `vmlsl.s32 q0, d0, d0[0]`
- int32x4_t vmlsl_n_s16 (int32x4_t, int16x4_t, int16_t)
  Form of expected instruction(s): `vmlsl.s16 q0, d0, d0[0]`
- int64x2_t vqdmlsl_n_s32 (int64x2_t, int32x2_t, int32_t)
  Form of expected instruction(s): `vqdmlsl.s32 q0, d0, d0[0]`
- int32x4_t vqdmlsl_n_s16 (int32x4_t, int16x4_t, int16_t)
  Form of expected instruction(s): `vqdmlsl.s16 q0, d0, d0[0]`

6.55.3.62 Vector extract

- uint32x2_t vext_u32 (uint32x2_t, uint32x2_t, const int)
  Form of expected instruction(s): `vext.32 d0, d0, d0, #0`

- uint16x4_t vext_u16 (uint16x4_t, uint16x4_t, const int)
  Form of expected instruction(s): `vext.16 d0, d0, d0, #0`
- uint8x8_t vext_u8 (uint8x8_t, uint8x8_t, const int)
  Form of expected instruction(s): `vext.8 d0, d0, d0, #0`
- int32x2_t vext_s32 (int32x2_t, int32x2_t, const int)
  Form of expected instruction(s): `vext.32 d0, d0, d0, #0`
- int16x4_t vext_s16 (int16x4_t, int16x4_t, const int)
  Form of expected instruction(s): `vext.16 d0, d0, d0, #0`
- int8x8_t vext_s8 (int8x8_t, int8x8_t, const int)
  Form of expected instruction(s): `vext.8 d0, d0, d0, #0`
- uint64x1_t vext_u64 (uint64x1_t, uint64x1_t, const int)
  Form of expected instruction(s): `vext.64 d0, d0, d0, #0`
- int64x1_t vext_s64 (int64x1_t, int64x1_t, const int)
  Form of expected instruction(s): `vext.64 d0, d0, d0, #0`
- float32x2_t vext_f32 (float32x2_t, float32x2_t, const int)
  Form of expected instruction(s): `vext.32 d0, d0, d0, #0`
- poly16x4_t vext_p16 (poly16x4_t, poly16x4_t, const int)
  Form of expected instruction(s): `vext.16 d0, d0, d0, #0`
- poly8x8_t vext_p8 (poly8x8_t, poly8x8_t, const int)
  Form of expected instruction(s): `vext.8 d0, d0, d0, #0`
- uint32x4_t vextq_u32 (uint32x4_t, uint32x4_t, const int)
  Form of expected instruction(s): `vext.32 q0, q0, q0, #0`
- uint16x8_t vextq_u16 (uint16x8_t, uint16x8_t, const int)
  Form of expected instruction(s): `vext.16 q0, q0, q0, #0`
- uint8x16_t vextq_u8 (uint8x16_t, uint8x16_t, const int)
  Form of expected instruction(s): `vext.8 q0, q0, q0, #0`
- int32x4_t vextq_s32 (int32x4_t, int32x4_t, const int)
  Form of expected instruction(s): `vext.32 q0, q0, q0, #0`
- int16x8_t vextq_s16 (int16x8_t, int16x8_t, const int)
  Form of expected instruction(s): `vext.16 q0, q0, q0, #0`
- int8x16_t vextq_s8 (int8x16_t, int8x16_t, const int)
  Form of expected instruction(s): `vext.8 q0, q0, q0, #0`
- uint64x2_t vextq_u64 (uint64x2_t, uint64x2_t, const int)
  Form of expected instruction(s): `vext.64 q0, q0, q0, #0`
- int64x2_t vextq_s64 (int64x2_t, int64x2_t, const int)
  Form of expected instruction(s): `vext.64 q0, q0, q0, #0`
- float32x4_t vextq_f32 (float32x4_t, float32x4_t, const int)
  Form of expected instruction(s): `vext.32 q0, q0, q0, #0`
- poly16x8_t vextq_p16 (poly16x8_t, poly16x8_t, const int)
  Form of expected instruction(s): `vext.16 q0, q0, q0, #0`
- poly8x16_t vextq_p8 (poly8x16_t, poly8x16_t, const int)
  Form of expected instruction(s): `vext.8 q0, q0, q0, #0`

6.55.3.63 Reverse elements

- uint32x2_t vrev64_u32 (uint32x2_t)
  Form of expected instruction(s): `vrev64.32 d0, d0`
- uint16x4_t vrev64_u16 (uint16x4_t)
  Form of expected instruction(s): `vrev64.16 d0, d0`
- uint8x8_t vrev64_u8 (uint8x8_t)
  Form of expected instruction(s): `vrev64.8 d0, d0`
- int32x2_t vrev64_s32 (int32x2_t)
  Form of expected instruction(s): `vrev64.32 d0, d0`
- int16x4_t vrev64_s16 (int16x4_t)
  Form of expected instruction(s): `vrev64.16 d0, d0`
- int8x8_t vrev64_s8 (int8x8_t)
  Form of expected instruction(s): `vrev64.8 d0, d0`
- float32x2_t vrev64_f32 (float32x2_t)
  Form of expected instruction(s): `vrev64.32 d0, d0`
- poly16x4_t vrev64_p16 (poly16x4_t)
  Form of expected instruction(s): `vrev64.16 d0, d0`
- poly8x8_t vrev64_p8 (poly8x8_t)
  Form of expected instruction(s): `vrev64.8 d0, d0`
- uint32x4_t vrev64q_u32 (uint32x4_t)
  Form of expected instruction(s): `vrev64.32 q0, q0`
- uint16x8_t vrev64q_u16 (uint16x8_t)
  Form of expected instruction(s): `vrev64.16 q0, q0`
- uint8x16_t vrev64q_u8 (uint8x16_t)
  Form of expected instruction(s): `vrev64.8 q0, q0`
- int32x4_t vrev64q_s32 (int32x4_t)
  Form of expected instruction(s): `vrev64.32 q0, q0`
- int16x8_t vrev64q_s16 (int16x8_t)
  Form of expected instruction(s): `vrev64.16 q0, q0`
- int8x16_t vrev64q_s8 (int8x16_t)
  Form of expected instruction(s): `vrev64.8 q0, q0`
- float32x4_t vrev64q_f32 (float32x4_t)
  Form of expected instruction(s): `vrev64.32 q0, q0`
- poly16x8_t vrev64q_p16 (poly16x8_t)
  Form of expected instruction(s): `vrev64.16 q0, q0`
- poly8x16_t vrev64q_p8 (poly8x16_t)
  Form of expected instruction(s): `vrev64.8 q0, q0`
- uint16x4_t vrev32_u16 (uint16x4_t)
  Form of expected instruction(s): `vrev32.16 d0, d0`
- int16x4_t vrev32_s16 (int16x4_t)
  Form of expected instruction(s): `vrev32.16 d0, d0`
- uint8x8_t vrev32_u8 (uint8x8_t)
  Form of expected instruction(s): `vrev32.8 d0, d0`

- int8x8_t vrev32_s8 (int8x8_t)
  Form of expected instruction(s): `vrev32.8 d0, d0`
- poly16x4_t vrev32_p16 (poly16x4_t)
  Form of expected instruction(s): `vrev32.16 d0, d0`
- poly8x8_t vrev32_p8 (poly8x8_t)
  Form of expected instruction(s): `vrev32.8 d0, d0`
- uint16x8_t vrev32q_u16 (uint16x8_t)
  Form of expected instruction(s): `vrev32.16 q0, q0`
- int16x8_t vrev32q_s16 (int16x8_t)
  Form of expected instruction(s): `vrev32.16 q0, q0`
- uint8x16_t vrev32q_u8 (uint8x16_t)
  Form of expected instruction(s): `vrev32.8 q0, q0`
- int8x16_t vrev32q_s8 (int8x16_t)
  Form of expected instruction(s): `vrev32.8 q0, q0`
- poly16x8_t vrev32q_p16 (poly16x8_t)
  Form of expected instruction(s): `vrev32.16 q0, q0`
- poly8x16_t vrev32q_p8 (poly8x16_t)
  Form of expected instruction(s): `vrev32.8 q0, q0`
- uint8x8_t vrev16_u8 (uint8x8_t)
  Form of expected instruction(s): `vrev16.8 d0, d0`
- int8x8_t vrev16_s8 (int8x8_t)
  Form of expected instruction(s): `vrev16.8 d0, d0`
- poly8x8_t vrev16_p8 (poly8x8_t)
  Form of expected instruction(s): `vrev16.8 d0, d0`
- uint8x16_t vrev16q_u8 (uint8x16_t)
  Form of expected instruction(s): `vrev16.8 q0, q0`
- int8x16_t vrev16q_s8 (int8x16_t)
  Form of expected instruction(s): `vrev16.8 q0, q0`
- poly8x16_t vrev16q_p8 (poly8x16_t)
  Form of expected instruction(s): `vrev16.8 q0, q0`

6.55.3.64 Bit selection

- uint32x2_t vbsl_u32 (uint32x2_t, uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- uint16x4_t vbsl_u16 (uint16x4_t, uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- uint8x8_t vbsl_u8 (uint8x8_t, uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- int32x2_t vbsl_s32 (uint32x2_t, int32x2_t, int32x2_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`

- int16x4_t vbsl_s16 (uint16x4_t, int16x4_t, int16x4_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- int8x8_t vbsl_s8 (uint8x8_t, int8x8_t, int8x8_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- uint64x1_t vbsl_u64 (uint64x1_t, uint64x1_t, uint64x1_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- int64x1_t vbsl_s64 (uint64x1_t, int64x1_t, int64x1_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- float32x2_t vbsl_f32 (uint32x2_t, float32x2_t, float32x2_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- poly16x4_t vbsl_p16 (uint16x4_t, poly16x4_t, poly16x4_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- poly8x8_t vbsl_p8 (uint8x8_t, poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vbsl d0, d0, d0` *or* `vbit d0, d0, d0` *or* `vbif d0, d0, d0`
- uint32x4_t vbslq_u32 (uint32x4_t, uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- uint16x8_t vbslq_u16 (uint16x8_t, uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- uint8x16_t vbslq_u8 (uint8x16_t, uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- int32x4_t vbslq_s32 (uint32x4_t, int32x4_t, int32x4_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- int16x8_t vbslq_s16 (uint16x8_t, int16x8_t, int16x8_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- int8x16_t vbslq_s8 (uint8x16_t, int8x16_t, int8x16_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- uint64x2_t vbslq_u64 (uint64x2_t, uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`

- int64x2_t vbslq_s64 (uint64x2_t, int64x2_t, int64x2_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- float32x4_t vbslq_f32 (uint32x4_t, float32x4_t, float32x4_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- poly16x8_t vbslq_p16 (uint16x8_t, poly16x8_t, poly16x8_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`
- poly8x16_t vbslq_p8 (uint8x16_t, poly8x16_t, poly8x16_t)
  Form of expected instruction(s): `vbsl q0, q0, q0` *or* `vbit q0, q0, q0` *or* `vbif q0, q0, q0`

6.55.3.65 Transpose elements

- uint32x2x2_t vtrn_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vtrn.32 d0, d1`
- uint16x4x2_t vtrn_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vtrn.16 d0, d1`
- uint8x8x2_t vtrn_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vtrn.8 d0, d1`
- int32x2x2_t vtrn_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vtrn.32 d0, d1`
- int16x4x2_t vtrn_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vtrn.16 d0, d1`
- int8x8x2_t vtrn_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vtrn.8 d0, d1`
- float32x2x2_t vtrn_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vtrn.32 d0, d1`
- poly16x4x2_t vtrn_p16 (poly16x4_t, poly16x4_t)
  Form of expected instruction(s): `vtrn.16 d0, d1`
- poly8x8x2_t vtrn_p8 (poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vtrn.8 d0, d1`
- uint32x4x2_t vtrnq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vtrn.32 q0, q1`
- uint16x8x2_t vtrnq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vtrn.16 q0, q1`
- uint8x16x2_t vtrnq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vtrn.8 q0, q1`
- int32x4x2_t vtrnq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vtrn.32 q0, q1`
- int16x8x2_t vtrnq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vtrn.16 q0, q1`
- int8x16x2_t vtrnq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vtrn.8 q0, q1`

- float32x4x2_t vtrnq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vtrn.32 q0, q1`
- poly16x8x2_t vtrnq_p16 (poly16x8_t, poly16x8_t)
  Form of expected instruction(s): `vtrn.16 q0, q1`
- poly8x16x2_t vtrnq_p8 (poly8x16_t, poly8x16_t)
  Form of expected instruction(s): `vtrn.8 q0, q1`

6.55.3.66 Zip elements

- uint32x2x2_t vzip_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vzip.32 d0, d1`
- uint16x4x2_t vzip_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vzip.16 d0, d1`
- uint8x8x2_t vzip_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vzip.8 d0, d1`
- int32x2x2_t vzip_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vzip.32 d0, d1`
- int16x4x2_t vzip_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vzip.16 d0, d1`
- int8x8x2_t vzip_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vzip.8 d0, d1`
- float32x2x2_t vzip_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vzip.32 d0, d1`
- poly16x4x2_t vzip_p16 (poly16x4_t, poly16x4_t)
  Form of expected instruction(s): `vzip.16 d0, d1`
- poly8x8x2_t vzip_p8 (poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vzip.8 d0, d1`
- uint32x4x2_t vzipq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vzip.32 q0, q1`
- uint16x8x2_t vzipq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vzip.16 q0, q1`
- uint8x16x2_t vzipq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vzip.8 q0, q1`
- int32x4x2_t vzipq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vzip.32 q0, q1`
- int16x8x2_t vzipq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vzip.16 q0, q1`
- int8x16x2_t vzipq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vzip.8 q0, q1`
- float32x4x2_t vzipq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vzip.32 q0, q1`
- poly16x8x2_t vzipq_p16 (poly16x8_t, poly16x8_t)
  Form of expected instruction(s): `vzip.16 q0, q1`

- poly8x16x2_t vzipq_p8 (poly8x16_t, poly8x16_t)
  Form of expected instruction(s): `vzip.8 q0, q1`

6.55.3.67 Unzip elements

- uint32x2x2_t vuzp_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vuzp.32 d0, d1`
- uint16x4x2_t vuzp_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vuzp.16 d0, d1`
- uint8x8x2_t vuzp_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vuzp.8 d0, d1`
- int32x2x2_t vuzp_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vuzp.32 d0, d1`
- int16x4x2_t vuzp_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vuzp.16 d0, d1`
- int8x8x2_t vuzp_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vuzp.8 d0, d1`
- float32x2x2_t vuzp_f32 (float32x2_t, float32x2_t)
  Form of expected instruction(s): `vuzp.32 d0, d1`
- poly16x4x2_t vuzp_p16 (poly16x4_t, poly16x4_t)
  Form of expected instruction(s): `vuzp.16 d0, d1`
- poly8x8x2_t vuzp_p8 (poly8x8_t, poly8x8_t)
  Form of expected instruction(s): `vuzp.8 d0, d1`
- uint32x4x2_t vuzpq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vuzp.32 q0, q1`
- uint16x8x2_t vuzpq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vuzp.16 q0, q1`
- uint8x16x2_t vuzpq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vuzp.8 q0, q1`
- int32x4x2_t vuzpq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vuzp.32 q0, q1`
- int16x8x2_t vuzpq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vuzp.16 q0, q1`
- int8x16x2_t vuzpq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vuzp.8 q0, q1`
- float32x4x2_t vuzpq_f32 (float32x4_t, float32x4_t)
  Form of expected instruction(s): `vuzp.32 q0, q1`
- poly16x8x2_t vuzpq_p16 (poly16x8_t, poly16x8_t)
  Form of expected instruction(s): `vuzp.16 q0, q1`
- poly8x16x2_t vuzpq_p8 (poly8x16_t, poly8x16_t)
  Form of expected instruction(s): `vuzp.8 q0, q1`

6.55.3.68 Element/structure loads, VLD1 variants

- uint32x2_t vld1_u32 (const uint32_t *)
  Form of expected instruction(s): `vld1.32 {d0}, [r0]`

- uint16x4_t vld1_u16 (const uint16_t *)
  Form of expected instruction(s): `vld1.16 {d0}, [r0]`
- uint8x8_t vld1_u8 (const uint8_t *)
  Form of expected instruction(s): `vld1.8 {d0}, [r0]`
- int32x2_t vld1_s32 (const int32_t *)
  Form of expected instruction(s): `vld1.32 {d0}, [r0]`
- int16x4_t vld1_s16 (const int16_t *)
  Form of expected instruction(s): `vld1.16 {d0}, [r0]`
- int8x8_t vld1_s8 (const int8_t *)
  Form of expected instruction(s): `vld1.8 {d0}, [r0]`
- uint64x1_t vld1_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`
- int64x1_t vld1_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`
- float32x2_t vld1_f32 (const float32_t *)
  Form of expected instruction(s): `vld1.32 {d0}, [r0]`
- poly16x4_t vld1_p16 (const poly16_t *)
  Form of expected instruction(s): `vld1.16 {d0}, [r0]`
- poly8x8_t vld1_p8 (const poly8_t *)
  Form of expected instruction(s): `vld1.8 {d0}, [r0]`
- uint32x4_t vld1q_u32 (const uint32_t *)
  Form of expected instruction(s): `vld1.32 {d0, d1}, [r0]`
- uint16x8_t vld1q_u16 (const uint16_t *)
  Form of expected instruction(s): `vld1.16 {d0, d1}, [r0]`
- uint8x16_t vld1q_u8 (const uint8_t *)
  Form of expected instruction(s): `vld1.8 {d0, d1}, [r0]`
- int32x4_t vld1q_s32 (const int32_t *)
  Form of expected instruction(s): `vld1.32 {d0, d1}, [r0]`
- int16x8_t vld1q_s16 (const int16_t *)
  Form of expected instruction(s): `vld1.16 {d0, d1}, [r0]`
- int8x16_t vld1q_s8 (const int8_t *)
  Form of expected instruction(s): `vld1.8 {d0, d1}, [r0]`
- uint64x2_t vld1q_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`
- int64x2_t vld1q_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`
- float32x4_t vld1q_f32 (const float32_t *)
  Form of expected instruction(s): `vld1.32 {d0, d1}, [r0]`
- poly16x8_t vld1q_p16 (const poly16_t *)
  Form of expected instruction(s): `vld1.16 {d0, d1}, [r0]`
- poly8x16_t vld1q_p8 (const poly8_t *)
  Form of expected instruction(s): `vld1.8 {d0, d1}, [r0]`

- uint32x2_t vld1_lane_u32 (const uint32_t *, uint32x2_t, const int)
  Form of expected instruction(s): `vld1.32 {d0[0]}, [r0]`
- uint16x4_t vld1_lane_u16 (const uint16_t *, uint16x4_t, const int)
  Form of expected instruction(s): `vld1.16 {d0[0]}, [r0]`
- uint8x8_t vld1_lane_u8 (const uint8_t *, uint8x8_t, const int)
  Form of expected instruction(s): `vld1.8 {d0[0]}, [r0]`
- int32x2_t vld1_lane_s32 (const int32_t *, int32x2_t, const int)
  Form of expected instruction(s): `vld1.32 {d0[0]}, [r0]`
- int16x4_t vld1_lane_s16 (const int16_t *, int16x4_t, const int)
  Form of expected instruction(s): `vld1.16 {d0[0]}, [r0]`
- int8x8_t vld1_lane_s8 (const int8_t *, int8x8_t, const int)
  Form of expected instruction(s): `vld1.8 {d0[0]}, [r0]`
- float32x2_t vld1_lane_f32 (const float32_t *, float32x2_t, const int)
  Form of expected instruction(s): `vld1.32 {d0[0]}, [r0]`
- poly16x4_t vld1_lane_p16 (const poly16_t *, poly16x4_t, const int)
  Form of expected instruction(s): `vld1.16 {d0[0]}, [r0]`
- poly8x8_t vld1_lane_p8 (const poly8_t *, poly8x8_t, const int)
  Form of expected instruction(s): `vld1.8 {d0[0]}, [r0]`
- uint64x1_t vld1_lane_u64 (const uint64_t *, uint64x1_t, const int)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`
- int64x1_t vld1_lane_s64 (const int64_t *, int64x1_t, const int)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`
- uint32x4_t vld1q_lane_u32 (const uint32_t *, uint32x4_t, const int)
  Form of expected instruction(s): `vld1.32 {d0[0]}, [r0]`
- uint16x8_t vld1q_lane_u16 (const uint16_t *, uint16x8_t, const int)
  Form of expected instruction(s): `vld1.16 {d0[0]}, [r0]`
- uint8x16_t vld1q_lane_u8 (const uint8_t *, uint8x16_t, const int)
  Form of expected instruction(s): `vld1.8 {d0[0]}, [r0]`
- int32x4_t vld1q_lane_s32 (const int32_t *, int32x4_t, const int)
  Form of expected instruction(s): `vld1.32 {d0[0]}, [r0]`
- int16x8_t vld1q_lane_s16 (const int16_t *, int16x8_t, const int)
  Form of expected instruction(s): `vld1.16 {d0[0]}, [r0]`
- int8x16_t vld1q_lane_s8 (const int8_t *, int8x16_t, const int)
  Form of expected instruction(s): `vld1.8 {d0[0]}, [r0]`
- float32x4_t vld1q_lane_f32 (const float32_t *, float32x4_t, const int)
  Form of expected instruction(s): `vld1.32 {d0[0]}, [r0]`
- poly16x8_t vld1q_lane_p16 (const poly16_t *, poly16x8_t, const int)
  Form of expected instruction(s): `vld1.16 {d0[0]}, [r0]`
- poly8x16_t vld1q_lane_p8 (const poly8_t *, poly8x16_t, const int)
  Form of expected instruction(s): `vld1.8 {d0[0]}, [r0]`
- uint64x2_t vld1q_lane_u64 (const uint64_t *, uint64x2_t, const int)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`

- int64x2_t vld1q_lane_s64 (const int64_t *, int64x2_t, const int)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`
- uint32x2_t vld1_dup_u32 (const uint32_t *)
  Form of expected instruction(s): `vld1.32 {d0[]}, [r0]`
- uint16x4_t vld1_dup_u16 (const uint16_t *)
  Form of expected instruction(s): `vld1.16 {d0[]}, [r0]`
- uint8x8_t vld1_dup_u8 (const uint8_t *)
  Form of expected instruction(s): `vld1.8 {d0[]}, [r0]`
- int32x2_t vld1_dup_s32 (const int32_t *)
  Form of expected instruction(s): `vld1.32 {d0[]}, [r0]`
- int16x4_t vld1_dup_s16 (const int16_t *)
  Form of expected instruction(s): `vld1.16 {d0[]}, [r0]`
- int8x8_t vld1_dup_s8 (const int8_t *)
  Form of expected instruction(s): `vld1.8 {d0[]}, [r0]`
- float32x2_t vld1_dup_f32 (const float32_t *)
  Form of expected instruction(s): `vld1.32 {d0[]}, [r0]`
- poly16x4_t vld1_dup_p16 (const poly16_t *)
  Form of expected instruction(s): `vld1.16 {d0[]}, [r0]`
- poly8x8_t vld1_dup_p8 (const poly8_t *)
  Form of expected instruction(s): `vld1.8 {d0[]}, [r0]`
- uint64x1_t vld1_dup_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`
- int64x1_t vld1_dup_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0}, [r0]`
- uint32x4_t vld1q_dup_u32 (const uint32_t *)
  Form of expected instruction(s): `vld1.32 {d0[], d1[]}, [r0]`
- uint16x8_t vld1q_dup_u16 (const uint16_t *)
  Form of expected instruction(s): `vld1.16 {d0[], d1[]}, [r0]`
- uint8x16_t vld1q_dup_u8 (const uint8_t *)
  Form of expected instruction(s): `vld1.8 {d0[], d1[]}, [r0]`
- int32x4_t vld1q_dup_s32 (const int32_t *)
  Form of expected instruction(s): `vld1.32 {d0[], d1[]}, [r0]`
- int16x8_t vld1q_dup_s16 (const int16_t *)
  Form of expected instruction(s): `vld1.16 {d0[], d1[]}, [r0]`
- int8x16_t vld1q_dup_s8 (const int8_t *)
  Form of expected instruction(s): `vld1.8 {d0[], d1[]}, [r0]`
- float32x4_t vld1q_dup_f32 (const float32_t *)
  Form of expected instruction(s): `vld1.32 {d0[], d1[]}, [r0]`
- poly16x8_t vld1q_dup_p16 (const poly16_t *)
  Form of expected instruction(s): `vld1.16 {d0[], d1[]}, [r0]`
- poly8x16_t vld1q_dup_p8 (const poly8_t *)
  Form of expected instruction(s): `vld1.8 {d0[], d1[]}, [r0]`

- uint64x2_t vld1q_dup_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`
- int64x2_t vld1q_dup_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`

6.55.3.69 Element/structure stores, VST1 variants

- void vst1_u32 (uint32_t *, uint32x2_t)
  Form of expected instruction(s): `vst1.32 {d0}, [r0]`
- void vst1_u16 (uint16_t *, uint16x4_t)
  Form of expected instruction(s): `vst1.16 {d0}, [r0]`
- void vst1_u8 (uint8_t *, uint8x8_t)
  Form of expected instruction(s): `vst1.8 {d0}, [r0]`
- void vst1_s32 (int32_t *, int32x2_t)
  Form of expected instruction(s): `vst1.32 {d0}, [r0]`
- void vst1_s16 (int16_t *, int16x4_t)
  Form of expected instruction(s): `vst1.16 {d0}, [r0]`
- void vst1_s8 (int8_t *, int8x8_t)
  Form of expected instruction(s): `vst1.8 {d0}, [r0]`
- void vst1_u64 (uint64_t *, uint64x1_t)
  Form of expected instruction(s): `vst1.64 {d0}, [r0]`
- void vst1_s64 (int64_t *, int64x1_t)
  Form of expected instruction(s): `vst1.64 {d0}, [r0]`
- void vst1_f32 (float32_t *, float32x2_t)
  Form of expected instruction(s): `vst1.32 {d0}, [r0]`
- void vst1_p16 (poly16_t *, poly16x4_t)
  Form of expected instruction(s): `vst1.16 {d0}, [r0]`
- void vst1_p8 (poly8_t *, poly8x8_t)
  Form of expected instruction(s): `vst1.8 {d0}, [r0]`
- void vst1q_u32 (uint32_t *, uint32x4_t)
  Form of expected instruction(s): `vst1.32 {d0, d1}, [r0]`
- void vst1q_u16 (uint16_t *, uint16x8_t)
  Form of expected instruction(s): `vst1.16 {d0, d1}, [r0]`
- void vst1q_u8 (uint8_t *, uint8x16_t)
  Form of expected instruction(s): `vst1.8 {d0, d1}, [r0]`
- void vst1q_s32 (int32_t *, int32x4_t)
  Form of expected instruction(s): `vst1.32 {d0, d1}, [r0]`
- void vst1q_s16 (int16_t *, int16x8_t)
  Form of expected instruction(s): `vst1.16 {d0, d1}, [r0]`
- void vst1q_s8 (int8_t *, int8x16_t)
  Form of expected instruction(s): `vst1.8 {d0, d1}, [r0]`
- void vst1q_u64 (uint64_t *, uint64x2_t)
  Form of expected instruction(s): `vst1.64 {d0, d1}, [r0]`

- void vst1q_s64 (int64_t *, int64x2_t)
  Form of expected instruction(s): `vst1.64 {d0, d1}, [r0]`
- void vst1q_f32 (float32_t *, float32x4_t)
  Form of expected instruction(s): `vst1.32 {d0, d1}, [r0]`
- void vst1q_p16 (poly16_t *, poly16x8_t)
  Form of expected instruction(s): `vst1.16 {d0, d1}, [r0]`
- void vst1q_p8 (poly8_t *, poly8x16_t)
  Form of expected instruction(s): `vst1.8 {d0, d1}, [r0]`
- void vst1_lane_u32 (uint32_t *, uint32x2_t, const int)
  Form of expected instruction(s): `vst1.32 {d0[0]}, [r0]`
- void vst1_lane_u16 (uint16_t *, uint16x4_t, const int)
  Form of expected instruction(s): `vst1.16 {d0[0]}, [r0]`
- void vst1_lane_u8 (uint8_t *, uint8x8_t, const int)
  Form of expected instruction(s): `vst1.8 {d0[0]}, [r0]`
- void vst1_lane_s32 (int32_t *, int32x2_t, const int)
  Form of expected instruction(s): `vst1.32 {d0[0]}, [r0]`
- void vst1_lane_s16 (int16_t *, int16x4_t, const int)
  Form of expected instruction(s): `vst1.16 {d0[0]}, [r0]`
- void vst1_lane_s8 (int8_t *, int8x8_t, const int)
  Form of expected instruction(s): `vst1.8 {d0[0]}, [r0]`
- void vst1_lane_f32 (float32_t *, float32x2_t, const int)
  Form of expected instruction(s): `vst1.32 {d0[0]}, [r0]`
- void vst1_lane_p16 (poly16_t *, poly16x4_t, const int)
  Form of expected instruction(s): `vst1.16 {d0[0]}, [r0]`
- void vst1_lane_p8 (poly8_t *, poly8x8_t, const int)
  Form of expected instruction(s): `vst1.8 {d0[0]}, [r0]`
- void vst1_lane_s64 (int64_t *, int64x1_t, const int)
  Form of expected instruction(s): `vst1.64 {d0}, [r0]`
- void vst1_lane_u64 (uint64_t *, uint64x1_t, const int)
  Form of expected instruction(s): `vst1.64 {d0}, [r0]`
- void vst1q_lane_u32 (uint32_t *, uint32x4_t, const int)
  Form of expected instruction(s): `vst1.32 {d0[0]}, [r0]`
- void vst1q_lane_u16 (uint16_t *, uint16x8_t, const int)
  Form of expected instruction(s): `vst1.16 {d0[0]}, [r0]`
- void vst1q_lane_u8 (uint8_t *, uint8x16_t, const int)
  Form of expected instruction(s): `vst1.8 {d0[0]}, [r0]`
- void vst1q_lane_s32 (int32_t *, int32x4_t, const int)
  Form of expected instruction(s): `vst1.32 {d0[0]}, [r0]`
- void vst1q_lane_s16 (int16_t *, int16x8_t, const int)
  Form of expected instruction(s): `vst1.16 {d0[0]}, [r0]`
- void vst1q_lane_s8 (int8_t *, int8x16_t, const int)
  Form of expected instruction(s): `vst1.8 {d0[0]}, [r0]`

- void vst1q_lane_f32 (float32_t *, float32x4_t, const int)
  Form of expected instruction(s): `vst1.32 {d0[0]}, [r0]`
- void vst1q_lane_p16 (poly16_t *, poly16x8_t, const int)
  Form of expected instruction(s): `vst1.16 {d0[0]}, [r0]`
- void vst1q_lane_p8 (poly8_t *, poly8x16_t, const int)
  Form of expected instruction(s): `vst1.8 {d0[0]}, [r0]`
- void vst1q_lane_s64 (int64_t *, int64x2_t, const int)
  Form of expected instruction(s): `vst1.64 {d0}, [r0]`
- void vst1q_lane_u64 (uint64_t *, uint64x2_t, const int)
  Form of expected instruction(s): `vst1.64 {d0}, [r0]`

6.55.3.70 Element/structure loads, VLD2 variants

- uint32x2x2_t vld2_u32 (const uint32_t *)
  Form of expected instruction(s): `vld2.32 {d0, d1}, [r0]`
- uint16x4x2_t vld2_u16 (const uint16_t *)
  Form of expected instruction(s): `vld2.16 {d0, d1}, [r0]`
- uint8x8x2_t vld2_u8 (const uint8_t *)
  Form of expected instruction(s): `vld2.8 {d0, d1}, [r0]`
- int32x2x2_t vld2_s32 (const int32_t *)
  Form of expected instruction(s): `vld2.32 {d0, d1}, [r0]`
- int16x4x2_t vld2_s16 (const int16_t *)
  Form of expected instruction(s): `vld2.16 {d0, d1}, [r0]`
- int8x8x2_t vld2_s8 (const int8_t *)
  Form of expected instruction(s): `vld2.8 {d0, d1}, [r0]`
- float32x2x2_t vld2_f32 (const float32_t *)
  Form of expected instruction(s): `vld2.32 {d0, d1}, [r0]`
- poly16x4x2_t vld2_p16 (const poly16_t *)
  Form of expected instruction(s): `vld2.16 {d0, d1}, [r0]`
- poly8x8x2_t vld2_p8 (const poly8_t *)
  Form of expected instruction(s): `vld2.8 {d0, d1}, [r0]`
- uint64x1x2_t vld2_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`
- int64x1x2_t vld2_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`
- uint32x4x2_t vld2q_u32 (const uint32_t *)
  Form of expected instruction(s): `vld2.32 {d0, d1}, [r0]`
- uint16x8x2_t vld2q_u16 (const uint16_t *)
  Form of expected instruction(s): `vld2.16 {d0, d1}, [r0]`
- uint8x16x2_t vld2q_u8 (const uint8_t *)
  Form of expected instruction(s): `vld2.8 {d0, d1}, [r0]`
- int32x4x2_t vld2q_s32 (const int32_t *)
  Form of expected instruction(s): `vld2.32 {d0, d1}, [r0]`

- int16x8x2_t vld2q_s16 (const int16_t *)
  Form of expected instruction(s): `vld2.16 {d0, d1}, [r0]`
- int8x16x2_t vld2q_s8 (const int8_t *)
  Form of expected instruction(s): `vld2.8 {d0, d1}, [r0]`
- float32x4x2_t vld2q_f32 (const float32_t *)
  Form of expected instruction(s): `vld2.32 {d0, d1}, [r0]`
- poly16x8x2_t vld2q_p16 (const poly16_t *)
  Form of expected instruction(s): `vld2.16 {d0, d1}, [r0]`
- poly8x16x2_t vld2q_p8 (const poly8_t *)
  Form of expected instruction(s): `vld2.8 {d0, d1}, [r0]`
- uint32x2x2_t vld2_lane_u32 (const uint32_t *, uint32x2x2_t, const int)
  Form of expected instruction(s): `vld2.32 {d0[0], d1[0]}, [r0]`
- uint16x4x2_t vld2_lane_u16 (const uint16_t *, uint16x4x2_t, const int)
  Form of expected instruction(s): `vld2.16 {d0[0], d1[0]}, [r0]`
- uint8x8x2_t vld2_lane_u8 (const uint8_t *, uint8x8x2_t, const int)
  Form of expected instruction(s): `vld2.8 {d0[0], d1[0]}, [r0]`
- int32x2x2_t vld2_lane_s32 (const int32_t *, int32x2x2_t, const int)
  Form of expected instruction(s): `vld2.32 {d0[0], d1[0]}, [r0]`
- int16x4x2_t vld2_lane_s16 (const int16_t *, int16x4x2_t, const int)
  Form of expected instruction(s): `vld2.16 {d0[0], d1[0]}, [r0]`
- int8x8x2_t vld2_lane_s8 (const int8_t *, int8x8x2_t, const int)
  Form of expected instruction(s): `vld2.8 {d0[0], d1[0]}, [r0]`
- float32x2x2_t vld2_lane_f32 (const float32_t *, float32x2x2_t, const int)
  Form of expected instruction(s): `vld2.32 {d0[0], d1[0]}, [r0]`
- poly16x4x2_t vld2_lane_p16 (const poly16_t *, poly16x4x2_t, const int)
  Form of expected instruction(s): `vld2.16 {d0[0], d1[0]}, [r0]`
- poly8x8x2_t vld2_lane_p8 (const poly8_t *, poly8x8x2_t, const int)
  Form of expected instruction(s): `vld2.8 {d0[0], d1[0]}, [r0]`
- int32x4x2_t vld2q_lane_s32 (const int32_t *, int32x4x2_t, const int)
  Form of expected instruction(s): `vld2.32 {d0[0], d1[0]}, [r0]`
- int16x8x2_t vld2q_lane_s16 (const int16_t *, int16x8x2_t, const int)
  Form of expected instruction(s): `vld2.16 {d0[0], d1[0]}, [r0]`
- uint32x4x2_t vld2q_lane_u32 (const uint32_t *, uint32x4x2_t, const int)
  Form of expected instruction(s): `vld2.32 {d0[0], d1[0]}, [r0]`
- uint16x8x2_t vld2q_lane_u16 (const uint16_t *, uint16x8x2_t, const int)
  Form of expected instruction(s): `vld2.16 {d0[0], d1[0]}, [r0]`
- float32x4x2_t vld2q_lane_f32 (const float32_t *, float32x4x2_t, const int)
  Form of expected instruction(s): `vld2.32 {d0[0], d1[0]}, [r0]`
- poly16x8x2_t vld2q_lane_p16 (const poly16_t *, poly16x8x2_t, const int)
  Form of expected instruction(s): `vld2.16 {d0[0], d1[0]}, [r0]`
- uint32x2x2_t vld2_dup_u32 (const uint32_t *)
  Form of expected instruction(s): `vld2.32 {d0[], d1[]}, [r0]`

- uint16x4x2_t vld2_dup_u16 (const uint16_t *)
  Form of expected instruction(s): `vld2.16 {d0[], d1[]}, [r0]`
- uint8x8x2_t vld2_dup_u8 (const uint8_t *)
  Form of expected instruction(s): `vld2.8 {d0[], d1[]}, [r0]`
- int32x2x2_t vld2_dup_s32 (const int32_t *)
  Form of expected instruction(s): `vld2.32 {d0[], d1[]}, [r0]`
- int16x4x2_t vld2_dup_s16 (const int16_t *)
  Form of expected instruction(s): `vld2.16 {d0[], d1[]}, [r0]`
- int8x8x2_t vld2_dup_s8 (const int8_t *)
  Form of expected instruction(s): `vld2.8 {d0[], d1[]}, [r0]`
- float32x2x2_t vld2_dup_f32 (const float32_t *)
  Form of expected instruction(s): `vld2.32 {d0[], d1[]}, [r0]`
- poly16x4x2_t vld2_dup_p16 (const poly16_t *)
  Form of expected instruction(s): `vld2.16 {d0[], d1[]}, [r0]`
- poly8x8x2_t vld2_dup_p8 (const poly8_t *)
  Form of expected instruction(s): `vld2.8 {d0[], d1[]}, [r0]`
- uint64x1x2_t vld2_dup_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`
- int64x1x2_t vld2_dup_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1}, [r0]`

6.55.3.71 Element/structure stores, VST2 variants

- void vst2_u32 (uint32_t *, uint32x2x2_t)
  Form of expected instruction(s): `vst2.32 {d0, d1}, [r0]`
- void vst2_u16 (uint16_t *, uint16x4x2_t)
  Form of expected instruction(s): `vst2.16 {d0, d1}, [r0]`
- void vst2_u8 (uint8_t *, uint8x8x2_t)
  Form of expected instruction(s): `vst2.8 {d0, d1}, [r0]`
- void vst2_s32 (int32_t *, int32x2x2_t)
  Form of expected instruction(s): `vst2.32 {d0, d1}, [r0]`
- void vst2_s16 (int16_t *, int16x4x2_t)
  Form of expected instruction(s): `vst2.16 {d0, d1}, [r0]`
- void vst2_s8 (int8_t *, int8x8x2_t)
  Form of expected instruction(s): `vst2.8 {d0, d1}, [r0]`
- void vst2_f32 (float32_t *, float32x2x2_t)
  Form of expected instruction(s): `vst2.32 {d0, d1}, [r0]`
- void vst2_p16 (poly16_t *, poly16x4x2_t)
  Form of expected instruction(s): `vst2.16 {d0, d1}, [r0]`
- void vst2_p8 (poly8_t *, poly8x8x2_t)
  Form of expected instruction(s): `vst2.8 {d0, d1}, [r0]`
- void vst2_u64 (uint64_t *, uint64x1x2_t)
  Form of expected instruction(s): `vst1.64 {d0, d1}, [r0]`

- void vst2_s64 (int64_t *, int64x1x2_t)
  Form of expected instruction(s): `vst1.64 {d0, d1}, [r0]`
- void vst2q_u32 (uint32_t *, uint32x4x2_t)
  Form of expected instruction(s): `vst2.32 {d0, d1}, [r0]`
- void vst2q_u16 (uint16_t *, uint16x8x2_t)
  Form of expected instruction(s): `vst2.16 {d0, d1}, [r0]`
- void vst2q_u8 (uint8_t *, uint8x16x2_t)
  Form of expected instruction(s): `vst2.8 {d0, d1}, [r0]`
- void vst2q_s32 (int32_t *, int32x4x2_t)
  Form of expected instruction(s): `vst2.32 {d0, d1}, [r0]`
- void vst2q_s16 (int16_t *, int16x8x2_t)
  Form of expected instruction(s): `vst2.16 {d0, d1}, [r0]`
- void vst2q_s8 (int8_t *, int8x16x2_t)
  Form of expected instruction(s): `vst2.8 {d0, d1}, [r0]`
- void vst2q_f32 (float32_t *, float32x4x2_t)
  Form of expected instruction(s): `vst2.32 {d0, d1}, [r0]`
- void vst2q_p16 (poly16_t *, poly16x8x2_t)
  Form of expected instruction(s): `vst2.16 {d0, d1}, [r0]`
- void vst2q_p8 (poly8_t *, poly8x16x2_t)
  Form of expected instruction(s): `vst2.8 {d0, d1}, [r0]`
- void vst2_lane_u32 (uint32_t *, uint32x2x2_t, const int)
  Form of expected instruction(s): `vst2.32 {d0[0], d1[0]}, [r0]`
- void vst2_lane_u16 (uint16_t *, uint16x4x2_t, const int)
  Form of expected instruction(s): `vst2.16 {d0[0], d1[0]}, [r0]`
- void vst2_lane_u8 (uint8_t *, uint8x8x2_t, const int)
  Form of expected instruction(s): `vst2.8 {d0[0], d1[0]}, [r0]`
- void vst2_lane_s32 (int32_t *, int32x2x2_t, const int)
  Form of expected instruction(s): `vst2.32 {d0[0], d1[0]}, [r0]`
- void vst2_lane_s16 (int16_t *, int16x4x2_t, const int)
  Form of expected instruction(s): `vst2.16 {d0[0], d1[0]}, [r0]`
- void vst2_lane_s8 (int8_t *, int8x8x2_t, const int)
  Form of expected instruction(s): `vst2.8 {d0[0], d1[0]}, [r0]`
- void vst2_lane_f32 (float32_t *, float32x2x2_t, const int)
  Form of expected instruction(s): `vst2.32 {d0[0], d1[0]}, [r0]`
- void vst2_lane_p16 (poly16_t *, poly16x4x2_t, const int)
  Form of expected instruction(s): `vst2.16 {d0[0], d1[0]}, [r0]`
- void vst2_lane_p8 (poly8_t *, poly8x8x2_t, const int)
  Form of expected instruction(s): `vst2.8 {d0[0], d1[0]}, [r0]`
- void vst2q_lane_s32 (int32_t *, int32x4x2_t, const int)
  Form of expected instruction(s): `vst2.32 {d0[0], d1[0]}, [r0]`
- void vst2q_lane_s16 (int16_t *, int16x8x2_t, const int)
  Form of expected instruction(s): `vst2.16 {d0[0], d1[0]}, [r0]`

- void vst2q_lane_u32 (uint32_t *, uint32x4x2_t, const int)
  Form of expected instruction(s): `vst2.32 {d0[0], d1[0]}, [r0]`
- void vst2q_lane_u16 (uint16_t *, uint16x8x2_t, const int)
  Form of expected instruction(s): `vst2.16 {d0[0], d1[0]}, [r0]`
- void vst2q_lane_f32 (float32_t *, float32x4x2_t, const int)
  Form of expected instruction(s): `vst2.32 {d0[0], d1[0]}, [r0]`
- void vst2q_lane_p16 (poly16_t *, poly16x8x2_t, const int)
  Form of expected instruction(s): `vst2.16 {d0[0], d1[0]}, [r0]`

6.55.3.72 Element/structure loads, VLD3 variants

- uint32x2x3_t vld3_u32 (const uint32_t *)
  Form of expected instruction(s): `vld3.32 {d0, d1, d2}, [r0]`
- uint16x4x3_t vld3_u16 (const uint16_t *)
  Form of expected instruction(s): `vld3.16 {d0, d1, d2}, [r0]`
- uint8x8x3_t vld3_u8 (const uint8_t *)
  Form of expected instruction(s): `vld3.8 {d0, d1, d2}, [r0]`
- int32x2x3_t vld3_s32 (const int32_t *)
  Form of expected instruction(s): `vld3.32 {d0, d1, d2}, [r0]`
- int16x4x3_t vld3_s16 (const int16_t *)
  Form of expected instruction(s): `vld3.16 {d0, d1, d2}, [r0]`
- int8x8x3_t vld3_s8 (const int8_t *)
  Form of expected instruction(s): `vld3.8 {d0, d1, d2}, [r0]`
- float32x2x3_t vld3_f32 (const float32_t *)
  Form of expected instruction(s): `vld3.32 {d0, d1, d2}, [r0]`
- poly16x4x3_t vld3_p16 (const poly16_t *)
  Form of expected instruction(s): `vld3.16 {d0, d1, d2}, [r0]`
- poly8x8x3_t vld3_p8 (const poly8_t *)
  Form of expected instruction(s): `vld3.8 {d0, d1, d2}, [r0]`
- uint64x1x3_t vld3_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2}, [r0]`
- int64x1x3_t vld3_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2}, [r0]`
- uint32x4x3_t vld3q_u32 (const uint32_t *)
  Form of expected instruction(s): `vld3.32 {d0, d1, d2}, [r0]`
- uint16x8x3_t vld3q_u16 (const uint16_t *)
  Form of expected instruction(s): `vld3.16 {d0, d1, d2}, [r0]`
- uint8x16x3_t vld3q_u8 (const uint8_t *)
  Form of expected instruction(s): `vld3.8 {d0, d1, d2}, [r0]`
- int32x4x3_t vld3q_s32 (const int32_t *)
  Form of expected instruction(s): `vld3.32 {d0, d1, d2}, [r0]`
- int16x8x3_t vld3q_s16 (const int16_t *)
  Form of expected instruction(s): `vld3.16 {d0, d1, d2}, [r0]`

- int8x16x3_t vld3q_s8 (const int8_t *)
  Form of expected instruction(s): `vld3.8 {d0, d1, d2}, [r0]`
- float32x4x3_t vld3q_f32 (const float32_t *)
  Form of expected instruction(s): `vld3.32 {d0, d1, d2}, [r0]`
- poly16x8x3_t vld3q_p16 (const poly16_t *)
  Form of expected instruction(s): `vld3.16 {d0, d1, d2}, [r0]`
- poly8x16x3_t vld3q_p8 (const poly8_t *)
  Form of expected instruction(s): `vld3.8 {d0, d1, d2}, [r0]`
- uint32x2x3_t vld3_lane_u32 (const uint32_t *, uint32x2x3_t, const int)
  Form of expected instruction(s): `vld3.32 {d0[0], d1[0], d2[0]}, [r0]`
- uint16x4x3_t vld3_lane_u16 (const uint16_t *, uint16x4x3_t, const int)
  Form of expected instruction(s): `vld3.16 {d0[0], d1[0], d2[0]}, [r0]`
- uint8x8x3_t vld3_lane_u8 (const uint8_t *, uint8x8x3_t, const int)
  Form of expected instruction(s): `vld3.8 {d0[0], d1[0], d2[0]}, [r0]`
- int32x2x3_t vld3_lane_s32 (const int32_t *, int32x2x3_t, const int)
  Form of expected instruction(s): `vld3.32 {d0[0], d1[0], d2[0]}, [r0]`
- int16x4x3_t vld3_lane_s16 (const int16_t *, int16x4x3_t, const int)
  Form of expected instruction(s): `vld3.16 {d0[0], d1[0], d2[0]}, [r0]`
- int8x8x3_t vld3_lane_s8 (const int8_t *, int8x8x3_t, const int)
  Form of expected instruction(s): `vld3.8 {d0[0], d1[0], d2[0]}, [r0]`
- float32x2x3_t vld3_lane_f32 (const float32_t *, float32x2x3_t, const int)
  Form of expected instruction(s): `vld3.32 {d0[0], d1[0], d2[0]}, [r0]`
- poly16x4x3_t vld3_lane_p16 (const poly16_t *, poly16x4x3_t, const int)
  Form of expected instruction(s): `vld3.16 {d0[0], d1[0], d2[0]}, [r0]`
- poly8x8x3_t vld3_lane_p8 (const poly8_t *, poly8x8x3_t, const int)
  Form of expected instruction(s): `vld3.8 {d0[0], d1[0], d2[0]}, [r0]`
- int32x4x3_t vld3q_lane_s32 (const int32_t *, int32x4x3_t, const int)
  Form of expected instruction(s): `vld3.32 {d0[0], d1[0], d2[0]}, [r0]`
- int16x8x3_t vld3q_lane_s16 (const int16_t *, int16x8x3_t, const int)
  Form of expected instruction(s): `vld3.16 {d0[0], d1[0], d2[0]}, [r0]`
- uint32x4x3_t vld3q_lane_u32 (const uint32_t *, uint32x4x3_t, const int)
  Form of expected instruction(s): `vld3.32 {d0[0], d1[0], d2[0]}, [r0]`
- uint16x8x3_t vld3q_lane_u16 (const uint16_t *, uint16x8x3_t, const int)
  Form of expected instruction(s): `vld3.16 {d0[0], d1[0], d2[0]}, [r0]`
- float32x4x3_t vld3q_lane_f32 (const float32_t *, float32x4x3_t, const int)
  Form of expected instruction(s): `vld3.32 {d0[0], d1[0], d2[0]}, [r0]`
- poly16x8x3_t vld3q_lane_p16 (const poly16_t *, poly16x8x3_t, const int)
  Form of expected instruction(s): `vld3.16 {d0[0], d1[0], d2[0]}, [r0]`
- uint32x2x3_t vld3_dup_u32 (const uint32_t *)
  Form of expected instruction(s): `vld3.32 {d0[], d1[], d2[]}, [r0]`
- uint16x4x3_t vld3_dup_u16 (const uint16_t *)
  Form of expected instruction(s): `vld3.16 {d0[], d1[], d2[]}, [r0]`

- uint8x8x3_t vld3_dup_u8 (const uint8_t *)
  Form of expected instruction(s): `vld3.8 {d0[], d1[], d2[]}, [r0]`
- int32x2x3_t vld3_dup_s32 (const int32_t *)
  Form of expected instruction(s): `vld3.32 {d0[], d1[], d2[]}, [r0]`
- int16x4x3_t vld3_dup_s16 (const int16_t *)
  Form of expected instruction(s): `vld3.16 {d0[], d1[], d2[]}, [r0]`
- int8x8x3_t vld3_dup_s8 (const int8_t *)
  Form of expected instruction(s): `vld3.8 {d0[], d1[], d2[]}, [r0]`
- float32x2x3_t vld3_dup_f32 (const float32_t *)
  Form of expected instruction(s): `vld3.32 {d0[], d1[], d2[]}, [r0]`
- poly16x4x3_t vld3_dup_p16 (const poly16_t *)
  Form of expected instruction(s): `vld3.16 {d0[], d1[], d2[]}, [r0]`
- poly8x8x3_t vld3_dup_p8 (const poly8_t *)
  Form of expected instruction(s): `vld3.8 {d0[], d1[], d2[]}, [r0]`
- uint64x1x3_t vld3_dup_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2}, [r0]`
- int64x1x3_t vld3_dup_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2}, [r0]`

6.55.3.73 Element/structure stores, VST3 variants

- void vst3_u32 (uint32_t *, uint32x2x3_t)
  Form of expected instruction(s): `vst3.32 {d0, d1, d2, d3}, [r0]`
- void vst3_u16 (uint16_t *, uint16x4x3_t)
  Form of expected instruction(s): `vst3.16 {d0, d1, d2, d3}, [r0]`
- void vst3_u8 (uint8_t *, uint8x8x3_t)
  Form of expected instruction(s): `vst3.8 {d0, d1, d2, d3}, [r0]`
- void vst3_s32 (int32_t *, int32x2x3_t)
  Form of expected instruction(s): `vst3.32 {d0, d1, d2, d3}, [r0]`
- void vst3_s16 (int16_t *, int16x4x3_t)
  Form of expected instruction(s): `vst3.16 {d0, d1, d2, d3}, [r0]`
- void vst3_s8 (int8_t *, int8x8x3_t)
  Form of expected instruction(s): `vst3.8 {d0, d1, d2, d3}, [r0]`
- void vst3_f32 (float32_t *, float32x2x3_t)
  Form of expected instruction(s): `vst3.32 {d0, d1, d2, d3}, [r0]`
- void vst3_p16 (poly16_t *, poly16x4x3_t)
  Form of expected instruction(s): `vst3.16 {d0, d1, d2, d3}, [r0]`
- void vst3_p8 (poly8_t *, poly8x8x3_t)
  Form of expected instruction(s): `vst3.8 {d0, d1, d2, d3}, [r0]`
- void vst3_u64 (uint64_t *, uint64x1x3_t)
  Form of expected instruction(s): `vst1.64 {d0, d1, d2, d3}, [r0]`
- void vst3_s64 (int64_t *, int64x1x3_t)
  Form of expected instruction(s): `vst1.64 {d0, d1, d2, d3}, [r0]`

- void vst3q_u32 (uint32_t *, uint32x4x3_t)
  Form of expected instruction(s): `vst3.32 {d0, d1, d2}, [r0]`
- void vst3q_u16 (uint16_t *, uint16x8x3_t)
  Form of expected instruction(s): `vst3.16 {d0, d1, d2}, [r0]`
- void vst3q_u8 (uint8_t *, uint8x16x3_t)
  Form of expected instruction(s): `vst3.8 {d0, d1, d2}, [r0]`
- void vst3q_s32 (int32_t *, int32x4x3_t)
  Form of expected instruction(s): `vst3.32 {d0, d1, d2}, [r0]`
- void vst3q_s16 (int16_t *, int16x8x3_t)
  Form of expected instruction(s): `vst3.16 {d0, d1, d2}, [r0]`
- void vst3q_s8 (int8_t *, int8x16x3_t)
  Form of expected instruction(s): `vst3.8 {d0, d1, d2}, [r0]`
- void vst3q_f32 (float32_t *, float32x4x3_t)
  Form of expected instruction(s): `vst3.32 {d0, d1, d2}, [r0]`
- void vst3q_p16 (poly16_t *, poly16x8x3_t)
  Form of expected instruction(s): `vst3.16 {d0, d1, d2}, [r0]`
- void vst3q_p8 (poly8_t *, poly8x16x3_t)
  Form of expected instruction(s): `vst3.8 {d0, d1, d2}, [r0]`
- void vst3_lane_u32 (uint32_t *, uint32x2x3_t, const int)
  Form of expected instruction(s): `vst3.32 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_u16 (uint16_t *, uint16x4x3_t, const int)
  Form of expected instruction(s): `vst3.16 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_u8 (uint8_t *, uint8x8x3_t, const int)
  Form of expected instruction(s): `vst3.8 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_s32 (int32_t *, int32x2x3_t, const int)
  Form of expected instruction(s): `vst3.32 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_s16 (int16_t *, int16x4x3_t, const int)
  Form of expected instruction(s): `vst3.16 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_s8 (int8_t *, int8x8x3_t, const int)
  Form of expected instruction(s): `vst3.8 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_f32 (float32_t *, float32x2x3_t, const int)
  Form of expected instruction(s): `vst3.32 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_p16 (poly16_t *, poly16x4x3_t, const int)
  Form of expected instruction(s): `vst3.16 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3_lane_p8 (poly8_t *, poly8x8x3_t, const int)
  Form of expected instruction(s): `vst3.8 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3q_lane_s32 (int32_t *, int32x4x3_t, const int)
  Form of expected instruction(s): `vst3.32 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3q_lane_s16 (int16_t *, int16x8x3_t, const int)
  Form of expected instruction(s): `vst3.16 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3q_lane_u32 (uint32_t *, uint32x4x3_t, const int)
  Form of expected instruction(s): `vst3.32 {d0[0], d1[0], d2[0]}, [r0]`

- void vst3q_lane_u16 (uint16_t *, uint16x8x3_t, const int)
  Form of expected instruction(s): `vst3.16 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3q_lane_f32 (float32_t *, float32x4x3_t, const int)
  Form of expected instruction(s): `vst3.32 {d0[0], d1[0], d2[0]}, [r0]`
- void vst3q_lane_p16 (poly16_t *, poly16x8x3_t, const int)
  Form of expected instruction(s): `vst3.16 {d0[0], d1[0], d2[0]}, [r0]`

6.55.3.74 Element/structure loads, VLD4 variants

- uint32x2x4_t vld4_u32 (const uint32_t *)
  Form of expected instruction(s): `vld4.32 {d0, d1, d2, d3}, [r0]`
- uint16x4x4_t vld4_u16 (const uint16_t *)
  Form of expected instruction(s): `vld4.16 {d0, d1, d2, d3}, [r0]`
- uint8x8x4_t vld4_u8 (const uint8_t *)
  Form of expected instruction(s): `vld4.8 {d0, d1, d2, d3}, [r0]`
- int32x2x4_t vld4_s32 (const int32_t *)
  Form of expected instruction(s): `vld4.32 {d0, d1, d2, d3}, [r0]`
- int16x4x4_t vld4_s16 (const int16_t *)
  Form of expected instruction(s): `vld4.16 {d0, d1, d2, d3}, [r0]`
- int8x8x4_t vld4_s8 (const int8_t *)
  Form of expected instruction(s): `vld4.8 {d0, d1, d2, d3}, [r0]`
- float32x2x4_t vld4_f32 (const float32_t *)
  Form of expected instruction(s): `vld4.32 {d0, d1, d2, d3}, [r0]`
- poly16x4x4_t vld4_p16 (const poly16_t *)
  Form of expected instruction(s): `vld4.16 {d0, d1, d2, d3}, [r0]`
- poly8x8x4_t vld4_p8 (const poly8_t *)
  Form of expected instruction(s): `vld4.8 {d0, d1, d2, d3}, [r0]`
- uint64x1x4_t vld4_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2, d3}, [r0]`
- int64x1x4_t vld4_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2, d3}, [r0]`
- uint32x4x4_t vld4q_u32 (const uint32_t *)
  Form of expected instruction(s): `vld4.32 {d0, d1, d2, d3}, [r0]`
- uint16x8x4_t vld4q_u16 (const uint16_t *)
  Form of expected instruction(s): `vld4.16 {d0, d1, d2, d3}, [r0]`
- uint8x16x4_t vld4q_u8 (const uint8_t *)
  Form of expected instruction(s): `vld4.8 {d0, d1, d2, d3}, [r0]`
- int32x4x4_t vld4q_s32 (const int32_t *)
  Form of expected instruction(s): `vld4.32 {d0, d1, d2, d3}, [r0]`
- int16x8x4_t vld4q_s16 (const int16_t *)
  Form of expected instruction(s): `vld4.16 {d0, d1, d2, d3}, [r0]`
- int8x16x4_t vld4q_s8 (const int8_t *)
  Form of expected instruction(s): `vld4.8 {d0, d1, d2, d3}, [r0]`

- float32x4x4_t vld4q_f32 (const float32_t *)
  Form of expected instruction(s): `vld4.32 {d0, d1, d2, d3}, [r0]`
- poly16x8x4_t vld4q_p16 (const poly16_t *)
  Form of expected instruction(s): `vld4.16 {d0, d1, d2, d3}, [r0]`
- poly8x16x4_t vld4q_p8 (const poly8_t *)
  Form of expected instruction(s): `vld4.8 {d0, d1, d2, d3}, [r0]`
- uint32x2x4_t vld4_lane_u32 (const uint32_t *, uint32x2x4_t, const int)
  Form of expected instruction(s): `vld4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- uint16x4x4_t vld4_lane_u16 (const uint16_t *, uint16x4x4_t, const int)
  Form of expected instruction(s): `vld4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- uint8x8x4_t vld4_lane_u8 (const uint8_t *, uint8x8x4_t, const int)
  Form of expected instruction(s): `vld4.8 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- int32x2x4_t vld4_lane_s32 (const int32_t *, int32x2x4_t, const int)
  Form of expected instruction(s): `vld4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- int16x4x4_t vld4_lane_s16 (const int16_t *, int16x4x4_t, const int)
  Form of expected instruction(s): `vld4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- int8x8x4_t vld4_lane_s8 (const int8_t *, int8x8x4_t, const int)
  Form of expected instruction(s): `vld4.8 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- float32x2x4_t vld4_lane_f32 (const float32_t *, float32x2x4_t, const int)
  Form of expected instruction(s): `vld4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- poly16x4x4_t vld4_lane_p16 (const poly16_t *, poly16x4x4_t, const int)
  Form of expected instruction(s): `vld4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- poly8x8x4_t vld4_lane_p8 (const poly8_t *, poly8x8x4_t, const int)
  Form of expected instruction(s): `vld4.8 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- int32x4x4_t vld4q_lane_s32 (const int32_t *, int32x4x4_t, const int)
  Form of expected instruction(s): `vld4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- int16x8x4_t vld4q_lane_s16 (const int16_t *, int16x8x4_t, const int)
  Form of expected instruction(s): `vld4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- uint32x4x4_t vld4q_lane_u32 (const uint32_t *, uint32x4x4_t, const int)
  Form of expected instruction(s): `vld4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- uint16x8x4_t vld4q_lane_u16 (const uint16_t *, uint16x8x4_t, const int)
  Form of expected instruction(s): `vld4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- float32x4x4_t vld4q_lane_f32 (const float32_t *, float32x4x4_t, const int)
  Form of expected instruction(s): `vld4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- poly16x8x4_t vld4q_lane_p16 (const poly16_t *, poly16x8x4_t, const int)
  Form of expected instruction(s): `vld4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- uint32x2x4_t vld4_dup_u32 (const uint32_t *)
  Form of expected instruction(s): `vld4.32 {d0[], d1[], d2[], d3[]}, [r0]`
- uint16x4x4_t vld4_dup_u16 (const uint16_t *)
  Form of expected instruction(s): `vld4.16 {d0[], d1[], d2[], d3[]}, [r0]`
- uint8x8x4_t vld4_dup_u8 (const uint8_t *)
  Form of expected instruction(s): `vld4.8 {d0[], d1[], d2[], d3[]}, [r0]`

- int32x2x4_t vld4_dup_s32 (const int32_t *)
  Form of expected instruction(s): `vld4.32 {d0[], d1[], d2[], d3[]}, [r0]`
- int16x4x4_t vld4_dup_s16 (const int16_t *)
  Form of expected instruction(s): `vld4.16 {d0[], d1[], d2[], d3[]}, [r0]`
- int8x8x4_t vld4_dup_s8 (const int8_t *)
  Form of expected instruction(s): `vld4.8 {d0[], d1[], d2[], d3[]}, [r0]`
- float32x2x4_t vld4_dup_f32 (const float32_t *)
  Form of expected instruction(s): `vld4.32 {d0[], d1[], d2[], d3[]}, [r0]`
- poly16x4x4_t vld4_dup_p16 (const poly16_t *)
  Form of expected instruction(s): `vld4.16 {d0[], d1[], d2[], d3[]}, [r0]`
- poly8x8x4_t vld4_dup_p8 (const poly8_t *)
  Form of expected instruction(s): `vld4.8 {d0[], d1[], d2[], d3[]}, [r0]`
- uint64x1x4_t vld4_dup_u64 (const uint64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2, d3}, [r0]`
- int64x1x4_t vld4_dup_s64 (const int64_t *)
  Form of expected instruction(s): `vld1.64 {d0, d1, d2, d3}, [r0]`

6.55.3.75 Element/structure stores, VST4 variants

- void vst4_u32 (uint32_t *, uint32x2x4_t)
  Form of expected instruction(s): `vst4.32 {d0, d1, d2, d3}, [r0]`
- void vst4_u16 (uint16_t *, uint16x4x4_t)
  Form of expected instruction(s): `vst4.16 {d0, d1, d2, d3}, [r0]`
- void vst4_u8 (uint8_t *, uint8x8x4_t)
  Form of expected instruction(s): `vst4.8 {d0, d1, d2, d3}, [r0]`
- void vst4_s32 (int32_t *, int32x2x4_t)
  Form of expected instruction(s): `vst4.32 {d0, d1, d2, d3}, [r0]`
- void vst4_s16 (int16_t *, int16x4x4_t)
  Form of expected instruction(s): `vst4.16 {d0, d1, d2, d3}, [r0]`
- void vst4_s8 (int8_t *, int8x8x4_t)
  Form of expected instruction(s): `vst4.8 {d0, d1, d2, d3}, [r0]`
- void vst4_f32 (float32_t *, float32x2x4_t)
  Form of expected instruction(s): `vst4.32 {d0, d1, d2, d3}, [r0]`
- void vst4_p16 (poly16_t *, poly16x4x4_t)
  Form of expected instruction(s): `vst4.16 {d0, d1, d2, d3}, [r0]`
- void vst4_p8 (poly8_t *, poly8x8x4_t)
  Form of expected instruction(s): `vst4.8 {d0, d1, d2, d3}, [r0]`
- void vst4_u64 (uint64_t *, uint64x1x4_t)
  Form of expected instruction(s): `vst1.64 {d0, d1, d2, d3}, [r0]`
- void vst4_s64 (int64_t *, int64x1x4_t)
  Form of expected instruction(s): `vst1.64 {d0, d1, d2, d3}, [r0]`
- void vst4q_u32 (uint32_t *, uint32x4x4_t)
  Form of expected instruction(s): `vst4.32 {d0, d1, d2, d3}, [r0]`

- void vst4q_u16 (uint16_t *, uint16x8x4_t)
  Form of expected instruction(s): `vst4.16 {d0, d1, d2, d3}, [r0]`
- void vst4q_u8 (uint8_t *, uint8x16x4_t)
  Form of expected instruction(s): `vst4.8 {d0, d1, d2, d3}, [r0]`
- void vst4q_s32 (int32_t *, int32x4x4_t)
  Form of expected instruction(s): `vst4.32 {d0, d1, d2, d3}, [r0]`
- void vst4q_s16 (int16_t *, int16x8x4_t)
  Form of expected instruction(s): `vst4.16 {d0, d1, d2, d3}, [r0]`
- void vst4q_s8 (int8_t *, int8x16x4_t)
  Form of expected instruction(s): `vst4.8 {d0, d1, d2, d3}, [r0]`
- void vst4q_f32 (float32_t *, float32x4x4_t)
  Form of expected instruction(s): `vst4.32 {d0, d1, d2, d3}, [r0]`
- void vst4q_p16 (poly16_t *, poly16x8x4_t)
  Form of expected instruction(s): `vst4.16 {d0, d1, d2, d3}, [r0]`
- void vst4q_p8 (poly8_t *, poly8x16x4_t)
  Form of expected instruction(s): `vst4.8 {d0, d1, d2, d3}, [r0]`
- void vst4_lane_u32 (uint32_t *, uint32x2x4_t, const int)
  Form of expected instruction(s): `vst4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_u16 (uint16_t *, uint16x4x4_t, const int)
  Form of expected instruction(s): `vst4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_u8 (uint8_t *, uint8x8x4_t, const int)
  Form of expected instruction(s): `vst4.8 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_s32 (int32_t *, int32x2x4_t, const int)
  Form of expected instruction(s): `vst4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_s16 (int16_t *, int16x4x4_t, const int)
  Form of expected instruction(s): `vst4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_s8 (int8_t *, int8x8x4_t, const int)
  Form of expected instruction(s): `vst4.8 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_f32 (float32_t *, float32x2x4_t, const int)
  Form of expected instruction(s): `vst4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_p16 (poly16_t *, poly16x4x4_t, const int)
  Form of expected instruction(s): `vst4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4_lane_p8 (poly8_t *, poly8x8x4_t, const int)
  Form of expected instruction(s): `vst4.8 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4q_lane_s32 (int32_t *, int32x4x4_t, const int)
  Form of expected instruction(s): `vst4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4q_lane_s16 (int16_t *, int16x8x4_t, const int)
  Form of expected instruction(s): `vst4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4q_lane_u32 (uint32_t *, uint32x4x4_t, const int)
  Form of expected instruction(s): `vst4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4q_lane_u16 (uint16_t *, uint16x8x4_t, const int)
  Form of expected instruction(s): `vst4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`

- void vst4q_lane_f32 (float32_t *, float32x4x4_t, const int)
  Form of expected instruction(s): `vst4.32 {d0[0], d1[0], d2[0], d3[0]}, [r0]`
- void vst4q_lane_p16 (poly16_t *, poly16x8x4_t, const int)
  Form of expected instruction(s): `vst4.16 {d0[0], d1[0], d2[0], d3[0]}, [r0]`

6.55.3.76 Logical operations (AND)

- uint32x2_t vand_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vand d0, d0, d0`
- uint16x4_t vand_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vand d0, d0, d0`
- uint8x8_t vand_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vand d0, d0, d0`
- int32x2_t vand_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vand d0, d0, d0`
- int16x4_t vand_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vand d0, d0, d0`
- int8x8_t vand_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vand d0, d0, d0`
- uint64x1_t vand_u64 (uint64x1_t, uint64x1_t)
- int64x1_t vand_s64 (int64x1_t, int64x1_t)
- uint32x4_t vandq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vand q0, q0, q0`
- uint16x8_t vandq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vand q0, q0, q0`
- uint8x16_t vandq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vand q0, q0, q0`
- int32x4_t vandq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vand q0, q0, q0`
- int16x8_t vandq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vand q0, q0, q0`
- int8x16_t vandq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vand q0, q0, q0`
- uint64x2_t vandq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vand q0, q0, q0`
- int64x2_t vandq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vand q0, q0, q0`

6.55.3.77 Logical operations (OR)

- uint32x2_t vorr_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vorr d0, d0, d0`
- uint16x4_t vorr_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vorr d0, d0, d0`

- uint8x8_t vorr_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vorr d0, d0, d0`
- int32x2_t vorr_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vorr d0, d0, d0`
- int16x4_t vorr_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vorr d0, d0, d0`
- int8x8_t vorr_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vorr d0, d0, d0`
- uint64x1_t vorr_u64 (uint64x1_t, uint64x1_t)
- int64x1_t vorr_s64 (int64x1_t, int64x1_t)
- uint32x4_t vorrq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vorr q0, q0, q0`
- uint16x8_t vorrq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vorr q0, q0, q0`
- uint8x16_t vorrq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vorr q0, q0, q0`
- int32x4_t vorrq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vorr q0, q0, q0`
- int16x8_t vorrq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vorr q0, q0, q0`
- int8x16_t vorrq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vorr q0, q0, q0`
- uint64x2_t vorrq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vorr q0, q0, q0`
- int64x2_t vorrq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vorr q0, q0, q0`

6.55.3.78 Logical operations (exclusive OR)

- uint32x2_t veor_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `veor d0, d0, d0`
- uint16x4_t veor_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `veor d0, d0, d0`
- uint8x8_t veor_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `veor d0, d0, d0`
- int32x2_t veor_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `veor d0, d0, d0`
- int16x4_t veor_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `veor d0, d0, d0`
- int8x8_t veor_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `veor d0, d0, d0`
- uint64x1_t veor_u64 (uint64x1_t, uint64x1_t)
- int64x1_t veor_s64 (int64x1_t, int64x1_t)

- uint32x4_t veorq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `veor q0, q0, q0`
- uint16x8_t veorq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `veor q0, q0, q0`
- uint8x16_t veorq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `veor q0, q0, q0`
- int32x4_t veorq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `veor q0, q0, q0`
- int16x8_t veorq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `veor q0, q0, q0`
- int8x16_t veorq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `veor q0, q0, q0`
- uint64x2_t veorq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `veor q0, q0, q0`
- int64x2_t veorq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `veor q0, q0, q0`

6.55.3.79 Logical operations (AND-NOT)

- uint32x2_t vbic_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vbic d0, d0, d0`
- uint16x4_t vbic_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vbic d0, d0, d0`
- uint8x8_t vbic_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vbic d0, d0, d0`
- int32x2_t vbic_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vbic d0, d0, d0`
- int16x4_t vbic_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vbic d0, d0, d0`
- int8x8_t vbic_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vbic d0, d0, d0`
- uint64x1_t vbic_u64 (uint64x1_t, uint64x1_t)
- int64x1_t vbic_s64 (int64x1_t, int64x1_t)
- uint32x4_t vbicq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vbic q0, q0, q0`
- uint16x8_t vbicq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vbic q0, q0, q0`
- uint8x16_t vbicq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vbic q0, q0, q0`
- int32x4_t vbicq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vbic q0, q0, q0`
- int16x8_t vbicq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vbic q0, q0, q0`

- int8x16_t vbicq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vbic q0, q0, q0`
- uint64x2_t vbicq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vbic q0, q0, q0`
- int64x2_t vbicq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vbic q0, q0, q0`

6.55.3.80 Logical operations (OR-NOT)

- uint32x2_t vorn_u32 (uint32x2_t, uint32x2_t)
  Form of expected instruction(s): `vorn d0, d0, d0`
- uint16x4_t vorn_u16 (uint16x4_t, uint16x4_t)
  Form of expected instruction(s): `vorn d0, d0, d0`
- uint8x8_t vorn_u8 (uint8x8_t, uint8x8_t)
  Form of expected instruction(s): `vorn d0, d0, d0`
- int32x2_t vorn_s32 (int32x2_t, int32x2_t)
  Form of expected instruction(s): `vorn d0, d0, d0`
- int16x4_t vorn_s16 (int16x4_t, int16x4_t)
  Form of expected instruction(s): `vorn d0, d0, d0`
- int8x8_t vorn_s8 (int8x8_t, int8x8_t)
  Form of expected instruction(s): `vorn d0, d0, d0`
- uint64x1_t vorn_u64 (uint64x1_t, uint64x1_t)
- int64x1_t vorn_s64 (int64x1_t, int64x1_t)
- uint32x4_t vornq_u32 (uint32x4_t, uint32x4_t)
  Form of expected instruction(s): `vorn q0, q0, q0`
- uint16x8_t vornq_u16 (uint16x8_t, uint16x8_t)
  Form of expected instruction(s): `vorn q0, q0, q0`
- uint8x16_t vornq_u8 (uint8x16_t, uint8x16_t)
  Form of expected instruction(s): `vorn q0, q0, q0`
- int32x4_t vornq_s32 (int32x4_t, int32x4_t)
  Form of expected instruction(s): `vorn q0, q0, q0`
- int16x8_t vornq_s16 (int16x8_t, int16x8_t)
  Form of expected instruction(s): `vorn q0, q0, q0`
- int8x16_t vornq_s8 (int8x16_t, int8x16_t)
  Form of expected instruction(s): `vorn q0, q0, q0`
- uint64x2_t vornq_u64 (uint64x2_t, uint64x2_t)
  Form of expected instruction(s): `vorn q0, q0, q0`
- int64x2_t vornq_s64 (int64x2_t, int64x2_t)
  Form of expected instruction(s): `vorn q0, q0, q0`

6.55.3.81 Reinterpret casts

- poly8x8_t vreinterpret_p8_u32 (uint32x2_t)
- poly8x8_t vreinterpret_p8_u16 (uint16x4_t)

- poly8x8_t vreinterpret_p8_u8 (uint8x8_t)
- poly8x8_t vreinterpret_p8_s32 (int32x2_t)
- poly8x8_t vreinterpret_p8_s16 (int16x4_t)
- poly8x8_t vreinterpret_p8_s8 (int8x8_t)
- poly8x8_t vreinterpret_p8_u64 (uint64x1_t)
- poly8x8_t vreinterpret_p8_s64 (int64x1_t)
- poly8x8_t vreinterpret_p8_f32 (float32x2_t)
- poly8x8_t vreinterpret_p8_p16 (poly16x4_t)
- poly8x16_t vreinterpretq_p8_u32 (uint32x4_t)
- poly8x16_t vreinterpretq_p8_u16 (uint16x8_t)
- poly8x16_t vreinterpretq_p8_u8 (uint8x16_t)
- poly8x16_t vreinterpretq_p8_s32 (int32x4_t)
- poly8x16_t vreinterpretq_p8_s16 (int16x8_t)
- poly8x16_t vreinterpretq_p8_s8 (int8x16_t)
- poly8x16_t vreinterpretq_p8_u64 (uint64x2_t)
- poly8x16_t vreinterpretq_p8_s64 (int64x2_t)
- poly8x16_t vreinterpretq_p8_f32 (float32x4_t)
- poly8x16_t vreinterpretq_p8_p16 (poly16x8_t)
- poly16x4_t vreinterpret_p16_u32 (uint32x2_t)
- poly16x4_t vreinterpret_p16_u16 (uint16x4_t)
- poly16x4_t vreinterpret_p16_u8 (uint8x8_t)
- poly16x4_t vreinterpret_p16_s32 (int32x2_t)
- poly16x4_t vreinterpret_p16_s16 (int16x4_t)
- poly16x4_t vreinterpret_p16_s8 (int8x8_t)
- poly16x4_t vreinterpret_p16_u64 (uint64x1_t)
- poly16x4_t vreinterpret_p16_s64 (int64x1_t)
- poly16x4_t vreinterpret_p16_f32 (float32x2_t)
- poly16x4_t vreinterpret_p16_p8 (poly8x8_t)
- poly16x8_t vreinterpretq_p16_u32 (uint32x4_t)
- poly16x8_t vreinterpretq_p16_u16 (uint16x8_t)
- poly16x8_t vreinterpretq_p16_u8 (uint8x16_t)
- poly16x8_t vreinterpretq_p16_s32 (int32x4_t)
- poly16x8_t vreinterpretq_p16_s16 (int16x8_t)
- poly16x8_t vreinterpretq_p16_s8 (int8x16_t)
- poly16x8_t vreinterpretq_p16_u64 (uint64x2_t)
- poly16x8_t vreinterpretq_p16_s64 (int64x2_t)
- poly16x8_t vreinterpretq_p16_f32 (float32x4_t)
- poly16x8_t vreinterpretq_p16_p8 (poly8x16_t)
- float32x2_t vreinterpret_f32_u32 (uint32x2_t)

- float32x2_t vreinterpret_f32_u16 (uint16x4_t)
- float32x2_t vreinterpret_f32_u8 (uint8x8_t)
- float32x2_t vreinterpret_f32_s32 (int32x2_t)
- float32x2_t vreinterpret_f32_s16 (int16x4_t)
- float32x2_t vreinterpret_f32_s8 (int8x8_t)
- float32x2_t vreinterpret_f32_u64 (uint64x1_t)
- float32x2_t vreinterpret_f32_s64 (int64x1_t)
- float32x2_t vreinterpret_f32_p16 (poly16x4_t)
- float32x2_t vreinterpret_f32_p8 (poly8x8_t)
- float32x4_t vreinterpretq_f32_u32 (uint32x4_t)
- float32x4_t vreinterpretq_f32_u16 (uint16x8_t)
- float32x4_t vreinterpretq_f32_u8 (uint8x16_t)
- float32x4_t vreinterpretq_f32_s32 (int32x4_t)
- float32x4_t vreinterpretq_f32_s16 (int16x8_t)
- float32x4_t vreinterpretq_f32_s8 (int8x16_t)
- float32x4_t vreinterpretq_f32_u64 (uint64x2_t)
- float32x4_t vreinterpretq_f32_s64 (int64x2_t)
- float32x4_t vreinterpretq_f32_p16 (poly16x8_t)
- float32x4_t vreinterpretq_f32_p8 (poly8x16_t)
- int64x1_t vreinterpret_s64_u32 (uint32x2_t)
- int64x1_t vreinterpret_s64_u16 (uint16x4_t)
- int64x1_t vreinterpret_s64_u8 (uint8x8_t)
- int64x1_t vreinterpret_s64_s32 (int32x2_t)
- int64x1_t vreinterpret_s64_s16 (int16x4_t)
- int64x1_t vreinterpret_s64_s8 (int8x8_t)
- int64x1_t vreinterpret_s64_u64 (uint64x1_t)
- int64x1_t vreinterpret_s64_f32 (float32x2_t)
- int64x1_t vreinterpret_s64_p16 (poly16x4_t)
- int64x1_t vreinterpret_s64_p8 (poly8x8_t)
- int64x2_t vreinterpretq_s64_u32 (uint32x4_t)
- int64x2_t vreinterpretq_s64_u16 (uint16x8_t)
- int64x2_t vreinterpretq_s64_u8 (uint8x16_t)
- int64x2_t vreinterpretq_s64_s32 (int32x4_t)
- int64x2_t vreinterpretq_s64_s16 (int16x8_t)
- int64x2_t vreinterpretq_s64_s8 (int8x16_t)
- int64x2_t vreinterpretq_s64_u64 (uint64x2_t)
- int64x2_t vreinterpretq_s64_f32 (float32x4_t)
- int64x2_t vreinterpretq_s64_p16 (poly16x8_t)
- int64x2_t vreinterpretq_s64_p8 (poly8x16_t)

- uint64x1_t vreinterpret_u64_u32 (uint32x2_t)
- uint64x1_t vreinterpret_u64_u16 (uint16x4_t)
- uint64x1_t vreinterpret_u64_u8 (uint8x8_t)
- uint64x1_t vreinterpret_u64_s32 (int32x2_t)
- uint64x1_t vreinterpret_u64_s16 (int16x4_t)
- uint64x1_t vreinterpret_u64_s8 (int8x8_t)
- uint64x1_t vreinterpret_u64_s64 (int64x1_t)
- uint64x1_t vreinterpret_u64_f32 (float32x2_t)
- uint64x1_t vreinterpret_u64_p16 (poly16x4_t)
- uint64x1_t vreinterpret_u64_p8 (poly8x8_t)
- uint64x2_t vreinterpretq_u64_u32 (uint32x4_t)
- uint64x2_t vreinterpretq_u64_u16 (uint16x8_t)
- uint64x2_t vreinterpretq_u64_u8 (uint8x16_t)
- uint64x2_t vreinterpretq_u64_s32 (int32x4_t)
- uint64x2_t vreinterpretq_u64_s16 (int16x8_t)
- uint64x2_t vreinterpretq_u64_s8 (int8x16_t)
- uint64x2_t vreinterpretq_u64_s64 (int64x2_t)
- uint64x2_t vreinterpretq_u64_f32 (float32x4_t)
- uint64x2_t vreinterpretq_u64_p16 (poly16x8_t)
- uint64x2_t vreinterpretq_u64_p8 (poly8x16_t)
- int8x8_t vreinterpret_s8_u32 (uint32x2_t)
- int8x8_t vreinterpret_s8_u16 (uint16x4_t)
- int8x8_t vreinterpret_s8_u8 (uint8x8_t)
- int8x8_t vreinterpret_s8_s32 (int32x2_t)
- int8x8_t vreinterpret_s8_s16 (int16x4_t)
- int8x8_t vreinterpret_s8_u64 (uint64x1_t)
- int8x8_t vreinterpret_s8_s64 (int64x1_t)
- int8x8_t vreinterpret_s8_f32 (float32x2_t)
- int8x8_t vreinterpret_s8_p16 (poly16x4_t)
- int8x8_t vreinterpret_s8_p8 (poly8x8_t)
- int8x16_t vreinterpretq_s8_u32 (uint32x4_t)
- int8x16_t vreinterpretq_s8_u16 (uint16x8_t)
- int8x16_t vreinterpretq_s8_u8 (uint8x16_t)
- int8x16_t vreinterpretq_s8_s32 (int32x4_t)
- int8x16_t vreinterpretq_s8_s16 (int16x8_t)
- int8x16_t vreinterpretq_s8_u64 (uint64x2_t)
- int8x16_t vreinterpretq_s8_s64 (int64x2_t)
- int8x16_t vreinterpretq_s8_f32 (float32x4_t)
- int8x16_t vreinterpretq_s8_p16 (poly16x8_t)

- int8x16_t vreinterpretq_s8_p8 (poly8x16_t)
- int16x4_t vreinterpret_s16_u32 (uint32x2_t)
- int16x4_t vreinterpret_s16_u16 (uint16x4_t)
- int16x4_t vreinterpret_s16_u8 (uint8x8_t)
- int16x4_t vreinterpret_s16_s32 (int32x2_t)
- int16x4_t vreinterpret_s16_s8 (int8x8_t)
- int16x4_t vreinterpret_s16_u64 (uint64x1_t)
- int16x4_t vreinterpret_s16_s64 (int64x1_t)
- int16x4_t vreinterpret_s16_f32 (float32x2_t)
- int16x4_t vreinterpret_s16_p16 (poly16x4_t)
- int16x4_t vreinterpret_s16_p8 (poly8x8_t)
- int16x8_t vreinterpretq_s16_u32 (uint32x4_t)
- int16x8_t vreinterpretq_s16_u16 (uint16x8_t)
- int16x8_t vreinterpretq_s16_u8 (uint8x16_t)
- int16x8_t vreinterpretq_s16_s32 (int32x4_t)
- int16x8_t vreinterpretq_s16_s8 (int8x16_t)
- int16x8_t vreinterpretq_s16_u64 (uint64x2_t)
- int16x8_t vreinterpretq_s16_s64 (int64x2_t)
- int16x8_t vreinterpretq_s16_f32 (float32x4_t)
- int16x8_t vreinterpretq_s16_p16 (poly16x8_t)
- int16x8_t vreinterpretq_s16_p8 (poly8x16_t)
- int32x2_t vreinterpret_s32_u32 (uint32x2_t)
- int32x2_t vreinterpret_s32_u16 (uint16x4_t)
- int32x2_t vreinterpret_s32_u8 (uint8x8_t)
- int32x2_t vreinterpret_s32_s16 (int16x4_t)
- int32x2_t vreinterpret_s32_s8 (int8x8_t)
- int32x2_t vreinterpret_s32_u64 (uint64x1_t)
- int32x2_t vreinterpret_s32_s64 (int64x1_t)
- int32x2_t vreinterpret_s32_f32 (float32x2_t)
- int32x2_t vreinterpret_s32_p16 (poly16x4_t)
- int32x2_t vreinterpret_s32_p8 (poly8x8_t)
- int32x4_t vreinterpretq_s32_u32 (uint32x4_t)
- int32x4_t vreinterpretq_s32_u16 (uint16x8_t)
- int32x4_t vreinterpretq_s32_u8 (uint8x16_t)
- int32x4_t vreinterpretq_s32_s16 (int16x8_t)
- int32x4_t vreinterpretq_s32_s8 (int8x16_t)
- int32x4_t vreinterpretq_s32_u64 (uint64x2_t)
- int32x4_t vreinterpretq_s32_s64 (int64x2_t)
- int32x4_t vreinterpretq_s32_f32 (float32x4_t)

- int32x4_t vreinterpretq_s32_p16 (poly16x8_t)
- int32x4_t vreinterpretq_s32_p8 (poly8x16_t)
- uint8x8_t vreinterpret_u8_u32 (uint32x2_t)
- uint8x8_t vreinterpret_u8_u16 (uint16x4_t)
- uint8x8_t vreinterpret_u8_s32 (int32x2_t)
- uint8x8_t vreinterpret_u8_s16 (int16x4_t)
- uint8x8_t vreinterpret_u8_s8 (int8x8_t)
- uint8x8_t vreinterpret_u8_u64 (uint64x1_t)
- uint8x8_t vreinterpret_u8_s64 (int64x1_t)
- uint8x8_t vreinterpret_u8_f32 (float32x2_t)
- uint8x8_t vreinterpret_u8_p16 (poly16x4_t)
- uint8x8_t vreinterpret_u8_p8 (poly8x8_t)
- uint8x16_t vreinterpretq_u8_u32 (uint32x4_t)
- uint8x16_t vreinterpretq_u8_u16 (uint16x8_t)
- uint8x16_t vreinterpretq_u8_s32 (int32x4_t)
- uint8x16_t vreinterpretq_u8_s16 (int16x8_t)
- uint8x16_t vreinterpretq_u8_s8 (int8x16_t)
- uint8x16_t vreinterpretq_u8_u64 (uint64x2_t)
- uint8x16_t vreinterpretq_u8_s64 (int64x2_t)
- uint8x16_t vreinterpretq_u8_f32 (float32x4_t)
- uint8x16_t vreinterpretq_u8_p16 (poly16x8_t)
- uint8x16_t vreinterpretq_u8_p8 (poly8x16_t)
- uint16x4_t vreinterpret_u16_u32 (uint32x2_t)
- uint16x4_t vreinterpret_u16_u8 (uint8x8_t)
- uint16x4_t vreinterpret_u16_s32 (int32x2_t)
- uint16x4_t vreinterpret_u16_s16 (int16x4_t)
- uint16x4_t vreinterpret_u16_s8 (int8x8_t)
- uint16x4_t vreinterpret_u16_u64 (uint64x1_t)
- uint16x4_t vreinterpret_u16_s64 (int64x1_t)
- uint16x4_t vreinterpret_u16_f32 (float32x2_t)
- uint16x4_t vreinterpret_u16_p16 (poly16x4_t)
- uint16x4_t vreinterpret_u16_p8 (poly8x8_t)
- uint16x8_t vreinterpretq_u16_u32 (uint32x4_t)
- uint16x8_t vreinterpretq_u16_u8 (uint8x16_t)
- uint16x8_t vreinterpretq_u16_s32 (int32x4_t)
- uint16x8_t vreinterpretq_u16_s16 (int16x8_t)
- uint16x8_t vreinterpretq_u16_s8 (int8x16_t)
- uint16x8_t vreinterpretq_u16_u64 (uint64x2_t)
- uint16x8_t vreinterpretq_u16_s64 (int64x2_t)

- uint16x8_t vreinterpretq_u16_f32 (float32x4_t)
- uint16x8_t vreinterpretq_u16_p16 (poly16x8_t)
- uint16x8_t vreinterpretq_u16_p8 (poly8x16_t)
- uint32x2_t vreinterpret_u32_u16 (uint16x4_t)
- uint32x2_t vreinterpret_u32_u8 (uint8x8_t)
- uint32x2_t vreinterpret_u32_s32 (int32x2_t)
- uint32x2_t vreinterpret_u32_s16 (int16x4_t)
- uint32x2_t vreinterpret_u32_s8 (int8x8_t)
- uint32x2_t vreinterpret_u32_u64 (uint64x1_t)
- uint32x2_t vreinterpret_u32_s64 (int64x1_t)
- uint32x2_t vreinterpret_u32_f32 (float32x2_t)
- uint32x2_t vreinterpret_u32_p16 (poly16x4_t)
- uint32x2_t vreinterpret_u32_p8 (poly8x8_t)
- uint32x4_t vreinterpretq_u32_u16 (uint16x8_t)
- uint32x4_t vreinterpretq_u32_u8 (uint8x16_t)
- uint32x4_t vreinterpretq_u32_s32 (int32x4_t)
- uint32x4_t vreinterpretq_u32_s16 (int16x8_t)
- uint32x4_t vreinterpretq_u32_s8 (int8x16_t)
- uint32x4_t vreinterpretq_u32_u64 (uint64x2_t)
- uint32x4_t vreinterpretq_u32_s64 (int64x2_t)
- uint32x4_t vreinterpretq_u32_f32 (float32x4_t)
- uint32x4_t vreinterpretq_u32_p16 (poly16x8_t)
- uint32x4_t vreinterpretq_u32_p8 (poly8x16_t)

6.55.4 AVR Built-in Functions

For each built-in function for AVR, there is an equally named, uppercase built-in macro defined. That way users can easily query if or if not a specific built-in is implemented or not. For example, if `__builtin_avr_nop` is available the macro `__BUILTIN_AVR_NOP` is defined to `1` and undefined otherwise.

The following built-in functions map to the respective machine instruction, i.e. `nop`, `sei`, `cli`, `sleep`, `wdr`, `swap`, `fmul`, `fmuls` resp. `fmulsu`. The three `fmul*` built-ins are implemented as library call if no hardware multiplier is available.

```
void __builtin_avr_nop (void)
void __builtin_avr_sei (void)
void __builtin_avr_cli (void)
void __builtin_avr_sleep (void)
void __builtin_avr_wdr (void)
unsigned char __builtin_avr_swap (unsigned char)
unsigned int __builtin_avr_fmul (unsigned char, unsigned char)
int __builtin_avr_fmuls (char, char)
int __builtin_avr_fmulsu (char, unsigned char)
```

In order to delay execution for a specific number of cycles, GCC implements

```
void __builtin_avr_delay_cycles (unsigned long ticks)
```

`ticks` is the number of ticks to delay execution. Note that this built-in does not take into account the effect of interrupts which might increase delay time. `ticks` must be a compile time integer constant; delays with a variable number of cycles are not supported.

```
char __builtin_avr_flash_segment (const __memx void*)
```

This built-in takes a byte address to the 24-bit [AVR Named Address Spaces], page 322 `__memx` and returns the number of the flash segment (the 64 KiB chunk) where the address points to. Counting starts at `0`. If the address does not point to flash memory, return `-1`.

```
unsigned char __builtin_avr_insert_bits (unsigned long map, unsigned char bits, unsigned char val)
```

Insert bits from *bits* into *val* and return the resulting value. The nibbles of *map* determine how the insertion is performed: Let X be the n-th nibble of *map*

1. If X is `0xf`, then the n-th bit of *val* is returned unaltered.
2. If X is in the range 0...7, then the n-th result bit is set to the X-th bit of *bits*
3. If X is in the range 8...`0xe`, then the n-th result bit is undefined.

One typical use case for this built-in is adjusting input and output values to non-contiguous port layouts. Some examples:

```
// same as val, bits is unused
__builtin_avr_insert_bits (0xffffffff, bits, val)

// same as bits, val is unused
__builtin_avr_insert_bits (0x76543210, bits, val)

// same as rotating bits by 4
__builtin_avr_insert_bits (0x32107654, bits, 0)

// high-nibble of result is the high-nibble of val
// low-nibble of result is the low-nibble of bits
__builtin_avr_insert_bits (0xffff3210, bits, val)

// reverse the bit order of bits
__builtin_avr_insert_bits (0x01234567, bits, 0)
```

6.55.5 Blackfin Built-in Functions

Currently, there are two Blackfin-specific built-in functions. These are used for generating `CSYNC` and `SSYNC` machine insns without using inline assembly; by using these built-in functions the compiler can automatically add workarounds for hardware errata involving these instructions. These functions are named as follows:

```
void __builtin_bfin_csync (void)
void __builtin_bfin_ssync (void)
```

6.55.6 FR-V Built-in Functions

GCC provides many FR-V-specific built-in functions. In general, these functions are intended to be compatible with those described by *FR-V Family, Softune C/C++ Compiler Manual (V6), Fujitsu Semiconductor*. The two exceptions are `__MDUNPACKH` and `__MBTOHE`, the gcc forms of which pass 128-bit values by pointer rather than by value.

Most of the functions are named after specific FR-V instructions. Such functions are said to be "directly mapped" and are summarized here in tabular form.

6.55.6.1 Argument Types

The arguments to the built-in functions can be divided into three groups: register numbers, compile-time constants and run-time values. In order to make this classification clear at a glance, the arguments and return values are given the following pseudo types:

Pseudo type	Real C type	Constant?	Description
`uh`	`unsigned short`	No	an unsigned halfword
`uw1`	`unsigned int`	No	an unsigned word
`sw1`	`int`	No	a signed word
`uw2`	`unsigned long long`	No	an unsigned doubleword
`sw2`	`long long`	No	a signed doubleword
`const`	`int`	Yes	an integer constant
`acc`	`int`	Yes	an ACC register number
`iacc`	`int`	Yes	an IACC register number

These pseudo types are not defined by GCC, they are simply a notational convenience used in this manual.

Arguments of type `uh`, `uw1`, `sw1`, `uw2` and `sw2` are evaluated at run time. They correspond to register operands in the underlying FR-V instructions.

`const` arguments represent immediate operands in the underlying FR-V instructions. They must be compile-time constants.

`acc` arguments are evaluated at compile time and specify the number of an accumulator register. For example, an `acc` argument of 2 will select the ACC2 register.

`iacc` arguments are similar to `acc` arguments but specify the number of an IACC register. See see Section 6.55.6.5 [Other Built-in Functions], page 537 for more details.

6.55.6.2 Directly-mapped Integer Functions

The functions listed below map directly to FR-V I-type instructions.

Function prototype	Example usage	Assembly output
`sw1 __ADDSS (sw1, sw1)`	`c = __ADDSS (a, b)`	`ADDSS a,b,c`
`sw1 __SCAN (sw1, sw1)`	`c = __SCAN (a, b)`	`SCAN a,b,c`
`sw1 __SCUTSS (sw1)`	`b = __SCUTSS (a)`	`SCUTSS a,b`
`sw1 __SLASS (sw1, sw1)`	`c = __SLASS (a, b)`	`SLASS a,b,c`
`void __SMASS (sw1, sw1)`	`__SMASS (a, b)`	`SMASS a,b`
`void __SMSSS (sw1, sw1)`	`__SMSSS (a, b)`	`SMSSS a,b`
`void __SMU (sw1, sw1)`	`__SMU (a, b)`	`SMU a,b`
`sw2 __SMUL (sw1, sw1)`	`c = __SMUL (a, b)`	`SMUL a,b,c`
`sw1 __SUBSS (sw1, sw1)`	`c = __SUBSS (a, b)`	`SUBSS a,b,c`
`uw2 __UMUL (uw1, uw1)`	`c = __UMUL (a, b)`	`UMUL a,b,c`

6.55.6.3 Directly-mapped Media Functions

The functions listed below map directly to FR-V M-type instructions.

Function prototype	Example usage	Assembly output
`uw1 __MABSHS (sw1)`	`b = __MABSHS (a)`	`MABSHS a,b`
`void __MADDACCS (acc, acc)`	`__MADDACCS (b, a)`	`MADDACCS a,b`
`sw1 __MADDHSS (sw1, sw1)`	`c = __MADDHSS (a, b)`	`MADDHSS a,b,c`

`uw1 __MADDHUS (uw1, uw1)`	`c = __MADDHUS (a, b)`	`MADDHUS a,b,c`
`uw1 __MAND (uw1, uw1)`	`c = __MAND (a, b)`	`MAND a,b,c`
`void __MASACCS (acc, acc)`	`__MASACCS (b, a)`	`MASACCS a,b`
`uw1 __MAVEH (uw1, uw1)`	`c = __MAVEH (a, b)`	`MAVEH a,b,c`
`uw2 __MBTOH (uw1)`	`b = __MBTOH (a)`	`MBTOH a,b`
`void __MBTOHE (uw1 *, uw1)`	`__MBTOHE (&b, a)`	`MBTOHE a,b`
`void __MCLRACC (acc)`	`__MCLRACC (a)`	`MCLRACC a`
`void __MCLRACCA (void)`	`__MCLRACCA ()`	`MCLRACCA`
`uw1 __Mcop1 (uw1, uw1)`	`c = __Mcop1 (a, b)`	`Mcop1 a,b,c`
`uw1 __Mcop2 (uw1, uw1)`	`c = __Mcop2 (a, b)`	`Mcop2 a,b,c`
`uw1 __MCPLHI (uw2, const)`	`c = __MCPLHI (a, b)`	`MCPLHI a,#b,c`
`uw1 __MCPLI (uw2, const)`	`c = __MCPLI (a, b)`	`MCPLI a,#b,c`
`void __MCPXIS (acc, sw1, sw1)`	`__MCPXIS (c, a, b)`	`MCPXIS a,b,c`
`void __MCPXIU (acc, uw1, uw1)`	`__MCPXIU (c, a, b)`	`MCPXIU a,b,c`
`void __MCPXRS (acc, sw1, sw1)`	`__MCPXRS (c, a, b)`	`MCPXRS a,b,c`
`void __MCPXRU (acc, uw1, uw1)`	`__MCPXRU (c, a, b)`	`MCPXRU a,b,c`
`uw1 __MCUT (acc, uw1)`	`c = __MCUT (a, b)`	`MCUT a,b,c`
`uw1 __MCUTSS (acc, sw1)`	`c = __MCUTSS (a, b)`	`MCUTSS a,b,c`
`void __MDADDACCS (acc, acc)`	`__MDADDACCS (b, a)`	`MDADDACCS a,b`
`void __MDASACCS (acc, acc)`	`__MDASACCS (b, a)`	`MDASACCS a,b`
`uw2 __MDCUTSSI (acc, const)`	`c = __MDCUTSSI (a, b)`	`MDCUTSSI a,#b,c`
`uw2 __MDPACKH (uw2, uw2)`	`c = __MDPACKH (a, b)`	`MDPACKH a,b,c`
`uw2 __MDROTLI (uw2, const)`	`c = __MDROTLI (a, b)`	`MDROTLI a,#b,c`
`void __MDSUBACCS (acc, acc)`	`__MDSUBACCS (b, a)`	`MDSUBACCS a,b`
`void __MDUNPACKH (uw1 *, uw2)`	`__MDUNPACKH (&b, a)`	`MDUNPACKH a,b`
`uw2 __MEXPDHD (uw1, const)`	`c = __MEXPDHD (a, b)`	`MEXPDHD a,#b,c`
`uw1 __MEXPDHW (uw1, const)`	`c = __MEXPDHW (a, b)`	`MEXPDHW a,#b,c`
`uw1 __MHDSETH (uw1, const)`	`c = __MHDSETH (a, b)`	`MHDSETH a,#b,c`
`sw1 __MHDSETS (const)`	`b = __MHDSETS (a)`	`MHDSETS #a,b`
`uw1 __MHSETHIH (uw1, const)`	`b = __MHSETHIH (b, a)`	`MHSETHIH #a,b`
`sw1 __MHSETHIS (sw1, const)`	`b = __MHSETHIS (b, a)`	`MHSETHIS #a,b`
`uw1 __MHSETLOH (uw1, const)`	`b = __MHSETLOH (b, a)`	`MHSETLOH #a,b`
`sw1 __MHSETLOS (sw1, const)`	`b = __MHSETLOS (b, a)`	`MHSETLOS #a,b`
`uw1 __MHTOB (uw2)`	`b = __MHTOB (a)`	`MHTOB a,b`
`void __MMACHS (acc, sw1, sw1)`	`__MMACHS (c, a, b)`	`MMACHS a,b,c`
`void __MMACHU (acc, uw1, uw1)`	`__MMACHU (c, a, b)`	`MMACHU a,b,c`
`void __MMRDHS (acc, sw1, sw1)`	`__MMRDHS (c, a, b)`	`MMRDHS a,b,c`
`void __MMRDHU (acc, uw1, uw1)`	`__MMRDHU (c, a, b)`	`MMRDHU a,b,c`
`void __MMULHS (acc, sw1, sw1)`	`__MMULHS (c, a, b)`	`MMULHS a,b,c`
`void __MMULHU (acc, uw1, uw1)`	`__MMULHU (c, a, b)`	`MMULHU a,b,c`
`void __MMULXHS (acc, sw1, sw1)`	`__MMULXHS (c, a, b)`	`MMULXHS a,b,c`
`void __MMULXHU (acc, uw1, uw1)`	`__MMULXHU (c, a, b)`	`MMULXHU a,b,c`
`uw1 __MNOT (uw1)`	`b = __MNOT (a)`	`MNOT a,b`
`uw1 __MOR (uw1, uw1)`	`c = __MOR (a, b)`	`MOR a,b,c`
`uw1 __MPACKH (uh, uh)`	`c = __MPACKH (a, b)`	`MPACKH a,b,c`
`sw2 __MQADDHSS (sw2, sw2)`	`c = __MQADDHSS (a, b)`	`MQADDHSS a,b,c`
`uw2 __MQADDHUS (uw2, uw2)`	`c = __MQADDHUS (a, b)`	`MQADDHUS a,b,c`

`void __MQCPXIS (acc, sw2, sw2)`	`__MQCPXIS (c, a, b)`	`MQCPXIS a,b,c`
`void __MQCPXIU (acc, uw2, uw2)`	`__MQCPXIU (c, a, b)`	`MQCPXIU a,b,c`
`void __MQCPXRS (acc, sw2, sw2)`	`__MQCPXRS (c, a, b)`	`MQCPXRS a,b,c`
`void __MQCPXRU (acc, uw2, uw2)`	`__MQCPXRU (c, a, b)`	`MQCPXRU a,b,c`
`sw2 __MQLCLRHS (sw2, sw2)`	`c = __MQLCLRHS (a, b)`	`MQLCLRHS a,b,c`
`sw2 __MQLMTHS (sw2, sw2)`	`c = __MQLMTHS (a, b)`	`MQLMTHS a,b,c`
`void __MQMACHS (acc, sw2, sw2)`	`__MQMACHS (c, a, b)`	`MQMACHS a,b,c`
`void __MQMACHU (acc, uw2, uw2)`	`__MQMACHU (c, a, b)`	`MQMACHU a,b,c`
`void __MQMACXHS (acc, sw2, sw2)`	`__MQMACXHS (c, a, b)`	`MQMACXHS a,b,c`
`void __MQMULHS (acc, sw2, sw2)`	`__MQMULHS (c, a, b)`	`MQMULHS a,b,c`
`void __MQMULHU (acc, uw2, uw2)`	`__MQMULHU (c, a, b)`	`MQMULHU a,b,c`
`void __MQMULXHS (acc, sw2, sw2)`	`__MQMULXHS (c, a, b)`	`MQMULXHS a,b,c`
`void __MQMULXHU (acc, uw2, uw2)`	`__MQMULXHU (c, a, b)`	`MQMULXHU a,b,c`
`sw2 __MQSATHS (sw2, sw2)`	`c = __MQSATHS (a, b)`	`MQSATHS a,b,c`
`uw2 __MQSLLHI (uw2, int)`	`c = __MQSLLHI (a, b)`	`MQSLLHI a,b,c`
`sw2 __MQSRAHI (sw2, int)`	`c = __MQSRAHI (a, b)`	`MQSRAHI a,b,c`
`sw2 __MQSUBHSS (sw2, sw2)`	`c = __MQSUBHSS (a, b)`	`MQSUBHSS a,b,c`
`uw2 __MQSUBHUS (uw2, uw2)`	`c = __MQSUBHUS (a, b)`	`MQSUBHUS a,b,c`
`void __MQXMACHS (acc, sw2, sw2)`	`__MQXMACHS (c, a, b)`	`MQXMACHS a,b,c`
`void __MQXMACXHS (acc, sw2, sw2)`	`__MQXMACXHS (c, a, b)`	`MQXMACXHS a,b,c`
`uw1 __MRDACC (acc)`	`b = __MRDACC (a)`	`MRDACC a,b`
`uw1 __MRDACCG (acc)`	`b = __MRDACCG (a)`	`MRDACCG a,b`
`uw1 __MROTLI (uw1, const)`	`c = __MROTLI (a, b)`	`MROTLI a,#b,c`
`uw1 __MROTRI (uw1, const)`	`c = __MROTRI (a, b)`	`MROTRI a,#b,c`
`sw1 __MSATHS (sw1, sw1)`	`c = __MSATHS (a, b)`	`MSATHS a,b,c`
`uw1 __MSATHU (uw1, uw1)`	`c = __MSATHU (a, b)`	`MSATHU a,b,c`
`uw1 __MSLLHI (uw1, const)`	`c = __MSLLHI (a, b)`	`MSLLHI a,#b,c`
`sw1 __MSRAHI (sw1, const)`	`c = __MSRAHI (a, b)`	`MSRAHI a,#b,c`
`uw1 __MSRLHI (uw1, const)`	`c = __MSRLHI (a, b)`	`MSRLHI a,#b,c`
`void __MSUBACCS (acc, acc)`	`__MSUBACCS (b, a)`	`MSUBACCS a,b`
`sw1 __MSUBHSS (sw1, sw1)`	`c = __MSUBHSS (a, b)`	`MSUBHSS a,b,c`
`uw1 __MSUBHUS (uw1, uw1)`	`c = __MSUBHUS (a, b)`	`MSUBHUS a,b,c`
`void __MTRAP (void)`	`__MTRAP ()`	`MTRAP`
`uw2 __MUNPACKH (uw1)`	`b = __MUNPACKH (a)`	`MUNPACKH a,b`
`uw1 __MWCUT (uw2, uw1)`	`c = __MWCUT (a, b)`	`MWCUT a,b,c`
`void __MWTACC (acc, uw1)`	`__MWTACC (b, a)`	`MWTACC a,b`
`void __MWTACCG (acc, uw1)`	`__MWTACCG (b, a)`	`MWTACCG a,b`
`uw1 __MXOR (uw1, uw1)`	`c = __MXOR (a, b)`	`MXOR a,b,c`

6.55.6.4 Raw read/write Functions

This sections describes built-in functions related to read and write instructions to access memory. These functions generate `membar` instructions to flush the I/O load and stores where appropriate, as described in Fujitsu's manual described above.

```
unsigned char __builtin_read8 (void *data)
unsigned short __builtin_read16 (void *data)
unsigned long __builtin_read32 (void *data)
unsigned long long __builtin_read64 (void *data)
void __builtin_write8 (void *data, unsigned char datum)
void __builtin_write16 (void *data, unsigned short datum)
void __builtin_write32 (void *data, unsigned long datum)
void __builtin_write64 (void *data, unsigned long long datum)
```

6.55.6.5 Other Built-in Functions

This section describes built-in functions that are not named after a specific FR-V instruction.

`sw2 __IACCreadll (iacc reg)`
: Return the full 64-bit value of IACC0. The *reg* argument is reserved for future expansion and must be 0.

`sw1 __IACCreadl (iacc reg)`
: Return the value of IACC0H if *reg* is 0 and IACC0L if *reg* is 1. Other values of *reg* are rejected as invalid.

`void __IACCsetll (iacc reg, sw2 x)`
: Set the full 64-bit value of IACC0 to *x*. The *reg* argument is reserved for future expansion and must be 0.

`void __IACCsetl (iacc reg, sw1 x)`
: Set IACC0H to *x* if *reg* is 0 and IACC0L to *x* if *reg* is 1. Other values of *reg* are rejected as invalid.

`void __data_prefetch0 (const void *x)`
: Use the `dcpl` instruction to load the contents of address *x* into the data cache.

`void __data_prefetch (const void *x)`
: Use the `nldub` instruction to load the contents of address *x* into the data cache. The instruction will be issued in slot I1.

6.55.7 X86 Built-in Functions

These built-in functions are available for the i386 and x86-64 family of computers, depending on the command-line switches used.

Note that, if you specify command-line switches such as '`-msse`', the compiler could use the extended instruction sets even if the built-ins are not used explicitly in the program. For this reason, applications which perform runtime 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.

The following machine modes are available for use with MMX built-in functions (see Section 6.49 [Vector Extensions], page 423): `V2SI` for a vector of two 32-bit integers, `V4HI` for a vector of four 16-bit integers, and `V8QI` for a vector of eight 8-bit integers. Some of the built-in functions operate on MMX registers as a whole 64-bit entity, these use `V1DI` as their mode.

If 3DNow! extensions are enabled, `V2SF` is used as a mode for a vector of two 32-bit floating point values.

If SSE extensions are enabled, `V4SF` is used for a vector of four 32-bit floating point values. Some instructions use a vector of four 32-bit integers, these use `V4SI`. Finally, some instructions operate on an entire vector register, interpreting it as a 128-bit integer, these use mode `TI`.

In 64-bit mode, the x86-64 family of processors uses additional built-in functions for efficient use of `TF` (`__float128`) 128-bit floating point and `TC` 128-bit complex floating point values.

The following floating point built-in functions are available in 64-bit mode. All of them implement the function that is part of the name.

```
__float128 __builtin_fabsq (__float128)
__float128 __builtin_copysignq (__float128, __float128)
```

The following built-in function is always available.

`void __builtin_ia32_pause (void)`
: Generates the `pause` machine instruction with a compiler memory barrier.

The following floating point built-in functions are made available in the 64-bit mode.

`__float128 __builtin_infq (void)`
: Similar to `__builtin_inf`, except the return type is `__float128`.

`__float128 __builtin_huge_valq (void)`
: Similar to `__builtin_huge_val`, except the return type is `__float128`.

The following built-in functions are made available by '`-mmmx`'. All of them generate the machine instruction that is part of the name.

```
v8qi __builtin_ia32_paddb (v8qi, v8qi)
v4hi __builtin_ia32_paddw (v4hi, v4hi)
v2si __builtin_ia32_paddd (v2si, v2si)
v8qi __builtin_ia32_psubb (v8qi, v8qi)
v4hi __builtin_ia32_psubw (v4hi, v4hi)
v2si __builtin_ia32_psubd (v2si, v2si)
v8qi __builtin_ia32_paddsb (v8qi, v8qi)
v4hi __builtin_ia32_paddsw (v4hi, v4hi)
v8qi __builtin_ia32_psubsb (v8qi, v8qi)
v4hi __builtin_ia32_psubsw (v4hi, v4hi)
v8qi __builtin_ia32_paddusb (v8qi, v8qi)
v4hi __builtin_ia32_paddusw (v4hi, v4hi)
v8qi __builtin_ia32_psubusb (v8qi, v8qi)
v4hi __builtin_ia32_psubusw (v4hi, v4hi)
v4hi __builtin_ia32_pmullw (v4hi, v4hi)
v4hi __builtin_ia32_pmulhw (v4hi, v4hi)
di __builtin_ia32_pand (di, di)
di __builtin_ia32_pandn (di,di)
di __builtin_ia32_por (di, di)
di __builtin_ia32_pxor (di, di)
v8qi __builtin_ia32_pcmpeqb (v8qi, v8qi)
v4hi __builtin_ia32_pcmpeqw (v4hi, v4hi)
v2si __builtin_ia32_pcmpeqd (v2si, v2si)
v8qi __builtin_ia32_pcmpgtb (v8qi, v8qi)
v4hi __builtin_ia32_pcmpgtw (v4hi, v4hi)
v2si __builtin_ia32_pcmpgtd (v2si, v2si)
```

```
v8qi __builtin_ia32_punpckhbw (v8qi, v8qi)
v4hi __builtin_ia32_punpckhwd (v4hi, v4hi)
v2si __builtin_ia32_punpckhdq (v2si, v2si)
v8qi __builtin_ia32_punpcklbw (v8qi, v8qi)
v4hi __builtin_ia32_punpcklwd (v4hi, v4hi)
v2si __builtin_ia32_punpckldq (v2si, v2si)
v8qi __builtin_ia32_packsswb (v4hi, v4hi)
v4hi __builtin_ia32_packssdw (v2si, v2si)
v8qi __builtin_ia32_packuswb (v4hi, v4hi)

v4hi __builtin_ia32_psllw (v4hi, v4hi)
v2si __builtin_ia32_pslld (v2si, v2si)
v1di __builtin_ia32_psllq (v1di, v1di)
v4hi __builtin_ia32_psrlw (v4hi, v4hi)
v2si __builtin_ia32_psrld (v2si, v2si)
v1di __builtin_ia32_psrlq (v1di, v1di)
v4hi __builtin_ia32_psraw (v4hi, v4hi)
v2si __builtin_ia32_psrad (v2si, v2si)
v4hi __builtin_ia32_psllwi (v4hi, int)
v2si __builtin_ia32_pslldi (v2si, int)
v1di __builtin_ia32_psllqi (v1di, int)
v4hi __builtin_ia32_psrlwi (v4hi, int)
v2si __builtin_ia32_psrldi (v2si, int)
v1di __builtin_ia32_psrlqi (v1di, int)
v4hi __builtin_ia32_psrawi (v4hi, int)
v2si __builtin_ia32_psradi (v2si, int)
```

The following built-in functions are made available either with '`-msse`', or with a combination of '`-m3dnow`' and '`-march=athlon`'. All of them generate the machine instruction that is part of the name.

```
v4hi __builtin_ia32_pmulhuw (v4hi, v4hi)
v8qi __builtin_ia32_pavgb (v8qi, v8qi)
v4hi __builtin_ia32_pavgw (v4hi, v4hi)
v1di __builtin_ia32_psadbw (v8qi, v8qi)
v8qi __builtin_ia32_pmaxub (v8qi, v8qi)
v4hi __builtin_ia32_pmaxsw (v4hi, v4hi)
v8qi __builtin_ia32_pminub (v8qi, v8qi)
v4hi __builtin_ia32_pminsw (v4hi, v4hi)
int __builtin_ia32_pextrw (v4hi, int)
v4hi __builtin_ia32_pinsrw (v4hi, int, int)
int __builtin_ia32_pmovmskb (v8qi)
void __builtin_ia32_maskmovq (v8qi, v8qi, char *)
void __builtin_ia32_movntq (di *, di)
void __builtin_ia32_sfence (void)
```

The following built-in functions are available when '`-msse`' is used. All of them generate the machine instruction that is part of the name.

```
int __builtin_ia32_comieq (v4sf, v4sf)
int __builtin_ia32_comineq (v4sf, v4sf)
int __builtin_ia32_comilt (v4sf, v4sf)
int __builtin_ia32_comile (v4sf, v4sf)
int __builtin_ia32_comigt (v4sf, v4sf)
int __builtin_ia32_comige (v4sf, v4sf)
int __builtin_ia32_ucomieq (v4sf, v4sf)
int __builtin_ia32_ucomineq (v4sf, v4sf)
int __builtin_ia32_ucomilt (v4sf, v4sf)
int __builtin_ia32_ucomile (v4sf, v4sf)
```

```
int __builtin_ia32_ucomigt (v4sf, v4sf)
int __builtin_ia32_ucomige (v4sf, v4sf)
v4sf __builtin_ia32_addps (v4sf, v4sf)
v4sf __builtin_ia32_subps (v4sf, v4sf)
v4sf __builtin_ia32_mulps (v4sf, v4sf)
v4sf __builtin_ia32_divps (v4sf, v4sf)
v4sf __builtin_ia32_addss (v4sf, v4sf)
v4sf __builtin_ia32_subss (v4sf, v4sf)
v4sf __builtin_ia32_mulss (v4sf, v4sf)
v4sf __builtin_ia32_divss (v4sf, v4sf)
v4si __builtin_ia32_cmpeqps (v4sf, v4sf)
v4si __builtin_ia32_cmpltps (v4sf, v4sf)
v4si __builtin_ia32_cmpleps (v4sf, v4sf)
v4si __builtin_ia32_cmpgtps (v4sf, v4sf)
v4si __builtin_ia32_cmpgeps (v4sf, v4sf)
v4si __builtin_ia32_cmpunordps (v4sf, v4sf)
v4si __builtin_ia32_cmpneqps (v4sf, v4sf)
v4si __builtin_ia32_cmpnltps (v4sf, v4sf)
v4si __builtin_ia32_cmpnleps (v4sf, v4sf)
v4si __builtin_ia32_cmpngtps (v4sf, v4sf)
v4si __builtin_ia32_cmpngeps (v4sf, v4sf)
v4si __builtin_ia32_cmpordps (v4sf, v4sf)
v4si __builtin_ia32_cmpeqss (v4sf, v4sf)
v4si __builtin_ia32_cmpltss (v4sf, v4sf)
v4si __builtin_ia32_cmpless (v4sf, v4sf)
v4si __builtin_ia32_cmpunordss (v4sf, v4sf)
v4si __builtin_ia32_cmpneqss (v4sf, v4sf)
v4si __builtin_ia32_cmpnlts (v4sf, v4sf)
v4si __builtin_ia32_cmpnless (v4sf, v4sf)
v4si __builtin_ia32_cmpordss (v4sf, v4sf)
v4sf __builtin_ia32_maxps (v4sf, v4sf)
v4sf __builtin_ia32_maxss (v4sf, v4sf)
v4sf __builtin_ia32_minps (v4sf, v4sf)
v4sf __builtin_ia32_minss (v4sf, v4sf)
v4sf __builtin_ia32_andps (v4sf, v4sf)
v4sf __builtin_ia32_andnps (v4sf, v4sf)
v4sf __builtin_ia32_orps (v4sf, v4sf)
v4sf __builtin_ia32_xorps (v4sf, v4sf)
v4sf __builtin_ia32_movss (v4sf, v4sf)
v4sf __builtin_ia32_movhlps (v4sf, v4sf)
v4sf __builtin_ia32_movlhps (v4sf, v4sf)
v4sf __builtin_ia32_unpckhps (v4sf, v4sf)
v4sf __builtin_ia32_unpcklps (v4sf, v4sf)
v4sf __builtin_ia32_cvtpi2ps (v4sf, v2si)
v4sf __builtin_ia32_cvtsi2ss (v4sf, int)
v2si __builtin_ia32_cvtps2pi (v4sf)
int __builtin_ia32_cvtss2si (v4sf)
v2si __builtin_ia32_cvttps2pi (v4sf)
int __builtin_ia32_cvttss2si (v4sf)
v4sf __builtin_ia32_rcpps (v4sf)
v4sf __builtin_ia32_rsqrtps (v4sf)
v4sf __builtin_ia32_sqrtps (v4sf)
v4sf __builtin_ia32_rcpss (v4sf)
v4sf __builtin_ia32_rsqrtss (v4sf)
v4sf __builtin_ia32_sqrtss (v4sf)
v4sf __builtin_ia32_shufps (v4sf, v4sf, int)
void __builtin_ia32_movntps (float *, v4sf)
int __builtin_ia32_movmskps (v4sf)
```

The following built-in functions are available when '`-msse`' is used.

`v4sf __builtin_ia32_loadaps (float *)`
: Generates the `movaps` machine instruction as a load from memory.

`void __builtin_ia32_storeaps (float *, v4sf)`
: Generates the `movaps` machine instruction as a store to memory.

`v4sf __builtin_ia32_loadups (float *)`
: Generates the `movups` machine instruction as a load from memory.

`void __builtin_ia32_storeups (float *, v4sf)`
: Generates the `movups` machine instruction as a store to memory.

`v4sf __builtin_ia32_loadsss (float *)`
: Generates the `movss` machine instruction as a load from memory.

`void __builtin_ia32_storess (float *, v4sf)`
: Generates the `movss` machine instruction as a store to memory.

`v4sf __builtin_ia32_loadhps (v4sf, const v2sf *)`
: Generates the `movhps` machine instruction as a load from memory.

`v4sf __builtin_ia32_loadlps (v4sf, const v2sf *)`
: Generates the `movlps` machine instruction as a load from memory

`void __builtin_ia32_storehps (v2sf *, v4sf)`
: Generates the `movhps` machine instruction as a store to memory.

`void __builtin_ia32_storelps (v2sf *, v4sf)`
: Generates the `movlps` machine instruction as a store to memory.

The following built-in functions are available when '`-msse2`' is used. All of them generate the machine instruction that is part of the name.

```
int __builtin_ia32_comisdeq (v2df, v2df)
int __builtin_ia32_comisdlt (v2df, v2df)
int __builtin_ia32_comisdle (v2df, v2df)
int __builtin_ia32_comisdgt (v2df, v2df)
int __builtin_ia32_comisdge (v2df, v2df)
int __builtin_ia32_comisdneq (v2df, v2df)
int __builtin_ia32_ucomisdeq (v2df, v2df)
int __builtin_ia32_ucomisdlt (v2df, v2df)
int __builtin_ia32_ucomisdle (v2df, v2df)
int __builtin_ia32_ucomisdgt (v2df, v2df)
int __builtin_ia32_ucomisdge (v2df, v2df)
int __builtin_ia32_ucomisdneq (v2df, v2df)
v2df __builtin_ia32_cmpeqpd (v2df, v2df)
v2df __builtin_ia32_cmpltpd (v2df, v2df)
v2df __builtin_ia32_cmplepd (v2df, v2df)
v2df __builtin_ia32_cmpgtpd (v2df, v2df)
v2df __builtin_ia32_cmpgepd (v2df, v2df)
v2df __builtin_ia32_cmpunordpd (v2df, v2df)
v2df __builtin_ia32_cmpneqpd (v2df, v2df)
v2df __builtin_ia32_cmpnltpd (v2df, v2df)
v2df __builtin_ia32_cmpnlepd (v2df, v2df)
v2df __builtin_ia32_cmpngtpd (v2df, v2df)
v2df __builtin_ia32_cmpngepd (v2df, v2df)
v2df __builtin_ia32_cmpordpd (v2df, v2df)
```

```
v2df __builtin_ia32_cmpeqsd (v2df, v2df)
v2df __builtin_ia32_cmpltsd (v2df, v2df)
v2df __builtin_ia32_cmplesd (v2df, v2df)
v2df __builtin_ia32_cmpunordsd (v2df, v2df)
v2df __builtin_ia32_cmpneqsd (v2df, v2df)
v2df __builtin_ia32_cmpnltsd (v2df, v2df)
v2df __builtin_ia32_cmpnlesd (v2df, v2df)
v2df __builtin_ia32_cmpordsd (v2df, v2df)
v2di __builtin_ia32_paddq (v2di, v2di)
v2di __builtin_ia32_psubq (v2di, v2di)
v2df __builtin_ia32_addpd (v2df, v2df)
v2df __builtin_ia32_subpd (v2df, v2df)
v2df __builtin_ia32_mulpd (v2df, v2df)
v2df __builtin_ia32_divpd (v2df, v2df)
v2df __builtin_ia32_addsd (v2df, v2df)
v2df __builtin_ia32_subsd (v2df, v2df)
v2df __builtin_ia32_mulsd (v2df, v2df)
v2df __builtin_ia32_divsd (v2df, v2df)
v2df __builtin_ia32_minpd (v2df, v2df)
v2df __builtin_ia32_maxpd (v2df, v2df)
v2df __builtin_ia32_minsd (v2df, v2df)
v2df __builtin_ia32_maxsd (v2df, v2df)
v2df __builtin_ia32_andpd (v2df, v2df)
v2df __builtin_ia32_andnpd (v2df, v2df)
v2df __builtin_ia32_orpd (v2df, v2df)
v2df __builtin_ia32_xorpd (v2df, v2df)
v2df __builtin_ia32_movsd (v2df, v2df)
v2df __builtin_ia32_unpckhpd (v2df, v2df)
v2df __builtin_ia32_unpcklpd (v2df, v2df)
v16qi __builtin_ia32_paddb128 (v16qi, v16qi)
v8hi __builtin_ia32_paddw128 (v8hi, v8hi)
v4si __builtin_ia32_paddd128 (v4si, v4si)
v2di __builtin_ia32_paddq128 (v2di, v2di)
v16qi __builtin_ia32_psubb128 (v16qi, v16qi)
v8hi __builtin_ia32_psubw128 (v8hi, v8hi)
v4si __builtin_ia32_psubd128 (v4si, v4si)
v2di __builtin_ia32_psubq128 (v2di, v2di)
v8hi __builtin_ia32_pmullw128 (v8hi, v8hi)
v8hi __builtin_ia32_pmulhw128 (v8hi, v8hi)
v2di __builtin_ia32_pand128 (v2di, v2di)
v2di __builtin_ia32_pandn128 (v2di, v2di)
v2di __builtin_ia32_por128 (v2di, v2di)
v2di __builtin_ia32_pxor128 (v2di, v2di)
v16qi __builtin_ia32_pavgb128 (v16qi, v16qi)
v8hi __builtin_ia32_pavgw128 (v8hi, v8hi)
v16qi __builtin_ia32_pcmpeqb128 (v16qi, v16qi)
v8hi __builtin_ia32_pcmpeqw128 (v8hi, v8hi)
v4si __builtin_ia32_pcmpeqd128 (v4si, v4si)
v16qi __builtin_ia32_pcmpgtb128 (v16qi, v16qi)
v8hi __builtin_ia32_pcmpgtw128 (v8hi, v8hi)
v4si __builtin_ia32_pcmpgtd128 (v4si, v4si)
v16qi __builtin_ia32_pmaxub128 (v16qi, v16qi)
v8hi __builtin_ia32_pmaxsw128 (v8hi, v8hi)
v16qi __builtin_ia32_pminub128 (v16qi, v16qi)
v8hi __builtin_ia32_pminsw128 (v8hi, v8hi)
v16qi __builtin_ia32_punpckhbw128 (v16qi, v16qi)
v8hi __builtin_ia32_punpckhwd128 (v8hi, v8hi)
v4si __builtin_ia32_punpckhdq128 (v4si, v4si)
```

```
v2di __builtin_ia32_punpckhqdq128 (v2di, v2di)
v16qi __builtin_ia32_punpcklbw128 (v16qi, v16qi)
v8hi __builtin_ia32_punpcklwd128 (v8hi, v8hi)
v4si __builtin_ia32_punpckldq128 (v4si, v4si)
v2di __builtin_ia32_punpcklqdq128 (v2di, v2di)
v16qi __builtin_ia32_packsswb128 (v8hi, v8hi)
v8hi __builtin_ia32_packssdw128 (v4si, v4si)
v16qi __builtin_ia32_packuswb128 (v8hi, v8hi)
v8hi __builtin_ia32_pmulhuw128 (v8hi, v8hi)
void __builtin_ia32_maskmovdqu (v16qi, v16qi)
v2df __builtin_ia32_loadupd (double *)
void __builtin_ia32_storeupd (double *, v2df)
v2df __builtin_ia32_loadhpd (v2df, double const *)
v2df __builtin_ia32_loadlpd (v2df, double const *)
int __builtin_ia32_movmskpd (v2df)
int __builtin_ia32_pmovmskb128 (v16qi)
void __builtin_ia32_movnti (int *, int)
void __builtin_ia32_movnti64 (long long int *, long long int)
void __builtin_ia32_movntpd (double *, v2df)
void __builtin_ia32_movntdq (v2df *, v2df)
v4si __builtin_ia32_pshufd (v4si, int)
v8hi __builtin_ia32_pshuflw (v8hi, int)
v8hi __builtin_ia32_pshufhw (v8hi, int)
v2di __builtin_ia32_psadbw128 (v16qi, v16qi)
v2df __builtin_ia32_sqrtpd (v2df)
v2df __builtin_ia32_sqrtsd (v2df)
v2df __builtin_ia32_shufpd (v2df, v2df, int)
v2df __builtin_ia32_cvtdq2pd (v4si)
v4sf __builtin_ia32_cvtdq2ps (v4si)
v4si __builtin_ia32_cvtpd2dq (v2df)
v2si __builtin_ia32_cvtpd2pi (v2df)
v4sf __builtin_ia32_cvtpd2ps (v2df)
v4si __builtin_ia32_cvttpd2dq (v2df)
v2si __builtin_ia32_cvttpd2pi (v2df)
v2df __builtin_ia32_cvtpi2pd (v2si)
int __builtin_ia32_cvtsd2si (v2df)
int __builtin_ia32_cvttsd2si (v2df)
long long __builtin_ia32_cvtsd2si64 (v2df)
long long __builtin_ia32_cvttsd2si64 (v2df)
v4si __builtin_ia32_cvtps2dq (v4sf)
v2df __builtin_ia32_cvtps2pd (v4sf)
v4si __builtin_ia32_cvttps2dq (v4sf)
v2df __builtin_ia32_cvtsi2sd (v2df, int)
v2df __builtin_ia32_cvtsi642sd (v2df, long long)
v4sf __builtin_ia32_cvtsd2ss (v4sf, v2df)
v2df __builtin_ia32_cvtss2sd (v2df, v4sf)
void __builtin_ia32_clflush (const void *)
void __builtin_ia32_lfence (void)
void __builtin_ia32_mfence (void)
v16qi __builtin_ia32_loaddqu (const char *)
void __builtin_ia32_storedqu (char *, v16qi)
v1di __builtin_ia32_pmuludq (v2si, v2si)
v2di __builtin_ia32_pmuludq128 (v4si, v4si)
v8hi __builtin_ia32_psllw128 (v8hi, v8hi)
v4si __builtin_ia32_pslld128 (v4si, v4si)
v2di __builtin_ia32_psllq128 (v2di, v2di)
v8hi __builtin_ia32_psrlw128 (v8hi, v8hi)
v4si __builtin_ia32_psrld128 (v4si, v4si)
```

```
v2di __builtin_ia32_psrlq128 (v2di, v2di)
v8hi __builtin_ia32_psraw128 (v8hi, v8hi)
v4si __builtin_ia32_psrad128 (v4si, v4si)
v2di __builtin_ia32_pslldqi128 (v2di, int)
v8hi __builtin_ia32_psllwi128 (v8hi, int)
v4si __builtin_ia32_pslldi128 (v4si, int)
v2di __builtin_ia32_psllqi128 (v2di, int)
v2di __builtin_ia32_psrldqi128 (v2di, int)
v8hi __builtin_ia32_psrlwi128 (v8hi, int)
v4si __builtin_ia32_psrldi128 (v4si, int)
v2di __builtin_ia32_psrlqi128 (v2di, int)
v8hi __builtin_ia32_psrawi128 (v8hi, int)
v4si __builtin_ia32_psradi128 (v4si, int)
v4si __builtin_ia32_pmaddwd128 (v8hi, v8hi)
v2di __builtin_ia32_movq128 (v2di)
```

The following built-in functions are available when '`-msse3`' is used. All of them generate the machine instruction that is part of the name.

```
v2df __builtin_ia32_addsubpd (v2df, v2df)
v4sf __builtin_ia32_addsubps (v4sf, v4sf)
v2df __builtin_ia32_haddpd (v2df, v2df)
v4sf __builtin_ia32_haddps (v4sf, v4sf)
v2df __builtin_ia32_hsubpd (v2df, v2df)
v4sf __builtin_ia32_hsubps (v4sf, v4sf)
v16qi __builtin_ia32_lddqu (char const *)
void __builtin_ia32_monitor (void *, unsigned int, unsigned int)
v2df __builtin_ia32_movddup (v2df)
v4sf __builtin_ia32_movshdup (v4sf)
v4sf __builtin_ia32_movsldup (v4sf)
void __builtin_ia32_mwait (unsigned int, unsigned int)
```

The following built-in functions are available when '`-msse3`' is used.

`v2df __builtin_ia32_loadddup (double const *)`
: Generates the `movddup` machine instruction as a load from memory.

The following built-in functions are available when '`-mssse3`' is used. All of them generate the machine instruction that is part of the name with MMX registers.

```
v2si __builtin_ia32_phaddd (v2si, v2si)
v4hi __builtin_ia32_phaddw (v4hi, v4hi)
v4hi __builtin_ia32_phaddsw (v4hi, v4hi)
v2si __builtin_ia32_phsubd (v2si, v2si)
v4hi __builtin_ia32_phsubw (v4hi, v4hi)
v4hi __builtin_ia32_phsubsw (v4hi, v4hi)
v4hi __builtin_ia32_pmaddubsw (v8qi, v8qi)
v4hi __builtin_ia32_pmulhrsw (v4hi, v4hi)
v8qi __builtin_ia32_pshufb (v8qi, v8qi)
v8qi __builtin_ia32_psignb (v8qi, v8qi)
v2si __builtin_ia32_psignd (v2si, v2si)
v4hi __builtin_ia32_psignw (v4hi, v4hi)
v1di __builtin_ia32_palignr (v1di, v1di, int)
v8qi __builtin_ia32_pabsb (v8qi)
v2si __builtin_ia32_pabsd (v2si)
v4hi __builtin_ia32_pabsw (v4hi)
```

The following built-in functions are available when '`-mssse3`' is used. All of them generate the machine instruction that is part of the name with SSE registers.

```
v4si __builtin_ia32_phaddd128 (v4si, v4si)
v8hi __builtin_ia32_phaddw128 (v8hi, v8hi)
```

```
v8hi __builtin_ia32_phaddsw128 (v8hi, v8hi)
v4si __builtin_ia32_phsubd128 (v4si, v4si)
v8hi __builtin_ia32_phsubw128 (v8hi, v8hi)
v8hi __builtin_ia32_phsubsw128 (v8hi, v8hi)
v8hi __builtin_ia32_pmaddubsw128 (v16qi, v16qi)
v8hi __builtin_ia32_pmulhrsw128 (v8hi, v8hi)
v16qi __builtin_ia32_pshufb128 (v16qi, v16qi)
v16qi __builtin_ia32_psignb128 (v16qi, v16qi)
v4si __builtin_ia32_psignd128 (v4si, v4si)
v8hi __builtin_ia32_psignw128 (v8hi, v8hi)
v2di __builtin_ia32_palignr128 (v2di, v2di, int)
v16qi __builtin_ia32_pabsb128 (v16qi)
v4si __builtin_ia32_pabsd128 (v4si)
v8hi __builtin_ia32_pabsw128 (v8hi)
```

The following built-in functions are available when '`-msse4.1`' is used. All of them generate the machine instruction that is part of the name.

```
v2df __builtin_ia32_blendpd (v2df, v2df, const int)
v4sf __builtin_ia32_blendps (v4sf, v4sf, const int)
v2df __builtin_ia32_blendvpd (v2df, v2df, v2df)
v4sf __builtin_ia32_blendvps (v4sf, v4sf, v4sf)
v2df __builtin_ia32_dppd (v2df, v2df, const int)
v4sf __builtin_ia32_dpps (v4sf, v4sf, const int)
v4sf __builtin_ia32_insertps128 (v4sf, v4sf, const int)
v2di __builtin_ia32_movntdqa (v2di *);
v16qi __builtin_ia32_mpsadbw128 (v16qi, v16qi, const int)
v8hi __builtin_ia32_packusdw128 (v4si, v4si)
v16qi __builtin_ia32_pblendvb128 (v16qi, v16qi, v16qi)
v8hi __builtin_ia32_pblendw128 (v8hi, v8hi, const int)
v2di __builtin_ia32_pcmpeqq (v2di, v2di)
v8hi __builtin_ia32_phminposuw128 (v8hi)
v16qi __builtin_ia32_pmaxsb128 (v16qi, v16qi)
v4si __builtin_ia32_pmaxsd128 (v4si, v4si)
v4si __builtin_ia32_pmaxud128 (v4si, v4si)
v8hi __builtin_ia32_pmaxuw128 (v8hi, v8hi)
v16qi __builtin_ia32_pminsb128 (v16qi, v16qi)
v4si __builtin_ia32_pminsd128 (v4si, v4si)
v4si __builtin_ia32_pminud128 (v4si, v4si)
v8hi __builtin_ia32_pminuw128 (v8hi, v8hi)
v4si __builtin_ia32_pmovsxbd128 (v16qi)
v2di __builtin_ia32_pmovsxbq128 (v16qi)
v8hi __builtin_ia32_pmovsxbw128 (v16qi)
v2di __builtin_ia32_pmovsxdq128 (v4si)
v4si __builtin_ia32_pmovsxwd128 (v8hi)
v2di __builtin_ia32_pmovsxwq128 (v8hi)
v4si __builtin_ia32_pmovzxbd128 (v16qi)
v2di __builtin_ia32_pmovzxbq128 (v16qi)
v8hi __builtin_ia32_pmovzxbw128 (v16qi)
v2di __builtin_ia32_pmovzxdq128 (v4si)
v4si __builtin_ia32_pmovzxwd128 (v8hi)
v2di __builtin_ia32_pmovzxwq128 (v8hi)
v2di __builtin_ia32_pmuldq128 (v4si, v4si)
v4si __builtin_ia32_pmulld128 (v4si, v4si)
int __builtin_ia32_ptestc128 (v2di, v2di)
int __builtin_ia32_ptestnzc128 (v2di, v2di)
int __builtin_ia32_ptestz128 (v2di, v2di)
v2df __builtin_ia32_roundpd (v2df, const int)
v4sf __builtin_ia32_roundps (v4sf, const int)
```

```
v2df __builtin_ia32_roundsd (v2df, v2df, const int)
v4sf __builtin_ia32_roundss (v4sf, v4sf, const int)
```

The following built-in functions are available when '`-msse4.1`' is used.

`v4sf __builtin_ia32_vec_set_v4sf (v4sf, float, const int)`
: Generates the `insertps` machine instruction.

`int __builtin_ia32_vec_ext_v16qi (v16qi, const int)`
: Generates the `pextrb` machine instruction.

`v16qi __builtin_ia32_vec_set_v16qi (v16qi, int, const int)`
: Generates the `pinsrb` machine instruction.

`v4si __builtin_ia32_vec_set_v4si (v4si, int, const int)`
: Generates the `pinsrd` machine instruction.

`v2di __builtin_ia32_vec_set_v2di (v2di, long long, const int)`
: Generates the `pinsrq` machine instruction in 64bit mode.

The following built-in functions are changed to generate new SSE4.1 instructions when '`-msse4.1`' is used.

`float __builtin_ia32_vec_ext_v4sf (v4sf, const int)`
: Generates the `extractps` machine instruction.

`int __builtin_ia32_vec_ext_v4si (v4si, const int)`
: Generates the `pextrd` machine instruction.

`long long __builtin_ia32_vec_ext_v2di (v2di, const int)`
: Generates the `pextrq` machine instruction in 64bit mode.

The following built-in functions are available when '`-msse4.2`' is used. All of them generate the machine instruction that is part of the name.

```
v16qi __builtin_ia32_pcmpestrm128 (v16qi, int, v16qi, int, const int)
int __builtin_ia32_pcmpestri128 (v16qi, int, v16qi, int, const int)
int __builtin_ia32_pcmpestria128 (v16qi, int, v16qi, int, const int)
int __builtin_ia32_pcmpestric128 (v16qi, int, v16qi, int, const int)
int __builtin_ia32_pcmpestrio128 (v16qi, int, v16qi, int, const int)
int __builtin_ia32_pcmpestris128 (v16qi, int, v16qi, int, const int)
int __builtin_ia32_pcmpestriz128 (v16qi, int, v16qi, int, const int)
v16qi __builtin_ia32_pcmpistrm128 (v16qi, v16qi, const int)
int __builtin_ia32_pcmpistri128 (v16qi, v16qi, const int)
int __builtin_ia32_pcmpistria128 (v16qi, v16qi, const int)
int __builtin_ia32_pcmpistric128 (v16qi, v16qi, const int)
int __builtin_ia32_pcmpistrio128 (v16qi, v16qi, const int)
int __builtin_ia32_pcmpistris128 (v16qi, v16qi, const int)
int __builtin_ia32_pcmpistriz128 (v16qi, v16qi, const int)
v2di __builtin_ia32_pcmpgtq (v2di, v2di)
```

The following built-in functions are available when '`-msse4.2`' is used.

`unsigned int __builtin_ia32_crc32qi (unsigned int, unsigned char)`
: Generates the `crc32b` machine instruction.

`unsigned int __builtin_ia32_crc32hi (unsigned int, unsigned short)`
: Generates the `crc32w` machine instruction.

`unsigned int __builtin_ia32_crc32si (unsigned int, unsigned int)`
Generates the `crc32l` machine instruction.

`unsigned long long __builtin_ia32_crc32di (unsigned long long, unsigned long long)`
Generates the `crc32q` machine instruction.

The following built-in functions are changed to generate new SSE4.2 instructions when '`-msse4.2`' is used.

`int __builtin_popcount (unsigned int)`
Generates the `popcntl` machine instruction.

`int __builtin_popcountl (unsigned long)`
Generates the `popcntl` or `popcntq` machine instruction, depending on the size of `unsigned long`.

`int __builtin_popcountll (unsigned long long)`
Generates the `popcntq` machine instruction.

The following built-in functions are available when '`-mavx`' is used. All of them generate the machine instruction that is part of the name.

```
v4df __builtin_ia32_addpd256 (v4df,v4df)
v8sf __builtin_ia32_addps256 (v8sf,v8sf)
v4df __builtin_ia32_addsubpd256 (v4df,v4df)
v8sf __builtin_ia32_addsubps256 (v8sf,v8sf)
v4df __builtin_ia32_andnpd256 (v4df,v4df)
v8sf __builtin_ia32_andnps256 (v8sf,v8sf)
v4df __builtin_ia32_andpd256 (v4df,v4df)
v8sf __builtin_ia32_andps256 (v8sf,v8sf)
v4df __builtin_ia32_blendpd256 (v4df,v4df,int)
v8sf __builtin_ia32_blendps256 (v8sf,v8sf,int)
v4df __builtin_ia32_blendvpd256 (v4df,v4df,v4df)
v8sf __builtin_ia32_blendvps256 (v8sf,v8sf,v8sf)
v2df __builtin_ia32_cmppd (v2df,v2df,int)
v4df __builtin_ia32_cmppd256 (v4df,v4df,int)
v4sf __builtin_ia32_cmpps (v4sf,v4sf,int)
v8sf __builtin_ia32_cmpps256 (v8sf,v8sf,int)
v2df __builtin_ia32_cmpsd (v2df,v2df,int)
v4sf __builtin_ia32_cmpss (v4sf,v4sf,int)
v4df __builtin_ia32_cvtdq2pd256 (v4si)
v8sf __builtin_ia32_cvtdq2ps256 (v8si)
v4si __builtin_ia32_cvtpd2dq256 (v4df)
v4sf __builtin_ia32_cvtpd2ps256 (v4df)
v8si __builtin_ia32_cvtps2dq256 (v8sf)
v4df __builtin_ia32_cvtps2pd256 (v4sf)
v4si __builtin_ia32_cvttpd2dq256 (v4df)
v8si __builtin_ia32_cvttps2dq256 (v8sf)
v4df __builtin_ia32_divpd256 (v4df,v4df)
v8sf __builtin_ia32_divps256 (v8sf,v8sf)
v8sf __builtin_ia32_dpps256 (v8sf,v8sf,int)
v4df __builtin_ia32_haddpd256 (v4df,v4df)
v8sf __builtin_ia32_haddps256 (v8sf,v8sf)
v4df __builtin_ia32_hsubpd256 (v4df,v4df)
v8sf __builtin_ia32_hsubps256 (v8sf,v8sf)
v32qi __builtin_ia32_lddqu256 (pcchar)
v32qi __builtin_ia32_loaddqu256 (pcchar)
```

```
v4df __builtin_ia32_loadupd256 (pcdouble)
v8sf __builtin_ia32_loadups256 (pcfloat)
v2df __builtin_ia32_maskloadpd (pcv2df,v2df)
v4df __builtin_ia32_maskloadpd256 (pcv4df,v4df)
v4sf __builtin_ia32_maskloadps (pcv4sf,v4sf)
v8sf __builtin_ia32_maskloadps256 (pcv8sf,v8sf)
void __builtin_ia32_maskstorepd (pv2df,v2df,v2df)
void __builtin_ia32_maskstorepd256 (pv4df,v4df,v4df)
void __builtin_ia32_maskstoreps (pv4sf,v4sf,v4sf)
void __builtin_ia32_maskstoreps256 (pv8sf,v8sf,v8sf)
v4df __builtin_ia32_maxpd256 (v4df,v4df)
v8sf __builtin_ia32_maxps256 (v8sf,v8sf)
v4df __builtin_ia32_minpd256 (v4df,v4df)
v8sf __builtin_ia32_minps256 (v8sf,v8sf)
v4df __builtin_ia32_movddup256 (v4df)
int __builtin_ia32_movmskpd256 (v4df)
int __builtin_ia32_movmskps256 (v8sf)
v8sf __builtin_ia32_movshdup256 (v8sf)
v8sf __builtin_ia32_movsldup256 (v8sf)
v4df __builtin_ia32_mulpd256 (v4df,v4df)
v8sf __builtin_ia32_mulps256 (v8sf,v8sf)
v4df __builtin_ia32_orpd256 (v4df,v4df)
v8sf __builtin_ia32_orps256 (v8sf,v8sf)
v2df __builtin_ia32_pd_pd256 (v4df)
v4df __builtin_ia32_pd256_pd (v2df)
v4sf __builtin_ia32_ps_ps256 (v8sf)
v8sf __builtin_ia32_ps256_ps (v4sf)
int __builtin_ia32_ptestc256 (v4di,v4di,ptest)
int __builtin_ia32_ptestnzc256 (v4di,v4di,ptest)
int __builtin_ia32_ptestz256 (v4di,v4di,ptest)
v8sf __builtin_ia32_rcpps256 (v8sf)
v4df __builtin_ia32_roundpd256 (v4df,int)
v8sf __builtin_ia32_roundps256 (v8sf,int)
v8sf __builtin_ia32_rsqrtps_nr256 (v8sf)
v8sf __builtin_ia32_rsqrtps256 (v8sf)
v4df __builtin_ia32_shufpd256 (v4df,v4df,int)
v8sf __builtin_ia32_shufps256 (v8sf,v8sf,int)
v4si __builtin_ia32_si_si256 (v8si)
v8si __builtin_ia32_si256_si (v4si)
v4df __builtin_ia32_sqrtpd256 (v4df)
v8sf __builtin_ia32_sqrtps_nr256 (v8sf)
v8sf __builtin_ia32_sqrtps256 (v8sf)
void __builtin_ia32_storedqu256 (pchar,v32qi)
void __builtin_ia32_storeupd256 (pdouble,v4df)
void __builtin_ia32_storeups256 (pfloat,v8sf)
v4df __builtin_ia32_subpd256 (v4df,v4df)
v8sf __builtin_ia32_subps256 (v8sf,v8sf)
v4df __builtin_ia32_unpckhpd256 (v4df,v4df)
v8sf __builtin_ia32_unpckhps256 (v8sf,v8sf)
v4df __builtin_ia32_unpcklpd256 (v4df,v4df)
v8sf __builtin_ia32_unpcklps256 (v8sf,v8sf)
v4df __builtin_ia32_vbroadcastf128_pd256 (pcv2df)
v8sf __builtin_ia32_vbroadcastf128_ps256 (pcv4sf)
v4df __builtin_ia32_vbroadcastsd256 (pcdouble)
v4sf __builtin_ia32_vbroadcastss (pcfloat)
v8sf __builtin_ia32_vbroadcastss256 (pcfloat)
v2df __builtin_ia32_vextractf128_pd256 (v4df,int)
v4sf __builtin_ia32_vextractf128_ps256 (v8sf,int)
```

```
v4si __builtin_ia32_vextractf128_si256 (v8si,int)
v4df __builtin_ia32_vinsertf128_pd256 (v4df,v2df,int)
v8sf __builtin_ia32_vinsertf128_ps256 (v8sf,v4sf,int)
v8si __builtin_ia32_vinsertf128_si256 (v8si,v4si,int)
v4df __builtin_ia32_vperm2f128_pd256 (v4df,v4df,int)
v8sf __builtin_ia32_vperm2f128_ps256 (v8sf,v8sf,int)
v8si __builtin_ia32_vperm2f128_si256 (v8si,v8si,int)
v2df __builtin_ia32_vpermil2pd (v2df,v2df,v2di,int)
v4df __builtin_ia32_vpermil2pd256 (v4df,v4df,v4di,int)
v4sf __builtin_ia32_vpermil2ps (v4sf,v4sf,v4si,int)
v8sf __builtin_ia32_vpermil2ps256 (v8sf,v8sf,v8si,int)
v2df __builtin_ia32_vpermilpd (v2df,int)
v4df __builtin_ia32_vpermilpd256 (v4df,int)
v4sf __builtin_ia32_vpermilps (v4sf,int)
v8sf __builtin_ia32_vpermilps256 (v8sf,int)
v2df __builtin_ia32_vpermilvarpd (v2df,v2di)
v4df __builtin_ia32_vpermilvarpd256 (v4df,v4di)
v4sf __builtin_ia32_vpermilvarps (v4sf,v4si)
v8sf __builtin_ia32_vpermilvarps256 (v8sf,v8si)
int __builtin_ia32_vtestcpd (v2df,v2df,ptest)
int __builtin_ia32_vtestcpd256 (v4df,v4df,ptest)
int __builtin_ia32_vtestcps (v4sf,v4sf,ptest)
int __builtin_ia32_vtestcps256 (v8sf,v8sf,ptest)
int __builtin_ia32_vtestnzcpd (v2df,v2df,ptest)
int __builtin_ia32_vtestnzcpd256 (v4df,v4df,ptest)
int __builtin_ia32_vtestnzcps (v4sf,v4sf,ptest)
int __builtin_ia32_vtestnzcps256 (v8sf,v8sf,ptest)
int __builtin_ia32_vtestzpd (v2df,v2df,ptest)
int __builtin_ia32_vtestzpd256 (v4df,v4df,ptest)
int __builtin_ia32_vtestzps (v4sf,v4sf,ptest)
int __builtin_ia32_vtestzps256 (v8sf,v8sf,ptest)
void __builtin_ia32_vzeroall (void)
void __builtin_ia32_vzeroupper (void)
v4df __builtin_ia32_xorpd256 (v4df,v4df)
v8sf __builtin_ia32_xorps256 (v8sf,v8sf)
```

The following built-in functions are available when '`-mavx2`' is used. All of them generate the machine instruction that is part of the name.

```
v32qi __builtin_ia32_mpsadbw256 (v32qi,v32qi,v32qi,int)
v32qi __builtin_ia32_pabsb256 (v32qi)
v16hi __builtin_ia32_pabsw256 (v16hi)
v8si __builtin_ia32_pabsd256 (v8si)
v16hi builtin_ia32_packssdw256 (v8si,v8si)
v32qi __builtin_ia32_packsswb256 (v16hi,v16hi)
v16hi __builtin_ia32_packusdw256 (v8si,v8si)
v32qi __builtin_ia32_packuswb256 (v16hi,v16hi)
v32qi__builtin_ia32_paddb256 (v32qi,v32qi)
v16hi __builtin_ia32_paddw256 (v16hi,v16hi)
v8si __builtin_ia32_paddd256 (v8si,v8si)
v4di __builtin_ia32_paddq256 (v4di,v4di)
v32qi __builtin_ia32_paddsb256 (v32qi,v32qi)
v16hi __builtin_ia32_paddsw256 (v16hi,v16hi)
v32qi __builtin_ia32_paddusb256 (v32qi,v32qi)
v16hi __builtin_ia32_paddusw256 (v16hi,v16hi)
v4di __builtin_ia32_palignr256 (v4di,v4di,int)
v4di __builtin_ia32_andsi256 (v4di,v4di)
v4di __builtin_ia32_andnotsi256 (v4di,v4di)
v32qi__builtin_ia32_pavgb256 (v32qi,v32qi)
```

```
v16hi __builtin_ia32_pavgw256 (v16hi,v16hi)
v32qi __builtin_ia32_pblendvb256 (v32qi,v32qi,v32qi)
v16hi __builtin_ia32_pblendw256 (v16hi,v16hi,int)
v32qi __builtin_ia32_pcmpeqb256 (v32qi,v32qi)
v16hi __builtin_ia32_pcmpeqw256 (v16hi,v16hi)
v8si __builtin_ia32_pcmpeqd256 (c8si,v8si)
v4di __builtin_ia32_pcmpeqq256 (v4di,v4di)
v32qi __builtin_ia32_pcmpgtb256 (v32qi,v32qi)
v16hi __builtin_ia32_pcmpgtw256 (16hi,v16hi)
v8si __builtin_ia32_pcmpgtd256 (v8si,v8si)
v4di __builtin_ia32_pcmpgtq256 (v4di,v4di)
v16hi __builtin_ia32_phaddw256 (v16hi,v16hi)
v8si __builtin_ia32_phaddd256 (v8si,v8si)
v16hi __builtin_ia32_phaddsw256 (v16hi,v16hi)
v16hi __builtin_ia32_phsubw256 (v16hi,v16hi)
v8si __builtin_ia32_phsubd256 (v8si,v8si)
v16hi __builtin_ia32_phsubsw256 (v16hi,v16hi)
v32qi __builtin_ia32_pmaddubsw256 (v32qi,v32qi)
v16hi __builtin_ia32_pmaddwd256 (v16hi,v16hi)
v32qi __builtin_ia32_pmaxsb256 (v32qi,v32qi)
v16hi __builtin_ia32_pmaxsw256 (v16hi,v16hi)
v8si __builtin_ia32_pmaxsd256 (v8si,v8si)
v32qi __builtin_ia32_pmaxub256 (v32qi,v32qi)
v16hi __builtin_ia32_pmaxuw256 (v16hi,v16hi)
v8si __builtin_ia32_pmaxud256 (v8si,v8si)
v32qi __builtin_ia32_pminsb256 (v32qi,v32qi)
v16hi __builtin_ia32_pminsw256 (v16hi,v16hi)
v8si __builtin_ia32_pminsd256 (v8si,v8si)
v32qi __builtin_ia32_pminub256 (v32qi,v32qi)
v16hi __builtin_ia32_pminuw256 (v16hi,v16hi)
v8si __builtin_ia32_pminud256 (v8si,v8si)
int __builtin_ia32_pmovmskb256 (v32qi)
v16hi __builtin_ia32_pmovsxbw256 (v16qi)
v8si __builtin_ia32_pmovsxbd256 (v16qi)
v4di __builtin_ia32_pmovsxbq256 (v16qi)
v8si __builtin_ia32_pmovsxwd256 (v8hi)
v4di __builtin_ia32_pmovsxwq256 (v8hi)
v4di __builtin_ia32_pmovsxdq256 (v4si)
v16hi __builtin_ia32_pmovzxbw256 (v16qi)
v8si __builtin_ia32_pmovzxbd256 (v16qi)
v4di __builtin_ia32_pmovzxbq256 (v16qi)
v8si __builtin_ia32_pmovzxwd256 (v8hi)
v4di __builtin_ia32_pmovzxwq256 (v8hi)
v4di __builtin_ia32_pmovzxdq256 (v4si)
v4di __builtin_ia32_pmuldq256 (v8si,v8si)
v16hi __builtin_ia32_pmulhrsw256 (v16hi, v16hi)
v16hi __builtin_ia32_pmulhuw256 (v16hi,v16hi)
v16hi __builtin_ia32_pmulhw256 (v16hi,v16hi)
v16hi __builtin_ia32_pmullw256 (v16hi,v16hi)
v8si __builtin_ia32_pmulld256 (v8si,v8si)
v4di __builtin_ia32_pmuludq256 (v8si,v8si)
v4di __builtin_ia32_por256 (v4di,v4di)
v16hi __builtin_ia32_psadbw256 (v32qi,v32qi)
v32qi __builtin_ia32_pshufb256 (v32qi,v32qi)
v8si __builtin_ia32_pshufd256 (v8si,int)
v16hi __builtin_ia32_pshufhw256 (v16hi,int)
v16hi __builtin_ia32_pshuflw256 (v16hi,int)
v32qi __builtin_ia32_psignb256 (v32qi,v32qi)
```

```
v16hi __builtin_ia32_psignw256 (v16hi,v16hi)
v8si __builtin_ia32_psignd256 (v8si,v8si)
v4di __builtin_ia32_pslldqi256 (v4di,int)
v16hi __builtin_ia32_psllwi256 (16hi,int)
v16hi __builtin_ia32_psllw256(v16hi,v8hi)
v8si __builtin_ia32_pslldi256 (v8si,int)
v8si __builtin_ia32_pslld256(v8si,v4si)
v4di __builtin_ia32_psllqi256 (v4di,int)
v4di __builtin_ia32_psllq256(v4di,v2di)
v16hi __builtin_ia32_psrawi256 (v16hi,int)
v16hi __builtin_ia32_psraw256 (v16hi,v8hi)
v8si __builtin_ia32_psradi256 (v8si,int)
v8si __builtin_ia32_psrad256 (v8si,v4si)
v4di __builtin_ia32_psrldqi256 (v4di, int)
v16hi __builtin_ia32_psrlwi256 (v16hi,int)
v16hi __builtin_ia32_psrlw256 (v16hi,v8hi)
v8si __builtin_ia32_psrldi256 (v8si,int)
v8si __builtin_ia32_psrld256 (v8si,v4si)
v4di __builtin_ia32_psrlqi256 (v4di,int)
v4di __builtin_ia32_psrlq256(v4di,v2di)
v32qi __builtin_ia32_psubb256 (v32qi,v32qi)
v32hi __builtin_ia32_psubw256 (v16hi,v16hi)
v8si __builtin_ia32_psubd256 (v8si,v8si)
v4di __builtin_ia32_psubq256 (v4di,v4di)
v32qi __builtin_ia32_psubsb256 (v32qi,v32qi)
v16hi __builtin_ia32_psubsw256 (v16hi,v16hi)
v32qi __builtin_ia32_psubusb256 (v32qi,v32qi)
v16hi __builtin_ia32_psubusw256 (v16hi,v16hi)
v32qi __builtin_ia32_punpckhbw256 (v32qi,v32qi)
v16hi __builtin_ia32_punpckhwd256 (v16hi,v16hi)
v8si __builtin_ia32_punpckhdq256 (v8si,v8si)
v4di __builtin_ia32_punpckhqdq256 (v4di,v4di)
v32qi __builtin_ia32_punpcklbw256 (v32qi,v32qi)
v16hi __builtin_ia32_punpcklwd256 (v16hi,v16hi)
v8si __builtin_ia32_punpckldq256 (v8si,v8si)
v4di __builtin_ia32_punpcklqdq256 (v4di,v4di)
v4di __builtin_ia32_pxor256 (v4di,v4di)
v4di __builtin_ia32_movntdqa256 (pv4di)
v4sf __builtin_ia32_vbroadcastss_ps (v4sf)
v8sf __builtin_ia32_vbroadcastss_ps256 (v4sf)
v4df __builtin_ia32_vbroadcastsd_pd256 (v2df)
v4di __builtin_ia32_vbroadcastsi256 (v2di)
v4si __builtin_ia32_pblendd128 (v4si,v4si)
v8si __builtin_ia32_pblendd256 (v8si,v8si)
v32qi __builtin_ia32_pbroadcastb256 (v16qi)
v16hi __builtin_ia32_pbroadcastw256 (v8hi)
v8si __builtin_ia32_pbroadcastd256 (v4si)
v4di __builtin_ia32_pbroadcastq256 (v2di)
v16qi __builtin_ia32_pbroadcastb128 (v16qi)
v8hi __builtin_ia32_pbroadcastw128 (v8hi)
v4si __builtin_ia32_pbroadcastd128 (v4si)
v2di __builtin_ia32_pbroadcastq128 (v2di)
v8si __builtin_ia32_permvarsi256 (v8si,v8si)
v4df __builtin_ia32_permdf256 (v4df,int)
v8sf __builtin_ia32_permvarsf256 (v8sf,v8sf)
v4di __builtin_ia32_permdi256 (v4di,int)
v4di __builtin_ia32_permti256 (v4di,v4di,int)
v4di __builtin_ia32_extract128i256 (v4di,int)
```

```
v4di __builtin_ia32_insert128i256 (v4di,v2di,int)
v8si __builtin_ia32_maskloadd256 (pcv8si,v8si)
v4di __builtin_ia32_maskloadq256 (pcv4di,v4di)
v4si __builtin_ia32_maskloadd (pcv4si,v4si)
v2di __builtin_ia32_maskloadq (pcv2di,v2di)
void __builtin_ia32_maskstored256 (pv8si,v8si,v8si)
void __builtin_ia32_maskstoreq256 (pv4di,v4di,v4di)
void __builtin_ia32_maskstored (pv4si,v4si,v4si)
void __builtin_ia32_maskstoreq (pv2di,v2di,v2di)
v8si __builtin_ia32_psllv8si (v8si,v8si)
v4si __builtin_ia32_psllv4si (v4si,v4si)
v4di __builtin_ia32_psllv4di (v4di,v4di)
v2di __builtin_ia32_psllv2di (v2di,v2di)
v8si __builtin_ia32_psrav8si (v8si,v8si)
v4si __builtin_ia32_psrav4si (v4si,v4si)
v8si __builtin_ia32_psrlv8si (v8si,v8si)
v4si __builtin_ia32_psrlv4si (v4si,v4si)
v4di __builtin_ia32_psrlv4di (v4di,v4di)
v2di __builtin_ia32_psrlv2di (v2di,v2di)
v2df __builtin_ia32_gathersiv2df (v2df, pcdouble,v4si,v2df,int)
v4df __builtin_ia32_gathersiv4df (v4df, pcdouble,v4si,v4df,int)
v2df __builtin_ia32_gatherdiv2df (v2df, pcdouble,v2di,v2df,int)
v4df __builtin_ia32_gatherdiv4df (v4df, pcdouble,v4di,v4df,int)
v4sf __builtin_ia32_gathersiv4sf (v4sf, pcfloat,v4si,v4sf,int)
v8sf __builtin_ia32_gathersiv8sf (v8sf, pcfloat,v8si,v8sf,int)
v4sf __builtin_ia32_gatherdiv4sf (v4sf, pcfloat,v2di,v4sf,int)
v4sf __builtin_ia32_gatherdiv4sf256 (v4sf, pcfloat,v4di,v4sf,int)
v2di __builtin_ia32_gathersiv2di (v2di, pcint64,v4si,v2di,int)
v4di __builtin_ia32_gathersiv4di (v4di, pcint64,v4si,v4di,int)
v2di __builtin_ia32_gatherdiv2di (v2di, pcint64,v2di,v2di,int)
v4di __builtin_ia32_gatherdiv4di (v4di, pcint64,v4di,v4di,int)
v4si __builtin_ia32_gathersiv4si (v4si, pcint,v4si,v4si,int)
v8si __builtin_ia32_gathersiv8si (v8si, pcint,v8si,v8si,int)
v4si __builtin_ia32_gatherdiv4si (v4si, pcint,v2di,v4si,int)
v4si __builtin_ia32_gatherdiv4si256 (v4si, pcint,v4di,v4si,int)
```

The following built-in functions are available when '`-maes`' is used. All of them generate the machine instruction that is part of the name.

```
v2di __builtin_ia32_aesenc128 (v2di, v2di)
v2di __builtin_ia32_aesenclast128 (v2di, v2di)
v2di __builtin_ia32_aesdec128 (v2di, v2di)
v2di __builtin_ia32_aesdeclast128 (v2di, v2di)
v2di __builtin_ia32_aeskeygenassist128 (v2di, const int)
v2di __builtin_ia32_aesimc128 (v2di)
```

The following built-in function is available when '`-mpclmul`' is used.

`v2di __builtin_ia32_pclmulqdq128 (v2di, v2di, const int)`
: Generates the `pclmulqdq` machine instruction.

The following built-in function is available when '`-mfsgsbase`' is used. All of them generate the machine instruction that is part of the name.

```
unsigned int __builtin_ia32_rdfsbase32 (void)
unsigned long long __builtin_ia32_rdfsbase64 (void)
unsigned int __builtin_ia32_rdgsbase32 (void)
unsigned long long __builtin_ia32_rdgsbase64 (void)
void _writefsbase_u32 (unsigned int)
void _writefsbase_u64 (unsigned long long)
void _writegsbase_u32 (unsigned int)
```

```
void _writegsbase_u64 (unsigned long long)
```

The following built-in function is available when '`-mrdrnd`' is used. All of them generate the machine instruction that is part of the name.

```
unsigned int __builtin_ia32_rdrand16_step (unsigned short *)
unsigned int __builtin_ia32_rdrand32_step (unsigned int *)
unsigned int __builtin_ia32_rdrand64_step (unsigned long long *)
```

The following built-in functions are available when '`-msse4a`' is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_movntsd (double *, v2df)
void __builtin_ia32_movntss (float *, v4sf)
v2di __builtin_ia32_extrq  (v2di, v16qi)
v2di __builtin_ia32_extrqi (v2di, const unsigned int, const unsigned int)
v2di __builtin_ia32_insertq (v2di, v2di)
v2di __builtin_ia32_insertqi (v2di, v2di, const unsigned int, const unsigned int)
```

The following built-in functions are available when '`-mxop`' is used.

```
v2df __builtin_ia32_vfrczpd (v2df)
v4sf __builtin_ia32_vfrczps (v4sf)
v2df __builtin_ia32_vfrczsd (v2df, v2df)
v4sf __builtin_ia32_vfrczss (v4sf, v4sf)
v4df __builtin_ia32_vfrczpd256 (v4df)
v8sf __builtin_ia32_vfrczps256 (v8sf)
v2di __builtin_ia32_vpcmov (v2di, v2di, v2di)
v2di __builtin_ia32_vpcmov_v2di (v2di, v2di, v2di)
v4si __builtin_ia32_vpcmov_v4si (v4si, v4si, v4si)
v8hi __builtin_ia32_vpcmov_v8hi (v8hi, v8hi, v8hi)
v16qi __builtin_ia32_vpcmov_v16qi (v16qi, v16qi, v16qi)
v2df __builtin_ia32_vpcmov_v2df (v2df, v2df, v2df)
v4sf __builtin_ia32_vpcmov_v4sf (v4sf, v4sf, v4sf)
v4di __builtin_ia32_vpcmov_v4di256 (v4di, v4di, v4di)
v8si __builtin_ia32_vpcmov_v8si256 (v8si, v8si, v8si)
v16hi __builtin_ia32_vpcmov_v16hi256 (v16hi, v16hi, v16hi)
v32qi __builtin_ia32_vpcmov_v32qi256 (v32qi, v32qi, v32qi)
v4df __builtin_ia32_vpcmov_v4df256 (v4df, v4df, v4df)
v8sf __builtin_ia32_vpcmov_v8sf256 (v8sf, v8sf, v8sf)
v16qi __builtin_ia32_vpcomeqb (v16qi, v16qi)
v8hi __builtin_ia32_vpcomeqw (v8hi, v8hi)
v4si __builtin_ia32_vpcomeqd (v4si, v4si)
v2di __builtin_ia32_vpcomeqq (v2di, v2di)
v16qi __builtin_ia32_vpcomequb (v16qi, v16qi)
v4si __builtin_ia32_vpcomequd (v4si, v4si)
v2di __builtin_ia32_vpcomequq (v2di, v2di)
v8hi __builtin_ia32_vpcomequw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomeqw (v8hi, v8hi)
v16qi __builtin_ia32_vpcomfalseb (v16qi, v16qi)
v4si __builtin_ia32_vpcomfalsed (v4si, v4si)
v2di __builtin_ia32_vpcomfalseq (v2di, v2di)
v16qi __builtin_ia32_vpcomfalseub (v16qi, v16qi)
v4si __builtin_ia32_vpcomfalseud (v4si, v4si)
v2di __builtin_ia32_vpcomfalseuq (v2di, v2di)
v8hi __builtin_ia32_vpcomfalseuw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomfalsew (v8hi, v8hi)
v16qi __builtin_ia32_vpcomgeb (v16qi, v16qi)
v4si __builtin_ia32_vpcomged (v4si, v4si)
v2di __builtin_ia32_vpcomgeq (v2di, v2di)
v16qi __builtin_ia32_vpcomgeub (v16qi, v16qi)
v4si __builtin_ia32_vpcomgeud (v4si, v4si)
```

```
v2di __builtin_ia32_vpcomgeuq (v2di, v2di)
v8hi __builtin_ia32_vpcomgeuw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomgew (v8hi, v8hi)
v16qi __builtin_ia32_vpcomgtb (v16qi, v16qi)
v4si __builtin_ia32_vpcomgtd (v4si, v4si)
v2di __builtin_ia32_vpcomgtq (v2di, v2di)
v16qi __builtin_ia32_vpcomgtub (v16qi, v16qi)
v4si __builtin_ia32_vpcomgtud (v4si, v4si)
v2di __builtin_ia32_vpcomgtuq (v2di, v2di)
v8hi __builtin_ia32_vpcomgtuw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomgtw (v8hi, v8hi)
v16qi __builtin_ia32_vpcomleb (v16qi, v16qi)
v4si __builtin_ia32_vpcomled (v4si, v4si)
v2di __builtin_ia32_vpcomleq (v2di, v2di)
v16qi __builtin_ia32_vpcomleub (v16qi, v16qi)
v4si __builtin_ia32_vpcomleud (v4si, v4si)
v2di __builtin_ia32_vpcomleuq (v2di, v2di)
v8hi __builtin_ia32_vpcomleuw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomlew (v8hi, v8hi)
v16qi __builtin_ia32_vpcomltb (v16qi, v16qi)
v4si __builtin_ia32_vpcomltd (v4si, v4si)
v2di __builtin_ia32_vpcomltq (v2di, v2di)
v16qi __builtin_ia32_vpcomltub (v16qi, v16qi)
v4si __builtin_ia32_vpcomltud (v4si, v4si)
v2di __builtin_ia32_vpcomltuq (v2di, v2di)
v8hi __builtin_ia32_vpcomltuw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomltw (v8hi, v8hi)
v16qi __builtin_ia32_vpcomneb (v16qi, v16qi)
v4si __builtin_ia32_vpcomned (v4si, v4si)
v2di __builtin_ia32_vpcomneq (v2di, v2di)
v16qi __builtin_ia32_vpcomneub (v16qi, v16qi)
v4si __builtin_ia32_vpcomneud (v4si, v4si)
v2di __builtin_ia32_vpcomneuq (v2di, v2di)
v8hi __builtin_ia32_vpcomneuw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomnew (v8hi, v8hi)
v16qi __builtin_ia32_vpcomtrueb (v16qi, v16qi)
v4si __builtin_ia32_vpcomtrued (v4si, v4si)
v2di __builtin_ia32_vpcomtrueq (v2di, v2di)
v16qi __builtin_ia32_vpcomtrueub (v16qi, v16qi)
v4si __builtin_ia32_vpcomtrueud (v4si, v4si)
v2di __builtin_ia32_vpcomtrueuq (v2di, v2di)
v8hi __builtin_ia32_vpcomtrueuw (v8hi, v8hi)
v8hi __builtin_ia32_vpcomtruew (v8hi, v8hi)
v4si __builtin_ia32_vphaddbd (v16qi)
v2di __builtin_ia32_vphaddbq (v16qi)
v8hi __builtin_ia32_vphaddbw (v16qi)
v2di __builtin_ia32_vphadddq (v4si)
v4si __builtin_ia32_vphaddubd (v16qi)
v2di __builtin_ia32_vphaddubq (v16qi)
v8hi __builtin_ia32_vphaddubw (v16qi)
v2di __builtin_ia32_vphaddudq (v4si)
v4si __builtin_ia32_vphadduwd (v8hi)
v2di __builtin_ia32_vphadduwq (v8hi)
v4si __builtin_ia32_vphaddwd (v8hi)
v2di __builtin_ia32_vphaddwq (v8hi)
v8hi __builtin_ia32_vphsubbw (v16qi)
v2di __builtin_ia32_vphsubdq (v4si)
v4si __builtin_ia32_vphsubwd (v8hi)
```

```
v4si __builtin_ia32_vpmacsdd (v4si, v4si, v4si)
v2di __builtin_ia32_vpmacsdqh (v4si, v4si, v2di)
v2di __builtin_ia32_vpmacsdql (v4si, v4si, v2di)
v4si __builtin_ia32_vpmacssdd (v4si, v4si, v4si)
v2di __builtin_ia32_vpmacssdqh (v4si, v4si, v2di)
v2di __builtin_ia32_vpmacssdql (v4si, v4si, v2di)
v4si __builtin_ia32_vpmacsswd (v8hi, v8hi, v4si)
v8hi __builtin_ia32_vpmacssww (v8hi, v8hi, v8hi)
v4si __builtin_ia32_vpmacswd (v8hi, v8hi, v4si)
v8hi __builtin_ia32_vpmacsww (v8hi, v8hi, v8hi)
v4si __builtin_ia32_vpmadcsswd (v8hi, v8hi, v4si)
v4si __builtin_ia32_vpmadcswd (v8hi, v8hi, v4si)
v16qi __builtin_ia32_vpperm (v16qi, v16qi, v16qi)
v16qi __builtin_ia32_vprotb (v16qi, v16qi)
v4si __builtin_ia32_vprotd (v4si, v4si)
v2di __builtin_ia32_vprotq (v2di, v2di)
v8hi __builtin_ia32_vprotw (v8hi, v8hi)
v16qi __builtin_ia32_vpshab (v16qi, v16qi)
v4si __builtin_ia32_vpshad (v4si, v4si)
v2di __builtin_ia32_vpshaq (v2di, v2di)
v8hi __builtin_ia32_vpshaw (v8hi, v8hi)
v16qi __builtin_ia32_vpshlb (v16qi, v16qi)
v4si __builtin_ia32_vpshld (v4si, v4si)
v2di __builtin_ia32_vpshlq (v2di, v2di)
v8hi __builtin_ia32_vpshlw (v8hi, v8hi)
```

The following built-in functions are available when '`-mfma4`' is used. All of them generate the machine instruction that is part of the name with MMX registers.

```
v2df __builtin_ia32_fmaddpd (v2df, v2df, v2df)
v4sf __builtin_ia32_fmaddps (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fmaddsd (v2df, v2df, v2df)
v4sf __builtin_ia32_fmaddss (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fmsubpd (v2df, v2df, v2df)
v4sf __builtin_ia32_fmsubps (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fmsubsd (v2df, v2df, v2df)
v4sf __builtin_ia32_fmsubss (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fnmaddpd (v2df, v2df, v2df)
v4sf __builtin_ia32_fnmaddps (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fnmaddsd (v2df, v2df, v2df)
v4sf __builtin_ia32_fnmaddss (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fnmsubpd (v2df, v2df, v2df)
v4sf __builtin_ia32_fnmsubps (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fnmsubsd (v2df, v2df, v2df)
v4sf __builtin_ia32_fnmsubss (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fmaddsubpd  (v2df, v2df, v2df)
v4sf __builtin_ia32_fmaddsubps  (v4sf, v4sf, v4sf)
v2df __builtin_ia32_fmsubaddpd  (v2df, v2df, v2df)
v4sf __builtin_ia32_fmsubaddps  (v4sf, v4sf, v4sf)
v4df __builtin_ia32_fmaddpd256 (v4df, v4df, v4df)
v8sf __builtin_ia32_fmaddps256 (v8sf, v8sf, v8sf)
v4df __builtin_ia32_fmsubpd256 (v4df, v4df, v4df)
v8sf __builtin_ia32_fmsubps256 (v8sf, v8sf, v8sf)
v4df __builtin_ia32_fnmaddpd256 (v4df, v4df, v4df)
v8sf __builtin_ia32_fnmaddps256 (v8sf, v8sf, v8sf)
v4df __builtin_ia32_fnmsubpd256 (v4df, v4df, v4df)
v8sf __builtin_ia32_fnmsubps256 (v8sf, v8sf, v8sf)
v4df __builtin_ia32_fmaddsubpd256 (v4df, v4df, v4df)
v8sf __builtin_ia32_fmaddsubps256 (v8sf, v8sf, v8sf)
```

```
v4df __builtin_ia32_fmsubaddpd256 (v4df, v4df, v4df)
v8sf __builtin_ia32_fmsubaddps256 (v8sf, v8sf, v8sf)
```

The following built-in functions are available when '`-mlwp`' is used.

```
void __builtin_ia32_llwpcb16 (void *);
void __builtin_ia32_llwpcb32 (void *);
void __builtin_ia32_llwpcb64 (void *);
void * __builtin_ia32_llwpcb16 (void);
void * __builtin_ia32_llwpcb32 (void);
void * __builtin_ia32_llwpcb64 (void);
void __builtin_ia32_lwpval16 (unsigned short, unsigned int, unsigned short)
void __builtin_ia32_lwpval32 (unsigned int, unsigned int, unsigned int)
void __builtin_ia32_lwpval64 (unsigned __int64, unsigned int, unsigned int)
unsigned char __builtin_ia32_lwpins16 (unsigned short, unsigned int, unsigned short)
unsigned char __builtin_ia32_lwpins32 (unsigned int, unsigned int, unsigned int)
unsigned char __builtin_ia32_lwpins64 (unsigned __int64, unsigned int, unsigned int)
```

The following built-in functions are available when '`-mbmi`' is used. All of them generate the machine instruction that is part of the name.

```
unsigned int __builtin_ia32_bextr_u32(unsigned int, unsigned int);
unsigned long long __builtin_ia32_bextr_u64 (unsigned long long, unsigned long long);
```

The following built-in functions are available when '`-mbmi2`' is used. All of them generate the machine instruction that is part of the name.

```
unsigned int _bzhi_u32 (unsigned int, unsigned int)
unsigned int _pdep_u32 (unsigned int, unsigned int)
unsigned int _pext_u32 (unsigned int, unsigned int)
unsigned long long _bzhi_u64 (unsigned long long, unsigned long long)
unsigned long long _pdep_u64 (unsigned long long, unsigned long long)
unsigned long long _pext_u64 (unsigned long long, unsigned long long)
```

The following built-in functions are available when '`-mlzcnt`' is used. All of them generate the machine instruction that is part of the name.

```
unsigned short __builtin_ia32_lzcnt_16(unsigned short);
unsigned int __builtin_ia32_lzcnt_u32(unsigned int);
unsigned long long __builtin_ia32_lzcnt_u64 (unsigned long long);
```

The following built-in functions are available when '`-mtbm`' is used. Both of them generate the immediate form of the bextr machine instruction.

```
unsigned int __builtin_ia32_bextri_u32 (unsigned int, const unsigned int);
unsigned long long __builtin_ia32_bextri_u64 (unsigned long long, const unsigned long long);
```

The following built-in functions are available when '`-m3dnow`' is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_femms (void)
v8qi __builtin_ia32_pavgusb (v8qi, v8qi)
v2si __builtin_ia32_pf2id (v2sf)
v2sf __builtin_ia32_pfacc (v2sf, v2sf)
v2sf __builtin_ia32_pfadd (v2sf, v2sf)
v2si __builtin_ia32_pfcmpeq (v2sf, v2sf)
v2si __builtin_ia32_pfcmpge (v2sf, v2sf)
v2si __builtin_ia32_pfcmpgt (v2sf, v2sf)
v2sf __builtin_ia32_pfmax (v2sf, v2sf)
v2sf __builtin_ia32_pfmin (v2sf, v2sf)
v2sf __builtin_ia32_pfmul (v2sf, v2sf)
v2sf __builtin_ia32_pfrcp (v2sf)
v2sf __builtin_ia32_pfrcpit1 (v2sf, v2sf)
```

```
v2sf __builtin_ia32_pfrcpit2 (v2sf, v2sf)
v2sf __builtin_ia32_pfrsqrt (v2sf)
v2sf __builtin_ia32_pfrsqrtit1 (v2sf, v2sf)
v2sf __builtin_ia32_pfsub (v2sf, v2sf)
v2sf __builtin_ia32_pfsubr (v2sf, v2sf)
v2sf __builtin_ia32_pi2fd (v2si)
v4hi __builtin_ia32_pmulhrw (v4hi, v4hi)
```

The following built-in functions are available when both '`-m3dnow`' and '`-march=athlon`' are used. All of them generate the machine instruction that is part of the name.

```
v2si __builtin_ia32_pf2iw (v2sf)
v2sf __builtin_ia32_pfnacc (v2sf, v2sf)
v2sf __builtin_ia32_pfpnacc (v2sf, v2sf)
v2sf __builtin_ia32_pi2fw (v2si)
v2sf __builtin_ia32_pswapdsf (v2sf)
v2si __builtin_ia32_pswapdsi (v2si)
```

6.55.8 MIPS DSP Built-in Functions

The MIPS DSP Application-Specific Extension (ASE) includes new instructions that are designed to improve the performance of DSP and media applications. It provides instructions that operate on packed 8-bit/16-bit integer data, Q7, Q15 and Q31 fractional data.

GCC supports MIPS DSP operations using both the generic vector extensions (see Section 6.49 [Vector Extensions], page 423) and a collection of MIPS-specific built-in functions. Both kinds of support are enabled by the '`-mdsp`' command-line option.

Revision 2 of the ASE was introduced in the second half of 2006. This revision adds extra instructions to the original ASE, but is otherwise backwards-compatible with it. You can select revision 2 using the command-line option '`-mdspr2`'; this option implies '`-mdsp`'.

The SCOUNT and POS bits of the DSP control register are global. The WRDSP, EXTPDP, EXTPDPV and MTHLIP instructions modify the SCOUNT and POS bits. During optimization, the compiler will not delete these instructions and it will not delete calls to functions containing these instructions.

At present, GCC only provides support for operations on 32-bit vectors. The vector type associated with 8-bit integer data is usually called `v4i8`, the vector type associated with Q7 is usually called `v4q7`, the vector type associated with 16-bit integer data is usually called `v2i16`, and the vector type associated with Q15 is usually called `v2q15`. They can be defined in C as follows:

```
typedef signed char v4i8 __attribute__ ((vector_size(4)));
typedef signed char v4q7 __attribute__ ((vector_size(4)));
typedef short v2i16 __attribute__ ((vector_size(4)));
typedef short v2q15 __attribute__ ((vector_size(4)));
```

`v4i8`, `v4q7`, `v2i16` and `v2q15` values are initialized in the same way as aggregates. For example:

```
v4i8 a = {1, 2, 3, 4};
v4i8 b;
b = (v4i8) {5, 6, 7, 8};

v2q15 c = {0x0fcb, 0x3a75};
v2q15 d;
d = (v2q15) {0.1234 * 0x1.0p15, 0.4567 * 0x1.0p15};
```

Note: The CPU's endianness determines the order in which values are packed. On little-endian targets, the first value is the least significant and the last value is the most significant.

The opposite order applies to big-endian targets. For example, the code above will set the lowest byte of `a` to `1` on little-endian targets and `4` on big-endian targets.

Note: Q7, Q15 and Q31 values must be initialized with their integer representation. As shown in this example, the integer representation of a Q7 value can be obtained by multiplying the fractional value by `0x1.0p7`. The equivalent for Q15 values is to multiply by `0x1.0p15`. The equivalent for Q31 values is to multiply by `0x1.0p31`.

The table below lists the `v4i8` and `v2q15` operations for which hardware support exists. `a` and `b` are `v4i8` values, and `c` and `d` are `v2q15` values.

C code	MIPS instruction
`a + b`	`addu.qb`
`c + d`	`addq.ph`
`a - b`	`subu.qb`
`c - d`	`subq.ph`

The table below lists the `v2i16` operation for which hardware support exists for the DSP ASE REV 2. `e` and `f` are `v2i16` values.

C code	MIPS instruction
`e * f`	`mul.ph`

It is easier to describe the DSP built-in functions if we first define the following types:

```
typedef int q31;
typedef int i32;
typedef unsigned int ui32;
typedef long long a64;
```

`q31` and `i32` are actually the same as `int`, but we use `q31` to indicate a Q31 fractional value and `i32` to indicate a 32-bit integer value. Similarly, `a64` is the same as `long long`, but we use `a64` to indicate values that will be placed in one of the four DSP accumulators (`$ac0`, `$ac1`, `$ac2` or `$ac3`).

Also, some built-in functions prefer or require immediate numbers as parameters, because the corresponding DSP instructions accept both immediate numbers and register operands, or accept immediate numbers only. The immediate parameters are listed as follows.

```
imm0_3: 0 to 3.
imm0_7: 0 to 7.
imm0_15: 0 to 15.
imm0_31: 0 to 31.
imm0_63: 0 to 63.
imm0_255: 0 to 255.
imm_n32_31: -32 to 31.
imm_n512_511: -512 to 511.
```

The following built-in functions map directly to a particular MIPS DSP instruction. Please refer to the architecture specification for details on what each instruction does.

```
v2q15 __builtin_mips_addq_ph (v2q15, v2q15)
v2q15 __builtin_mips_addq_s_ph (v2q15, v2q15)
q31 __builtin_mips_addq_s_w (q31, q31)
v4i8 __builtin_mips_addu_qb (v4i8, v4i8)
v4i8 __builtin_mips_addu_s_qb (v4i8, v4i8)
v2q15 __builtin_mips_subq_ph (v2q15, v2q15)
v2q15 __builtin_mips_subq_s_ph (v2q15, v2q15)
q31 __builtin_mips_subq_s_w (q31, q31)
v4i8 __builtin_mips_subu_qb (v4i8, v4i8)
v4i8 __builtin_mips_subu_s_qb (v4i8, v4i8)
```

```
i32 __builtin_mips_addsc (i32, i32)
i32 __builtin_mips_addwc (i32, i32)
i32 __builtin_mips_modsub (i32, i32)
i32 __builtin_mips_raddu_w_qb (v4i8)
v2q15 __builtin_mips_absq_s_ph (v2q15)
q31 __builtin_mips_absq_s_w (q31)
v4i8 __builtin_mips_precrq_qb_ph (v2q15, v2q15)
v2q15 __builtin_mips_precrq_ph_w (q31, q31)
v2q15 __builtin_mips_precrq_rs_ph_w (q31, q31)
v4i8 __builtin_mips_precrqu_s_qb_ph (v2q15, v2q15)
q31 __builtin_mips_preceq_w_phl (v2q15)
q31 __builtin_mips_preceq_w_phr (v2q15)
v2q15 __builtin_mips_precequ_ph_qbl (v4i8)
v2q15 __builtin_mips_precequ_ph_qbr (v4i8)
v2q15 __builtin_mips_precequ_ph_qbla (v4i8)
v2q15 __builtin_mips_precequ_ph_qbra (v4i8)
v2q15 __builtin_mips_preceu_ph_qbl (v4i8)
v2q15 __builtin_mips_preceu_ph_qbr (v4i8)
v2q15 __builtin_mips_preceu_ph_qbla (v4i8)
v2q15 __builtin_mips_preceu_ph_qbra (v4i8)
v4i8 __builtin_mips_shll_qb (v4i8, imm0_7)
v4i8 __builtin_mips_shll_qb (v4i8, i32)
v2q15 __builtin_mips_shll_ph (v2q15, imm0_15)
v2q15 __builtin_mips_shll_ph (v2q15, i32)
v2q15 __builtin_mips_shll_s_ph (v2q15, imm0_15)
v2q15 __builtin_mips_shll_s_ph (v2q15, i32)
q31 __builtin_mips_shll_s_w (q31, imm0_31)
q31 __builtin_mips_shll_s_w (q31, i32)
v4i8 __builtin_mips_shrl_qb (v4i8, imm0_7)
v4i8 __builtin_mips_shrl_qb (v4i8, i32)
v2q15 __builtin_mips_shra_ph (v2q15, imm0_15)
v2q15 __builtin_mips_shra_ph (v2q15, i32)
v2q15 __builtin_mips_shra_r_ph (v2q15, imm0_15)
v2q15 __builtin_mips_shra_r_ph (v2q15, i32)
q31 __builtin_mips_shra_r_w (q31, imm0_31)
q31 __builtin_mips_shra_r_w (q31, i32)
v2q15 __builtin_mips_muleu_s_ph_qbl (v4i8, v2q15)
v2q15 __builtin_mips_muleu_s_ph_qbr (v4i8, v2q15)
v2q15 __builtin_mips_mulq_rs_ph (v2q15, v2q15)
q31 __builtin_mips_muleq_s_w_phl (v2q15, v2q15)
q31 __builtin_mips_muleq_s_w_phr (v2q15, v2q15)
a64 __builtin_mips_dpau_h_qbl (a64, v4i8, v4i8)
a64 __builtin_mips_dpau_h_qbr (a64, v4i8, v4i8)
a64 __builtin_mips_dpsu_h_qbl (a64, v4i8, v4i8)
a64 __builtin_mips_dpsu_h_qbr (a64, v4i8, v4i8)
a64 __builtin_mips_dpaq_s_w_ph (a64, v2q15, v2q15)
a64 __builtin_mips_dpaq_sa_l_w (a64, q31, q31)
a64 __builtin_mips_dpsq_s_w_ph (a64, v2q15, v2q15)
a64 __builtin_mips_dpsq_sa_l_w (a64, q31, q31)
a64 __builtin_mips_mulsaq_s_w_ph (a64, v2q15, v2q15)
a64 __builtin_mips_maq_s_w_phl (a64, v2q15, v2q15)
a64 __builtin_mips_maq_s_w_phr (a64, v2q15, v2q15)
a64 __builtin_mips_maq_sa_w_phl (a64, v2q15, v2q15)
a64 __builtin_mips_maq_sa_w_phr (a64, v2q15, v2q15)
i32 __builtin_mips_bitrev (i32)
i32 __builtin_mips_insv (i32, i32)
v4i8 __builtin_mips_repl_qb (imm0_255)
v4i8 __builtin_mips_repl_qb (i32)
```

```
v2q15 __builtin_mips_repl_ph (imm_n512_511)
v2q15 __builtin_mips_repl_ph (i32)
void __builtin_mips_cmpu_eq_qb (v4i8, v4i8)
void __builtin_mips_cmpu_lt_qb (v4i8, v4i8)
void __builtin_mips_cmpu_le_qb (v4i8, v4i8)
i32 __builtin_mips_cmpgu_eq_qb (v4i8, v4i8)
i32 __builtin_mips_cmpgu_lt_qb (v4i8, v4i8)
i32 __builtin_mips_cmpgu_le_qb (v4i8, v4i8)
void __builtin_mips_cmp_eq_ph (v2q15, v2q15)
void __builtin_mips_cmp_lt_ph (v2q15, v2q15)
void __builtin_mips_cmp_le_ph (v2q15, v2q15)
v4i8 __builtin_mips_pick_qb (v4i8, v4i8)
v2q15 __builtin_mips_pick_ph (v2q15, v2q15)
v2q15 __builtin_mips_packrl_ph (v2q15, v2q15)
i32 __builtin_mips_extr_w (a64, imm0_31)
i32 __builtin_mips_extr_w (a64, i32)
i32 __builtin_mips_extr_r_w (a64, imm0_31)
i32 __builtin_mips_extr_s_h (a64, i32)
i32 __builtin_mips_extr_rs_w (a64, imm0_31)
i32 __builtin_mips_extr_rs_w (a64, i32)
i32 __builtin_mips_extr_s_h (a64, imm0_31)
i32 __builtin_mips_extr_r_w (a64, i32)
i32 __builtin_mips_extp (a64, imm0_31)
i32 __builtin_mips_extp (a64, i32)
i32 __builtin_mips_extpdp (a64, imm0_31)
i32 __builtin_mips_extpdp (a64, i32)
a64 __builtin_mips_shilo (a64, imm_n32_31)
a64 __builtin_mips_shilo (a64, i32)
a64 __builtin_mips_mthlip (a64, i32)
void __builtin_mips_wrdsp (i32, imm0_63)
i32 __builtin_mips_rddsp (imm0_63)
i32 __builtin_mips_lbux (void *, i32)
i32 __builtin_mips_lhx (void *, i32)
i32 __builtin_mips_lwx (void *, i32)
a64 __builtin_mips_ldx (void *, i32) [MIPS64 only]
i32 __builtin_mips_bposge32 (void)
a64 __builtin_mips_madd (a64, i32, i32);
a64 __builtin_mips_maddu (a64, ui32, ui32);
a64 __builtin_mips_msub (a64, i32, i32);
a64 __builtin_mips_msubu (a64, ui32, ui32);
a64 __builtin_mips_mult (i32, i32);
a64 __builtin_mips_multu (ui32, ui32);
```

The following built-in functions map directly to a particular MIPS DSP REV 2 instruction. Please refer to the architecture specification for details on what each instruction does.

```
v4q7 __builtin_mips_absq_s_qb (v4q7);
v2i16 __builtin_mips_addu_ph (v2i16, v2i16);
v2i16 __builtin_mips_addu_s_ph (v2i16, v2i16);
v4i8 __builtin_mips_adduh_qb (v4i8, v4i8);
v4i8 __builtin_mips_adduh_r_qb (v4i8, v4i8);
i32 __builtin_mips_append (i32, i32, imm0_31);
i32 __builtin_mips_balign (i32, i32, imm0_3);
i32 __builtin_mips_cmpgdu_eq_qb (v4i8, v4i8);
i32 __builtin_mips_cmpgdu_lt_qb (v4i8, v4i8);
i32 __builtin_mips_cmpgdu_le_qb (v4i8, v4i8);
a64 __builtin_mips_dpa_w_ph (a64, v2i16, v2i16);
a64 __builtin_mips_dps_w_ph (a64, v2i16, v2i16);
v2i16 __builtin_mips_mul_ph (v2i16, v2i16);
```

```
v2i16 __builtin_mips_mul_s_ph (v2i16, v2i16);
q31 __builtin_mips_mulq_rs_w (q31, q31);
v2q15 __builtin_mips_mulq_s_ph (v2q15, v2q15);
q31 __builtin_mips_mulq_s_w (q31, q31);
a64 __builtin_mips_mulsa_w_ph (a64, v2i16, v2i16);
v4i8 __builtin_mips_precr_qb_ph (v2i16, v2i16);
v2i16 __builtin_mips_precr_sra_ph_w (i32, i32, imm0_31);
v2i16 __builtin_mips_precr_sra_r_ph_w (i32, i32, imm0_31);
i32 __builtin_mips_prepend (i32, i32, imm0_31);
v4i8 __builtin_mips_shra_qb (v4i8, imm0_7);
v4i8 __builtin_mips_shra_r_qb (v4i8, imm0_7);
v4i8 __builtin_mips_shra_qb (v4i8, i32);
v4i8 __builtin_mips_shra_r_qb (v4i8, i32);
v2i16 __builtin_mips_shrl_ph (v2i16, imm0_15);
v2i16 __builtin_mips_shrl_ph (v2i16, i32);
v2i16 __builtin_mips_subu_ph (v2i16, v2i16);
v2i16 __builtin_mips_subu_s_ph (v2i16, v2i16);
v4i8 __builtin_mips_subuh_qb (v4i8, v4i8);
v4i8 __builtin_mips_subuh_r_qb (v4i8, v4i8);
v2q15 __builtin_mips_addqh_ph (v2q15, v2q15);
v2q15 __builtin_mips_addqh_r_ph (v2q15, v2q15);
q31 __builtin_mips_addqh_w (q31, q31);
q31 __builtin_mips_addqh_r_w (q31, q31);
v2q15 __builtin_mips_subqh_ph (v2q15, v2q15);
v2q15 __builtin_mips_subqh_r_ph (v2q15, v2q15);
q31 __builtin_mips_subqh_w (q31, q31);
q31 __builtin_mips_subqh_r_w (q31, q31);
a64 __builtin_mips_dpax_w_ph (a64, v2i16, v2i16);
a64 __builtin_mips_dpsx_w_ph (a64, v2i16, v2i16);
a64 __builtin_mips_dpaqx_s_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpaqx_sa_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpsqx_s_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpsqx_sa_w_ph (a64, v2q15, v2q15);
```

6.55.9 MIPS Paired-Single Support

The MIPS64 architecture includes a number of instructions that operate on pairs of single-precision floating-point values. Each pair is packed into a 64-bit floating-point register, with one element being designated the "upper half" and the other being designated the "lower half".

GCC supports paired-single operations using both the generic vector extensions (see Section 6.49 [Vector Extensions], page 423) and a collection of MIPS-specific built-in functions. Both kinds of support are enabled by the '`-mpaired-single`' command-line option.

The vector type associated with paired-single values is usually called `v2sf`. It can be defined in C as follows:

```
typedef float v2sf __attribute__ ((vector_size (8)));
```

`v2sf` values are initialized in the same way as aggregates. For example:

```
v2sf a = {1.5, 9.1};
v2sf b;
float e, f;
b = (v2sf) {e, f};
```

Note: The CPU's endianness determines which value is stored in the upper half of a register and which value is stored in the lower half. On little-endian targets, the first value is the lower one and the second value is the upper one. The opposite order applies to

big-endian targets. For example, the code above will set the lower half of `a` to `1.5` on little-endian targets and `9.1` on big-endian targets.

6.55.10 MIPS Loongson Built-in Functions

GCC provides intrinsics to access the SIMD instructions provided by the ST Microelectronics Loongson-2E and -2F processors. These intrinsics, available after inclusion of the `loongson.h` header file, operate on the following 64-bit vector types:

- `uint8x8_t`, a vector of eight unsigned 8-bit integers;
- `uint16x4_t`, a vector of four unsigned 16-bit integers;
- `uint32x2_t`, a vector of two unsigned 32-bit integers;
- `int8x8_t`, a vector of eight signed 8-bit integers;
- `int16x4_t`, a vector of four signed 16-bit integers;
- `int32x2_t`, a vector of two signed 32-bit integers.

The intrinsics provided are listed below; each is named after the machine instruction to which it corresponds, with suffixes added as appropriate to distinguish intrinsics that expand to the same machine instruction yet have different argument types. Refer to the architecture documentation for a description of the functionality of each instruction.

```
int16x4_t packsswh (int32x2_t s, int32x2_t t);
int8x8_t packsshb (int16x4_t s, int16x4_t t);
uint8x8_t packushb (uint16x4_t s, uint16x4_t t);
uint32x2_t paddw_u (uint32x2_t s, uint32x2_t t);
uint16x4_t paddh_u (uint16x4_t s, uint16x4_t t);
uint8x8_t paddb_u (uint8x8_t s, uint8x8_t t);
int32x2_t paddw_s (int32x2_t s, int32x2_t t);
int16x4_t paddh_s (int16x4_t s, int16x4_t t);
int8x8_t paddb_s (int8x8_t s, int8x8_t t);
uint64_t paddd_u (uint64_t s, uint64_t t);
int64_t paddd_s (int64_t s, int64_t t);
int16x4_t paddsh (int16x4_t s, int16x4_t t);
int8x8_t paddsb (int8x8_t s, int8x8_t t);
uint16x4_t paddush (uint16x4_t s, uint16x4_t t);
uint8x8_t paddusb (uint8x8_t s, uint8x8_t t);
uint64_t pandn_ud (uint64_t s, uint64_t t);
uint32x2_t pandn_uw (uint32x2_t s, uint32x2_t t);
uint16x4_t pandn_uh (uint16x4_t s, uint16x4_t t);
uint8x8_t pandn_ub (uint8x8_t s, uint8x8_t t);
int64_t pandn_sd (int64_t s, int64_t t);
int32x2_t pandn_sw (int32x2_t s, int32x2_t t);
int16x4_t pandn_sh (int16x4_t s, int16x4_t t);
int8x8_t pandn_sb (int8x8_t s, int8x8_t t);
uint16x4_t pavgh (uint16x4_t s, uint16x4_t t);
uint8x8_t pavgb (uint8x8_t s, uint8x8_t t);
uint32x2_t pcmpeqw_u (uint32x2_t s, uint32x2_t t);
uint16x4_t pcmpeqh_u (uint16x4_t s, uint16x4_t t);
uint8x8_t pcmpeqb_u (uint8x8_t s, uint8x8_t t);
int32x2_t pcmpeqw_s (int32x2_t s, int32x2_t t);
int16x4_t pcmpeqh_s (int16x4_t s, int16x4_t t);
int8x8_t pcmpeqb_s (int8x8_t s, int8x8_t t);
uint32x2_t pcmpgtw_u (uint32x2_t s, uint32x2_t t);
uint16x4_t pcmpgth_u (uint16x4_t s, uint16x4_t t);
uint8x8_t pcmpgtb_u (uint8x8_t s, uint8x8_t t);
int32x2_t pcmpgtw_s (int32x2_t s, int32x2_t t);
```

```
int16x4_t pcmpgth_s (int16x4_t s, int16x4_t t);
int8x8_t pcmpgtb_s (int8x8_t s, int8x8_t t);
uint16x4_t pextrh_u (uint16x4_t s, int field);
int16x4_t pextrh_s (int16x4_t s, int field);
uint16x4_t pinsrh_0_u (uint16x4_t s, uint16x4_t t);
uint16x4_t pinsrh_1_u (uint16x4_t s, uint16x4_t t);
uint16x4_t pinsrh_2_u (uint16x4_t s, uint16x4_t t);
uint16x4_t pinsrh_3_u (uint16x4_t s, uint16x4_t t);
int16x4_t pinsrh_0_s (int16x4_t s, int16x4_t t);
int16x4_t pinsrh_1_s (int16x4_t s, int16x4_t t);
int16x4_t pinsrh_2_s (int16x4_t s, int16x4_t t);
int16x4_t pinsrh_3_s (int16x4_t s, int16x4_t t);
int32x2_t pmaddhw (int16x4_t s, int16x4_t t);
int16x4_t pmaxsh (int16x4_t s, int16x4_t t);
uint8x8_t pmaxub (uint8x8_t s, uint8x8_t t);
int16x4_t pminsh (int16x4_t s, int16x4_t t);
uint8x8_t pminub (uint8x8_t s, uint8x8_t t);
uint8x8_t pmovmskb_u (uint8x8_t s);
int8x8_t pmovmskb_s (int8x8_t s);
uint16x4_t pmulhuh (uint16x4_t s, uint16x4_t t);
int16x4_t pmulhh (int16x4_t s, int16x4_t t);
int16x4_t pmullh (int16x4_t s, int16x4_t t);
int64_t pmuluw (uint32x2_t s, uint32x2_t t);
uint8x8_t pasubub (uint8x8_t s, uint8x8_t t);
uint16x4_t biadd (uint8x8_t s);
uint16x4_t psadbh (uint8x8_t s, uint8x8_t t);
uint16x4_t pshufh_u (uint16x4_t dest, uint16x4_t s, uint8_t order);
int16x4_t pshufh_s (int16x4_t dest, int16x4_t s, uint8_t order);
uint16x4_t psllh_u (uint16x4_t s, uint8_t amount);
int16x4_t psllh_s (int16x4_t s, uint8_t amount);
uint32x2_t psllw_u (uint32x2_t s, uint8_t amount);
int32x2_t psllw_s (int32x2_t s, uint8_t amount);
uint16x4_t psrlh_u (uint16x4_t s, uint8_t amount);
int16x4_t psrlh_s (int16x4_t s, uint8_t amount);
uint32x2_t psrlw_u (uint32x2_t s, uint8_t amount);
int32x2_t psrlw_s (int32x2_t s, uint8_t amount);
uint16x4_t psrah_u (uint16x4_t s, uint8_t amount);
int16x4_t psrah_s (int16x4_t s, uint8_t amount);
uint32x2_t psraw_u (uint32x2_t s, uint8_t amount);
int32x2_t psraw_s (int32x2_t s, uint8_t amount);
uint32x2_t psubw_u (uint32x2_t s, uint32x2_t t);
uint16x4_t psubh_u (uint16x4_t s, uint16x4_t t);
uint8x8_t psubb_u (uint8x8_t s, uint8x8_t t);
int32x2_t psubw_s (int32x2_t s, int32x2_t t);
int16x4_t psubh_s (int16x4_t s, int16x4_t t);
int8x8_t psubb_s (int8x8_t s, int8x8_t t);
uint64_t psubd_u (uint64_t s, uint64_t t);
int64_t psubd_s (int64_t s, int64_t t);
int16x4_t psubsh (int16x4_t s, int16x4_t t);
int8x8_t psubsb (int8x8_t s, int8x8_t t);
uint16x4_t psubush (uint16x4_t s, uint16x4_t t);
uint8x8_t psubusb (uint8x8_t s, uint8x8_t t);
uint32x2_t punpckhwd_u (uint32x2_t s, uint32x2_t t);
uint16x4_t punpckhhw_u (uint16x4_t s, uint16x4_t t);
uint8x8_t punpckhbh_u (uint8x8_t s, uint8x8_t t);
int32x2_t punpckhwd_s (int32x2_t s, int32x2_t t);
int16x4_t punpckhhw_s (int16x4_t s, int16x4_t t);
int8x8_t punpckhbh_s (int8x8_t s, int8x8_t t);
```

```
uint32x2_t punpcklwd_u (uint32x2_t s, uint32x2_t t);
uint16x4_t punpcklhw_u (uint16x4_t s, uint16x4_t t);
uint8x8_t punpcklbh_u (uint8x8_t s, uint8x8_t t);
int32x2_t punpcklwd_s (int32x2_t s, int32x2_t t);
int16x4_t punpcklhw_s (int16x4_t s, int16x4_t t);
int8x8_t punpcklbh_s (int8x8_t s, int8x8_t t);
```

6.55.10.1 Paired-Single Arithmetic

The table below lists the `v2sf` operations for which hardware support exists. `a`, `b` and `c` are `v2sf` values and `x` is an integral value.

C code	MIPS instruction
`a + b`	`add.ps`
`a - b`	`sub.ps`
`-a`	`neg.ps`
`a * b`	`mul.ps`
`a * b + c`	`madd.ps`
`a * b - c`	`msub.ps`
`-(a * b + c)`	`nmadd.ps`
`-(a * b - c)`	`nmsub.ps`
`x ? a : b`	`movn.ps/movz.ps`

Note that the multiply-accumulate instructions can be disabled using the command-line option `-mno-fused-madd`.

6.55.10.2 Paired-Single Built-in Functions

The following paired-single functions map directly to a particular MIPS instruction. Please refer to the architecture specification for details on what each instruction does.

`v2sf __builtin_mips_pll_ps (v2sf, v2sf)`
: Pair lower lower (`pll.ps`).

`v2sf __builtin_mips_pul_ps (v2sf, v2sf)`
: Pair upper lower (`pul.ps`).

`v2sf __builtin_mips_plu_ps (v2sf, v2sf)`
: Pair lower upper (`plu.ps`).

`v2sf __builtin_mips_puu_ps (v2sf, v2sf)`
: Pair upper upper (`puu.ps`).

`v2sf __builtin_mips_cvt_ps_s (float, float)`
: Convert pair to paired single (`cvt.ps.s`).

`float __builtin_mips_cvt_s_pl (v2sf)`
: Convert pair lower to single (`cvt.s.pl`).

`float __builtin_mips_cvt_s_pu (v2sf)`
: Convert pair upper to single (`cvt.s.pu`).

`v2sf __builtin_mips_abs_ps (v2sf)`
: Absolute value (`abs.ps`).

`v2sf __builtin_mips_alnv_ps (v2sf, v2sf, int)`
: Align variable (`alnv.ps`).

Note: The value of the third parameter must be 0 or 4 modulo 8, otherwise the result will be unpredictable. Please read the instruction description for details.

The following multi-instruction functions are also available. In each case, *cond* can be any of the 16 floating-point conditions: `f`, `un`, `eq`, `ueq`, `olt`, `ult`, `ole`, `ule`, `sf`, `ngle`, `seq`, `ngl`, `lt`, `nge`, `le` or `ngt`.

`v2sf __builtin_mips_movt_c_`*cond*`_ps (v2sf a, v2sf b, v2sf c, v2sf d)`
`v2sf __builtin_mips_movf_c_`*cond*`_ps (v2sf a, v2sf b, v2sf c, v2sf d)`

Conditional move based on floating point comparison (`c.`*cond*`.ps`, `movt.ps`/`movf.ps`).

The `movt` functions return the value *x* computed by:

```
c.cond.ps cc,a,b
mov.ps x,c
movt.ps x,d,cc
```

The `movf` functions are similar but use `movf.ps` instead of `movt.ps`.

`int __builtin_mips_upper_c_`*cond*`_ps (v2sf a, v2sf b)`
`int __builtin_mips_lower_c_`*cond*`_ps (v2sf a, v2sf b)`

Comparison of two paired-single values (`c.`*cond*`.ps`, `bc1t`/`bc1f`).

These functions compare *a* and *b* using `c.`*cond*`.ps` and return either the upper or lower half of the result. For example:

```
v2sf a, b;
if (__builtin_mips_upper_c_eq_ps (a, b))
  upper_halves_are_equal ();
else
  upper_halves_are_unequal ();

if (__builtin_mips_lower_c_eq_ps (a, b))
  lower_halves_are_equal ();
else
  lower_halves_are_unequal ();
```

6.55.10.3 MIPS-3D Built-in Functions

The MIPS-3D Application-Specific Extension (ASE) includes additional paired-single instructions that are designed to improve the performance of 3D graphics operations. Support for these instructions is controlled by the '`-mips3d`' command-line option.

The functions listed below map directly to a particular MIPS-3D instruction. Please refer to the architecture specification for more details on what each instruction does.

`v2sf __builtin_mips_addr_ps (v2sf, v2sf)`

Reduction add (`addr.ps`).

`v2sf __builtin_mips_mulr_ps (v2sf, v2sf)`

Reduction multiply (`mulr.ps`).

`v2sf __builtin_mips_cvt_pw_ps (v2sf)`

Convert paired single to paired word (`cvt.pw.ps`).

`v2sf __builtin_mips_cvt_ps_pw (v2sf)`

Convert paired word to paired single (`cvt.ps.pw`).

`float __builtin_mips_recip1_s (float)`
`double __builtin_mips_recip1_d (double)`
`v2sf __builtin_mips_recip1_ps (v2sf)`
Reduced precision reciprocal (sequence step 1) (`recip1.`*`fmt`*).

`float __builtin_mips_recip2_s (float, float)`
`double __builtin_mips_recip2_d (double, double)`
`v2sf __builtin_mips_recip2_ps (v2sf, v2sf)`
Reduced precision reciprocal (sequence step 2) (`recip2.`*`fmt`*).

`float __builtin_mips_rsqrt1_s (float)`
`double __builtin_mips_rsqrt1_d (double)`
`v2sf __builtin_mips_rsqrt1_ps (v2sf)`
Reduced precision reciprocal square root (sequence step 1) (`rsqrt1.`*`fmt`*).

`float __builtin_mips_rsqrt2_s (float, float)`
`double __builtin_mips_rsqrt2_d (double, double)`
`v2sf __builtin_mips_rsqrt2_ps (v2sf, v2sf)`
Reduced precision reciprocal square root (sequence step 2) (`rsqrt2.`*`fmt`*).

The following multi-instruction functions are also available. In each case, *cond* can be any of the 16 floating-point conditions: `f`, `un`, `eq`, `ueq`, `olt`, `ult`, `ole`, `ule`, `sf`, `ngle`, `seq`, `ngl`, `lt`, `nge`, `le` or `ngt`.

`int __builtin_mips_cabs_`*`cond`*`_s (float a, float b)`
`int __builtin_mips_cabs_`*`cond`*`_d (double a, double b)`
Absolute comparison of two scalar values (`cabs.`*`cond`*`.`*`fmt`*, `bc1t`/`bc1f`).

These functions compare *a* and *b* using `cabs.`*`cond`*`.s` or `cabs.`*`cond`*`.d` and return the result as a boolean value. For example:

```
float a, b;
if (__builtin_mips_cabs_eq_s (a, b))
  true ();
else
  false ();
```

`int __builtin_mips_upper_cabs_`*`cond`*`_ps (v2sf a, v2sf b)`
`int __builtin_mips_lower_cabs_`*`cond`*`_ps (v2sf a, v2sf b)`
Absolute comparison of two paired-single values (`cabs.`*`cond`*`.ps`, `bc1t`/`bc1f`).

These functions compare *a* and *b* using `cabs.`*`cond`*`.ps` and return either the upper or lower half of the result. For example:

```
v2sf a, b;
if (__builtin_mips_upper_cabs_eq_ps (a, b))
  upper_halves_are_equal ();
else
  upper_halves_are_unequal ();

if (__builtin_mips_lower_cabs_eq_ps (a, b))
  lower_halves_are_equal ();
else
  lower_halves_are_unequal ();
```

`v2sf __builtin_mips_movt_cabs_`*`cond`*`_ps (v2sf a, v2sf b, v2sf c, v2sf d)`
`v2sf __builtin_mips_movf_cabs_`*`cond`*`_ps (v2sf a, v2sf b, v2sf c, v2sf d)`

Conditional move based on absolute comparison (`cabs.`*`cond`*`.ps`, `movt.ps`/`movf.ps`).

The `movt` functions return the value *x* computed by:

```
cabs.cond.ps cc,a,b
mov.ps x,c
movt.ps x,d,cc
```

The `movf` functions are similar but use `movf.ps` instead of `movt.ps`.

`int __builtin_mips_any_c_`*`cond`*`_ps (v2sf a, v2sf b)`
`int __builtin_mips_all_c_`*`cond`*`_ps (v2sf a, v2sf b)`
`int __builtin_mips_any_cabs_`*`cond`*`_ps (v2sf a, v2sf b)`
`int __builtin_mips_all_cabs_`*`cond`*`_ps (v2sf a, v2sf b)`

Comparison of two paired-single values (`c.`*`cond`*`.ps`/`cabs.`*`cond`*`.ps`, `bc1any2t`/`bc1any2f`).

These functions compare *a* and *b* using `c.`*`cond`*`.ps` or `cabs.`*`cond`*`.ps`. The `any` forms return true if either result is true and the `all` forms return true if both results are true. For example:

```
v2sf a, b;
if (__builtin_mips_any_c_eq_ps (a, b))
  one_is_true ();
else
  both_are_false ();

if (__builtin_mips_all_c_eq_ps (a, b))
  both_are_true ();
else
  one_is_false ();
```

`int __builtin_mips_any_c_`*`cond`*`_4s (v2sf a, v2sf b, v2sf c, v2sf d)`
`int __builtin_mips_all_c_`*`cond`*`_4s (v2sf a, v2sf b, v2sf c, v2sf d)`
`int __builtin_mips_any_cabs_`*`cond`*`_4s (v2sf a, v2sf b, v2sf c, v2sf d)`
`int __builtin_mips_all_cabs_`*`cond`*`_4s (v2sf a, v2sf b, v2sf c, v2sf d)`

Comparison of four paired-single values (`c.`*`cond`*`.ps`/`cabs.`*`cond`*`.ps`, `bc1any4t`/`bc1any4f`).

These functions use `c.`*`cond`*`.ps` or `cabs.`*`cond`*`.ps` to compare *a* with *b* and to compare *c* with *d*. The `any` forms return true if any of the four results are true and the `all` forms return true if all four results are true. For example:

```
v2sf a, b, c, d;
if (__builtin_mips_any_c_eq_4s (a, b, c, d))
  some_are_true ();
else
  all_are_false ();

if (__builtin_mips_all_c_eq_4s (a, b, c, d))
  all_are_true ();
else
  some_are_false ();
```

6.55.11 picoChip Built-in Functions

GCC provides an interface to selected machine instructions from the picoChip instruction set.

`int __builtin_sbc (int value)`
: Sign bit count. Return the number of consecutive bits in *value* which have the same value as the sign-bit. The result is the number of leading sign bits minus one, giving the number of redundant sign bits in *value*.

`int __builtin_byteswap (int value)`
: Byte swap. Return the result of swapping the upper and lower bytes of *value*.

`int __builtin_brev (int value)`
: Bit reversal. Return the result of reversing the bits in *value*. Bit 15 is swapped with bit 0, bit 14 is swapped with bit 1, and so on.

`int __builtin_adds (int x, int y)`
: Saturating addition. Return the result of adding x and y, storing the value 32767 if the result overflows.

`int __builtin_subs (int x, int y)`
: Saturating subtraction. Return the result of subtracting y from x, storing the value -32768 if the result overflows.

`void __builtin_halt (void)`
: Halt. The processor will stop execution. This built-in is useful for implementing assertions.

6.55.12 Other MIPS Built-in Functions

GCC provides other MIPS-specific built-in functions:

`void __builtin_mips_cache (int op, const volatile void *addr)`
: Insert a '`cache`' instruction with operands *op* and *addr*. GCC defines the preprocessor macro `___GCC_HAVE_BUILTIN_MIPS_CACHE` when this function is available.

6.55.13 PowerPC AltiVec Built-in Functions

GCC provides an interface for the PowerPC family of processors to access the AltiVec operations described in Motorola's AltiVec Programming Interface Manual. The interface is made available by including `<altivec.h>` and using '`-maltivec`' and '`-mabi=altivec`'. The interface supports the following vector types.

```
vector unsigned char
vector signed char
vector bool char

vector unsigned short
vector signed short
vector bool short
vector pixel

vector unsigned int
vector signed int
```

```
vector bool int
vector float
```

If '`-mvsx`' is used the following additional vector types are implemented.

```
vector unsigned long
vector signed long
vector double
```

The long types are only implemented for 64-bit code generation, and the long type is only used in the floating point/integer conversion instructions.

GCC's implementation of the high-level language interface available from C and C++ code differs from Motorola's documentation in several ways.

- A vector constant is a list of constant expressions within curly braces.
- A vector initializer requires no cast if the vector constant is of the same type as the variable it is initializing.
- If `signed` or `unsigned` is omitted, the signedness of the vector type is the default signedness of the base type. The default varies depending on the operating system, so a portable program should always specify the signedness.
- Compiling with '`-maltivec`' adds keywords `__vector`, `vector`, `__pixel`, `pixel`, `__bool` and `bool`. When compiling ISO C, the context-sensitive substitution of the keywords `vector`, `pixel` and `bool` is disabled. To use them, you must include `<altivec.h>` instead.
- GCC allows using a `typedef` name as the type specifier for a vector type.
- For C, overloaded functions are implemented with macros so the following does not work:

  ```
  vec_add ((vector signed int){1, 2, 3, 4}, foo);
  ```

  Since `vec_add` is a macro, the vector constant in the example is treated as four separate arguments. Wrap the entire argument in parentheses for this to work.

Note: Only the `<altivec.h>` interface is supported. Internally, GCC uses built-in functions to achieve the functionality in the aforementioned header file, but they are not supported and are subject to change without notice.

The following interfaces are supported for the generic and specific AltiVec operations and the AltiVec predicates. In cases where there is a direct mapping between generic and specific operations, only the generic names are shown here, although the specific operations can also be used.

Arguments that are documented as `const int` require literal integral values within the range required for that operation.

```
vector signed char vec_abs (vector signed char);
vector signed short vec_abs (vector signed short);
vector signed int vec_abs (vector signed int);
vector float vec_abs (vector float);

vector signed char vec_abss (vector signed char);
vector signed short vec_abss (vector signed short);
vector signed int vec_abss (vector signed int);

vector signed char vec_add (vector bool char, vector signed char);
vector signed char vec_add (vector signed char, vector bool char);
```

```
vector signed char vec_add (vector signed char, vector signed char);
vector unsigned char vec_add (vector bool char, vector unsigned char);
vector unsigned char vec_add (vector unsigned char, vector bool char);
vector unsigned char vec_add (vector unsigned char,
                              vector unsigned char);
vector signed short vec_add (vector bool short, vector signed short);
vector signed short vec_add (vector signed short, vector bool short);
vector signed short vec_add (vector signed short, vector signed short);
vector unsigned short vec_add (vector bool short,
                               vector unsigned short);
vector unsigned short vec_add (vector unsigned short,
                               vector bool short);
vector unsigned short vec_add (vector unsigned short,
                               vector unsigned short);
vector signed int vec_add (vector bool int, vector signed int);
vector signed int vec_add (vector signed int, vector bool int);
vector signed int vec_add (vector signed int, vector signed int);
vector unsigned int vec_add (vector bool int, vector unsigned int);
vector unsigned int vec_add (vector unsigned int, vector bool int);
vector unsigned int vec_add (vector unsigned int, vector unsigned int);
vector float vec_add (vector float, vector float);

vector float vec_vaddfp (vector float, vector float);

vector signed int vec_vadduwm (vector bool int, vector signed int);
vector signed int vec_vadduwm (vector signed int, vector bool int);
vector signed int vec_vadduwm (vector signed int, vector signed int);
vector unsigned int vec_vadduwm (vector bool int, vector unsigned int);
vector unsigned int vec_vadduwm (vector unsigned int, vector bool int);
vector unsigned int vec_vadduwm (vector unsigned int,
                                 vector unsigned int);

vector signed short vec_vadduhm (vector bool short,
                                 vector signed short);
vector signed short vec_vadduhm (vector signed short,
                                 vector bool short);
vector signed short vec_vadduhm (vector signed short,
                                 vector signed short);
vector unsigned short vec_vadduhm (vector bool short,
                                   vector unsigned short);
vector unsigned short vec_vadduhm (vector unsigned short,
                                   vector bool short);
vector unsigned short vec_vadduhm (vector unsigned short,
                                   vector unsigned short);

vector signed char vec_vaddubm (vector bool char, vector signed char);
vector signed char vec_vaddubm (vector signed char, vector bool char);
vector signed char vec_vaddubm (vector signed char, vector signed char);
vector unsigned char vec_vaddubm (vector bool char,
                                  vector unsigned char);
vector unsigned char vec_vaddubm (vector unsigned char,
                                  vector bool char);
vector unsigned char vec_vaddubm (vector unsigned char,
                                  vector unsigned char);

vector unsigned int vec_addc (vector unsigned int, vector unsigned int);

vector unsigned char vec_adds (vector bool char, vector unsigned char);
```

```
vector unsigned char vec_adds (vector unsigned char, vector bool char);
vector unsigned char vec_adds (vector unsigned char,
                               vector unsigned char);
vector signed char vec_adds (vector bool char, vector signed char);
vector signed char vec_adds (vector signed char, vector bool char);
vector signed char vec_adds (vector signed char, vector signed char);
vector unsigned short vec_adds (vector bool short,
                                vector unsigned short);
vector unsigned short vec_adds (vector unsigned short,
                                vector bool short);
vector unsigned short vec_adds (vector unsigned short,
                                vector unsigned short);
vector signed short vec_adds (vector bool short, vector signed short);
vector signed short vec_adds (vector signed short, vector bool short);
vector signed short vec_adds (vector signed short, vector signed short);
vector unsigned int vec_adds (vector bool int, vector unsigned int);
vector unsigned int vec_adds (vector unsigned int, vector bool int);
vector unsigned int vec_adds (vector unsigned int, vector unsigned int);
vector signed int vec_adds (vector bool int, vector signed int);
vector signed int vec_adds (vector signed int, vector bool int);
vector signed int vec_adds (vector signed int, vector signed int);

vector signed int vec_vaddsws (vector bool int, vector signed int);
vector signed int vec_vaddsws (vector signed int, vector bool int);
vector signed int vec_vaddsws (vector signed int, vector signed int);

vector unsigned int vec_vadduws (vector bool int, vector unsigned int);
vector unsigned int vec_vadduws (vector unsigned int, vector bool int);
vector unsigned int vec_vadduws (vector unsigned int,
                                 vector unsigned int);

vector signed short vec_vaddshs (vector bool short,
                                 vector signed short);
vector signed short vec_vaddshs (vector signed short,
                                 vector bool short);
vector signed short vec_vaddshs (vector signed short,
                                 vector signed short);

vector unsigned short vec_vadduhs (vector bool short,
                                   vector unsigned short);
vector unsigned short vec_vadduhs (vector unsigned short,
                                   vector bool short);
vector unsigned short vec_vadduhs (vector unsigned short,
                                   vector unsigned short);

vector signed char vec_vaddsbs (vector bool char, vector signed char);
vector signed char vec_vaddsbs (vector signed char, vector bool char);
vector signed char vec_vaddsbs (vector signed char, vector signed char);

vector unsigned char vec_vaddubs (vector bool char,
                                  vector unsigned char);
vector unsigned char vec_vaddubs (vector unsigned char,
                                  vector bool char);
vector unsigned char vec_vaddubs (vector unsigned char,
                                  vector unsigned char);

vector float vec_and (vector float, vector float);
vector float vec_and (vector float, vector bool int);
```

```
vector float vec_and (vector bool int, vector float);
vector bool int vec_and (vector bool int, vector bool int);
vector signed int vec_and (vector bool int, vector signed int);
vector signed int vec_and (vector signed int, vector bool int);
vector signed int vec_and (vector signed int, vector signed int);
vector unsigned int vec_and (vector bool int, vector unsigned int);
vector unsigned int vec_and (vector unsigned int, vector bool int);
vector unsigned int vec_and (vector unsigned int, vector unsigned int);
vector bool short vec_and (vector bool short, vector bool short);
vector signed short vec_and (vector bool short, vector signed short);
vector signed short vec_and (vector signed short, vector bool short);
vector signed short vec_and (vector signed short, vector signed short);
vector unsigned short vec_and (vector bool short,
                               vector unsigned short);
vector unsigned short vec_and (vector unsigned short,
                               vector bool short);
vector unsigned short vec_and (vector unsigned short,
                               vector unsigned short);
vector signed char vec_and (vector bool char, vector signed char);
vector bool char vec_and (vector bool char, vector bool char);
vector signed char vec_and (vector signed char, vector bool char);
vector signed char vec_and (vector signed char, vector signed char);
vector unsigned char vec_and (vector bool char, vector unsigned char);
vector unsigned char vec_and (vector unsigned char, vector bool char);
vector unsigned char vec_and (vector unsigned char,
                              vector unsigned char);

vector float vec_andc (vector float, vector float);
vector float vec_andc (vector float, vector bool int);
vector float vec_andc (vector bool int, vector float);
vector bool int vec_andc (vector bool int, vector bool int);
vector signed int vec_andc (vector bool int, vector signed int);
vector signed int vec_andc (vector signed int, vector bool int);
vector signed int vec_andc (vector signed int, vector signed int);
vector unsigned int vec_andc (vector bool int, vector unsigned int);
vector unsigned int vec_andc (vector unsigned int, vector bool int);
vector unsigned int vec_andc (vector unsigned int, vector unsigned int);
vector bool short vec_andc (vector bool short, vector bool short);
vector signed short vec_andc (vector bool short, vector signed short);
vector signed short vec_andc (vector signed short, vector bool short);
vector signed short vec_andc (vector signed short, vector signed short);
vector unsigned short vec_andc (vector bool short,
                                vector unsigned short);
vector unsigned short vec_andc (vector unsigned short,
                                vector bool short);
vector unsigned short vec_andc (vector unsigned short,
                                vector unsigned short);
vector signed char vec_andc (vector bool char, vector signed char);
vector bool char vec_andc (vector bool char, vector bool char);
vector signed char vec_andc (vector signed char, vector bool char);
vector signed char vec_andc (vector signed char, vector signed char);
vector unsigned char vec_andc (vector bool char, vector unsigned char);
vector unsigned char vec_andc (vector unsigned char, vector bool char);
vector unsigned char vec_andc (vector unsigned char,
                               vector unsigned char);

vector unsigned char vec_avg (vector unsigned char,
                              vector unsigned char);
```

```
vector signed char vec_avg (vector signed char, vector signed char);
vector unsigned short vec_avg (vector unsigned short,
                               vector unsigned short);
vector signed short vec_avg (vector signed short, vector signed short);
vector unsigned int vec_avg (vector unsigned int, vector unsigned int);
vector signed int vec_avg (vector signed int, vector signed int);

vector signed int vec_vavgsw (vector signed int, vector signed int);

vector unsigned int vec_vavguw (vector unsigned int,
                                vector unsigned int);

vector signed short vec_vavgsh (vector signed short,
                                vector signed short);

vector unsigned short vec_vavguh (vector unsigned short,
                                  vector unsigned short);

vector signed char vec_vavgsb (vector signed char, vector signed char);

vector unsigned char vec_vavgub (vector unsigned char,
                                 vector unsigned char);

vector float vec_copysign (vector float);

vector float vec_ceil (vector float);

vector signed int vec_cmpb (vector float, vector float);

vector bool char vec_cmpeq (vector signed char, vector signed char);
vector bool char vec_cmpeq (vector unsigned char, vector unsigned char);
vector bool short vec_cmpeq (vector signed short, vector signed short);
vector bool short vec_cmpeq (vector unsigned short,
                             vector unsigned short);
vector bool int vec_cmpeq (vector signed int, vector signed int);
vector bool int vec_cmpeq (vector unsigned int, vector unsigned int);
vector bool int vec_cmpeq (vector float, vector float);

vector bool int vec_vcmpeqfp (vector float, vector float);

vector bool int vec_vcmpequw (vector signed int, vector signed int);
vector bool int vec_vcmpequw (vector unsigned int, vector unsigned int);

vector bool short vec_vcmpequh (vector signed short,
                                vector signed short);
vector bool short vec_vcmpequh (vector unsigned short,
                                vector unsigned short);

vector bool char vec_vcmpequb (vector signed char, vector signed char);
vector bool char vec_vcmpequb (vector unsigned char,
                               vector unsigned char);

vector bool int vec_cmpge (vector float, vector float);

vector bool char vec_cmpgt (vector unsigned char, vector unsigned char);
vector bool char vec_cmpgt (vector signed char, vector signed char);
vector bool short vec_cmpgt (vector unsigned short,
                             vector unsigned short);
```

```
vector bool short vec_cmpgt (vector signed short, vector signed short);
vector bool int vec_cmpgt (vector unsigned int, vector unsigned int);
vector bool int vec_cmpgt (vector signed int, vector signed int);
vector bool int vec_cmpgt (vector float, vector float);

vector bool int vec_vcmpgtfp (vector float, vector float);

vector bool int vec_vcmpgtsw (vector signed int, vector signed int);

vector bool int vec_vcmpgtuw (vector unsigned int, vector unsigned int);

vector bool short vec_vcmpgtsh (vector signed short,
                                vector signed short);

vector bool short vec_vcmpgtuh (vector unsigned short,
                                vector unsigned short);

vector bool char vec_vcmpgtsb (vector signed char, vector signed char);

vector bool char vec_vcmpgtub (vector unsigned char,
                               vector unsigned char);

vector bool int vec_cmple (vector float, vector float);

vector bool char vec_cmplt (vector unsigned char, vector unsigned char);
vector bool char vec_cmplt (vector signed char, vector signed char);
vector bool short vec_cmplt (vector unsigned short,
                             vector unsigned short);
vector bool short vec_cmplt (vector signed short, vector signed short);
vector bool int vec_cmplt (vector unsigned int, vector unsigned int);
vector bool int vec_cmplt (vector signed int, vector signed int);
vector bool int vec_cmplt (vector float, vector float);

vector float vec_ctf (vector unsigned int, const int);
vector float vec_ctf (vector signed int, const int);

vector float vec_vcfsx (vector signed int, const int);

vector float vec_vcfux (vector unsigned int, const int);

vector signed int vec_cts (vector float, const int);

vector unsigned int vec_ctu (vector float, const int);

void vec_dss (const int);

void vec_dssall (void);

void vec_dst (const vector unsigned char *, int, const int);
void vec_dst (const vector signed char *, int, const int);
void vec_dst (const vector bool char *, int, const int);
void vec_dst (const vector unsigned short *, int, const int);
void vec_dst (const vector signed short *, int, const int);
void vec_dst (const vector bool short *, int, const int);
void vec_dst (const vector pixel *, int, const int);
void vec_dst (const vector unsigned int *, int, const int);
void vec_dst (const vector signed int *, int, const int);
void vec_dst (const vector bool int *, int, const int);
```

```
void vec_dst (const vector float *, int, const int);
void vec_dst (const unsigned char *, int, const int);
void vec_dst (const signed char *, int, const int);
void vec_dst (const unsigned short *, int, const int);
void vec_dst (const short *, int, const int);
void vec_dst (const unsigned int *, int, const int);
void vec_dst (const int *, int, const int);
void vec_dst (const unsigned long *, int, const int);
void vec_dst (const long *, int, const int);
void vec_dst (const float *, int, const int);

void vec_dstst (const vector unsigned char *, int, const int);
void vec_dstst (const vector signed char *, int, const int);
void vec_dstst (const vector bool char *, int, const int);
void vec_dstst (const vector unsigned short *, int, const int);
void vec_dstst (const vector signed short *, int, const int);
void vec_dstst (const vector bool short *, int, const int);
void vec_dstst (const vector pixel *, int, const int);
void vec_dstst (const vector unsigned int *, int, const int);
void vec_dstst (const vector signed int *, int, const int);
void vec_dstst (const vector bool int *, int, const int);
void vec_dstst (const vector float *, int, const int);
void vec_dstst (const unsigned char *, int, const int);
void vec_dstst (const signed char *, int, const int);
void vec_dstst (const unsigned short *, int, const int);
void vec_dstst (const short *, int, const int);
void vec_dstst (const unsigned int *, int, const int);
void vec_dstst (const int *, int, const int);
void vec_dstst (const unsigned long *, int, const int);
void vec_dstst (const long *, int, const int);
void vec_dstst (const float *, int, const int);

void vec_dststt (const vector unsigned char *, int, const int);
void vec_dststt (const vector signed char *, int, const int);
void vec_dststt (const vector bool char *, int, const int);
void vec_dststt (const vector unsigned short *, int, const int);
void vec_dststt (const vector signed short *, int, const int);
void vec_dststt (const vector bool short *, int, const int);
void vec_dststt (const vector pixel *, int, const int);
void vec_dststt (const vector unsigned int *, int, const int);
void vec_dststt (const vector signed int *, int, const int);
void vec_dststt (const vector bool int *, int, const int);
void vec_dststt (const vector float *, int, const int);
void vec_dststt (const unsigned char *, int, const int);
void vec_dststt (const signed char *, int, const int);
void vec_dststt (const unsigned short *, int, const int);
void vec_dststt (const short *, int, const int);
void vec_dststt (const unsigned int *, int, const int);
void vec_dststt (const int *, int, const int);
void vec_dststt (const unsigned long *, int, const int);
void vec_dststt (const long *, int, const int);
void vec_dststt (const float *, int, const int);

void vec_dstt (const vector unsigned char *, int, const int);
void vec_dstt (const vector signed char *, int, const int);
void vec_dstt (const vector bool char *, int, const int);
void vec_dstt (const vector unsigned short *, int, const int);
void vec_dstt (const vector signed short *, int, const int);
```

```
void vec_dstt (const vector bool short *, int, const int);
void vec_dstt (const vector pixel *, int, const int);
void vec_dstt (const vector unsigned int *, int, const int);
void vec_dstt (const vector signed int *, int, const int);
void vec_dstt (const vector bool int *, int, const int);
void vec_dstt (const vector float *, int, const int);
void vec_dstt (const unsigned char *, int, const int);
void vec_dstt (const signed char *, int, const int);
void vec_dstt (const unsigned short *, int, const int);
void vec_dstt (const short *, int, const int);
void vec_dstt (const unsigned int *, int, const int);
void vec_dstt (const int *, int, const int);
void vec_dstt (const unsigned long *, int, const int);
void vec_dstt (const long *, int, const int);
void vec_dstt (const float *, int, const int);

vector float vec_expte (vector float);

vector float vec_floor (vector float);

vector float vec_ld (int, const vector float *);
vector float vec_ld (int, const float *);
vector bool int vec_ld (int, const vector bool int *);
vector signed int vec_ld (int, const vector signed int *);
vector signed int vec_ld (int, const int *);
vector signed int vec_ld (int, const long *);
vector unsigned int vec_ld (int, const vector unsigned int *);
vector unsigned int vec_ld (int, const unsigned int *);
vector unsigned int vec_ld (int, const unsigned long *);
vector bool short vec_ld (int, const vector bool short *);
vector pixel vec_ld (int, const vector pixel *);
vector signed short vec_ld (int, const vector signed short *);
vector signed short vec_ld (int, const short *);
vector unsigned short vec_ld (int, const vector unsigned short *);
vector unsigned short vec_ld (int, const unsigned short *);
vector bool char vec_ld (int, const vector bool char *);
vector signed char vec_ld (int, const vector signed char *);
vector signed char vec_ld (int, const signed char *);
vector unsigned char vec_ld (int, const vector unsigned char *);
vector unsigned char vec_ld (int, const unsigned char *);

vector signed char vec_lde (int, const signed char *);
vector unsigned char vec_lde (int, const unsigned char *);
vector signed short vec_lde (int, const short *);
vector unsigned short vec_lde (int, const unsigned short *);
vector float vec_lde (int, const float *);
vector signed int vec_lde (int, const int *);
vector unsigned int vec_lde (int, const unsigned int *);
vector signed int vec_lde (int, const long *);
vector unsigned int vec_lde (int, const unsigned long *);

vector float vec_lvewx (int, float *);
vector signed int vec_lvewx (int, int *);
vector unsigned int vec_lvewx (int, unsigned int *);
vector signed int vec_lvewx (int, long *);
vector unsigned int vec_lvewx (int, unsigned long *);

vector signed short vec_lvehx (int, short *);
```

```
vector unsigned short vec_lvehx (int, unsigned short *);

vector signed char vec_lvebx (int, char *);
vector unsigned char vec_lvebx (int, unsigned char *);

vector float vec_ldl (int, const vector float *);
vector float vec_ldl (int, const float *);
vector bool int vec_ldl (int, const vector bool int *);
vector signed int vec_ldl (int, const vector signed int *);
vector signed int vec_ldl (int, const int *);
vector signed int vec_ldl (int, const long *);
vector unsigned int vec_ldl (int, const vector unsigned int *);
vector unsigned int vec_ldl (int, const unsigned int *);
vector unsigned int vec_ldl (int, const unsigned long *);
vector bool short vec_ldl (int, const vector bool short *);
vector pixel vec_ldl (int, const vector pixel *);
vector signed short vec_ldl (int, const vector signed short *);
vector signed short vec_ldl (int, const short *);
vector unsigned short vec_ldl (int, const vector unsigned short *);
vector unsigned short vec_ldl (int, const unsigned short *);
vector bool char vec_ldl (int, const vector bool char *);
vector signed char vec_ldl (int, const vector signed char *);
vector signed char vec_ldl (int, const signed char *);
vector unsigned char vec_ldl (int, const vector unsigned char *);
vector unsigned char vec_ldl (int, const unsigned char *);

vector float vec_loge (vector float);

vector unsigned char vec_lvsl (int, const volatile unsigned char *);
vector unsigned char vec_lvsl (int, const volatile signed char *);
vector unsigned char vec_lvsl (int, const volatile unsigned short *);
vector unsigned char vec_lvsl (int, const volatile short *);
vector unsigned char vec_lvsl (int, const volatile unsigned int *);
vector unsigned char vec_lvsl (int, const volatile int *);
vector unsigned char vec_lvsl (int, const volatile unsigned long *);
vector unsigned char vec_lvsl (int, const volatile long *);
vector unsigned char vec_lvsl (int, const volatile float *);

vector unsigned char vec_lvsr (int, const volatile unsigned char *);
vector unsigned char vec_lvsr (int, const volatile signed char *);
vector unsigned char vec_lvsr (int, const volatile unsigned short *);
vector unsigned char vec_lvsr (int, const volatile short *);
vector unsigned char vec_lvsr (int, const volatile unsigned int *);
vector unsigned char vec_lvsr (int, const volatile int *);
vector unsigned char vec_lvsr (int, const volatile unsigned long *);
vector unsigned char vec_lvsr (int, const volatile long *);
vector unsigned char vec_lvsr (int, const volatile float *);

vector float vec_madd (vector float, vector float, vector float);

vector signed short vec_madds (vector signed short,
                               vector signed short,
                               vector signed short);

vector unsigned char vec_max (vector bool char, vector unsigned char);
vector unsigned char vec_max (vector unsigned char, vector bool char);
vector unsigned char vec_max (vector unsigned char,
                              vector unsigned char);
```

```
vector signed char vec_max (vector bool char, vector signed char);
vector signed char vec_max (vector signed char, vector bool char);
vector signed char vec_max (vector signed char, vector signed char);
vector unsigned short vec_max (vector bool short,
                               vector unsigned short);
vector unsigned short vec_max (vector unsigned short,
                               vector bool short);
vector unsigned short vec_max (vector unsigned short,
                               vector unsigned short);
vector signed short vec_max (vector bool short, vector signed short);
vector signed short vec_max (vector signed short, vector bool short);
vector signed short vec_max (vector signed short, vector signed short);
vector unsigned int vec_max (vector bool int, vector unsigned int);
vector unsigned int vec_max (vector unsigned int, vector bool int);
vector unsigned int vec_max (vector unsigned int, vector unsigned int);
vector signed int vec_max (vector bool int, vector signed int);
vector signed int vec_max (vector signed int, vector bool int);
vector signed int vec_max (vector signed int, vector signed int);
vector float vec_max (vector float, vector float);

vector float vec_vmaxfp (vector float, vector float);

vector signed int vec_vmaxsw (vector bool int, vector signed int);
vector signed int vec_vmaxsw (vector signed int, vector bool int);
vector signed int vec_vmaxsw (vector signed int, vector signed int);

vector unsigned int vec_vmaxuw (vector bool int, vector unsigned int);
vector unsigned int vec_vmaxuw (vector unsigned int, vector bool int);
vector unsigned int vec_vmaxuw (vector unsigned int,
                                vector unsigned int);

vector signed short vec_vmaxsh (vector bool short, vector signed short);
vector signed short vec_vmaxsh (vector signed short, vector bool short);
vector signed short vec_vmaxsh (vector signed short,
                                vector signed short);

vector unsigned short vec_vmaxuh (vector bool short,
                                  vector unsigned short);
vector unsigned short vec_vmaxuh (vector unsigned short,
                                  vector bool short);
vector unsigned short vec_vmaxuh (vector unsigned short,
                                  vector unsigned short);

vector signed char vec_vmaxsb (vector bool char, vector signed char);
vector signed char vec_vmaxsb (vector signed char, vector bool char);
vector signed char vec_vmaxsb (vector signed char, vector signed char);

vector unsigned char vec_vmaxub (vector bool char,
                                 vector unsigned char);
vector unsigned char vec_vmaxub (vector unsigned char,
                                 vector bool char);
vector unsigned char vec_vmaxub (vector unsigned char,
                                 vector unsigned char);

vector bool char vec_mergeh (vector bool char, vector bool char);
vector signed char vec_mergeh (vector signed char, vector signed char);
vector unsigned char vec_mergeh (vector unsigned char,
                                 vector unsigned char);
```

```
vector bool short vec_mergeh (vector bool short, vector bool short);
vector pixel vec_mergeh (vector pixel, vector pixel);
vector signed short vec_mergeh (vector signed short,
                                vector signed short);
vector unsigned short vec_mergeh (vector unsigned short,
                                  vector unsigned short);
vector float vec_mergeh (vector float, vector float);
vector bool int vec_mergeh (vector bool int, vector bool int);
vector signed int vec_mergeh (vector signed int, vector signed int);
vector unsigned int vec_mergeh (vector unsigned int,
                                vector unsigned int);

vector float vec_vmrghw (vector float, vector float);
vector bool int vec_vmrghw (vector bool int, vector bool int);
vector signed int vec_vmrghw (vector signed int, vector signed int);
vector unsigned int vec_vmrghw (vector unsigned int,
                                vector unsigned int);

vector bool short vec_vmrghh (vector bool short, vector bool short);
vector signed short vec_vmrghh (vector signed short,
                                vector signed short);
vector unsigned short vec_vmrghh (vector unsigned short,
                                  vector unsigned short);
vector pixel vec_vmrghh (vector pixel, vector pixel);

vector bool char vec_vmrghb (vector bool char, vector bool char);
vector signed char vec_vmrghb (vector signed char, vector signed char);
vector unsigned char vec_vmrghb (vector unsigned char,
                                 vector unsigned char);

vector bool char vec_mergel (vector bool char, vector bool char);
vector signed char vec_mergel (vector signed char, vector signed char);
vector unsigned char vec_mergel (vector unsigned char,
                                 vector unsigned char);
vector bool short vec_mergel (vector bool short, vector bool short);
vector pixel vec_mergel (vector pixel, vector pixel);
vector signed short vec_mergel (vector signed short,
                                vector signed short);
vector unsigned short vec_mergel (vector unsigned short,
                                  vector unsigned short);
vector float vec_mergel (vector float, vector float);
vector bool int vec_mergel (vector bool int, vector bool int);
vector signed int vec_mergel (vector signed int, vector signed int);
vector unsigned int vec_mergel (vector unsigned int,
                                vector unsigned int);

vector float vec_vmrglw (vector float, vector float);
vector signed int vec_vmrglw (vector signed int, vector signed int);
vector unsigned int vec_vmrglw (vector unsigned int,
                                vector unsigned int);
vector bool int vec_vmrglw (vector bool int, vector bool int);

vector bool short vec_vmrglh (vector bool short, vector bool short);
vector signed short vec_vmrglh (vector signed short,
                                vector signed short);
vector unsigned short vec_vmrglh (vector unsigned short,
                                  vector unsigned short);
vector pixel vec_vmrglh (vector pixel, vector pixel);
```

```
vector bool char vec_vmrglb (vector bool char, vector bool char);
vector signed char vec_vmrglb (vector signed char, vector signed char);
vector unsigned char vec_vmrglb (vector unsigned char,
                                 vector unsigned char);

vector unsigned short vec_mfvscr (void);

vector unsigned char vec_min (vector bool char, vector unsigned char);
vector unsigned char vec_min (vector unsigned char, vector bool char);
vector unsigned char vec_min (vector unsigned char,
                              vector unsigned char);
vector signed char vec_min (vector bool char, vector signed char);
vector signed char vec_min (vector signed char, vector bool char);
vector signed char vec_min (vector signed char, vector signed char);
vector unsigned short vec_min (vector bool short,
                               vector unsigned short);
vector unsigned short vec_min (vector unsigned short,
                               vector bool short);
vector unsigned short vec_min (vector unsigned short,
                               vector unsigned short);
vector signed short vec_min (vector bool short, vector signed short);
vector signed short vec_min (vector signed short, vector bool short);
vector signed short vec_min (vector signed short, vector signed short);
vector unsigned int vec_min (vector bool int, vector unsigned int);
vector unsigned int vec_min (vector unsigned int, vector bool int);
vector unsigned int vec_min (vector unsigned int, vector unsigned int);
vector signed int vec_min (vector bool int, vector signed int);
vector signed int vec_min (vector signed int, vector bool int);
vector signed int vec_min (vector signed int, vector signed int);
vector float vec_min (vector float, vector float);

vector float vec_vminfp (vector float, vector float);

vector signed int vec_vminsw (vector bool int, vector signed int);
vector signed int vec_vminsw (vector signed int, vector bool int);
vector signed int vec_vminsw (vector signed int, vector signed int);

vector unsigned int vec_vminuw (vector bool int, vector unsigned int);
vector unsigned int vec_vminuw (vector unsigned int, vector bool int);
vector unsigned int vec_vminuw (vector unsigned int,
                                vector unsigned int);

vector signed short vec_vminsh (vector bool short, vector signed short);
vector signed short vec_vminsh (vector signed short, vector bool short);
vector signed short vec_vminsh (vector signed short,
                                vector signed short);

vector unsigned short vec_vminuh (vector bool short,
                                  vector unsigned short);
vector unsigned short vec_vminuh (vector unsigned short,
                                  vector bool short);
vector unsigned short vec_vminuh (vector unsigned short,
                                  vector unsigned short);

vector signed char vec_vminsb (vector bool char, vector signed char);
vector signed char vec_vminsb (vector signed char, vector bool char);
vector signed char vec_vminsb (vector signed char, vector signed char);
```

```
vector unsigned char vec_vminub (vector bool char,
                                 vector unsigned char);
vector unsigned char vec_vminub (vector unsigned char,
                                 vector bool char);
vector unsigned char vec_vminub (vector unsigned char,
                                 vector unsigned char);

vector signed short vec_mladd (vector signed short,
                               vector signed short,
                               vector signed short);
vector signed short vec_mladd (vector signed short,
                               vector unsigned short,
                               vector unsigned short);
vector signed short vec_mladd (vector unsigned short,
                               vector signed short,
                               vector signed short);
vector unsigned short vec_mladd (vector unsigned short,
                                 vector unsigned short,
                                 vector unsigned short);

vector signed short vec_mradds (vector signed short,
                                vector signed short,
                                vector signed short);

vector unsigned int vec_msum (vector unsigned char,
                              vector unsigned char,
                              vector unsigned int);
vector signed int vec_msum (vector signed char,
                            vector unsigned char,
                            vector signed int);
vector unsigned int vec_msum (vector unsigned short,
                              vector unsigned short,
                              vector unsigned int);
vector signed int vec_msum (vector signed short,
                            vector signed short,
                            vector signed int);

vector signed int vec_vmsumshm (vector signed short,
                                vector signed short,
                                vector signed int);

vector unsigned int vec_vmsumuhm (vector unsigned short,
                                  vector unsigned short,
                                  vector unsigned int);

vector signed int vec_vmsummbm (vector signed char,
                                vector unsigned char,
                                vector signed int);

vector unsigned int vec_vmsumubm (vector unsigned char,
                                  vector unsigned char,
                                  vector unsigned int);

vector unsigned int vec_msums (vector unsigned short,
                               vector unsigned short,
                               vector unsigned int);
vector signed int vec_msums (vector signed short,
```

```
                              vector signed short,
                              vector signed int);

vector signed int vec_vmsumshs (vector signed short,
                                vector signed short,
                                vector signed int);

vector unsigned int vec_vmsumuhs (vector unsigned short,
                                  vector unsigned short,
                                  vector unsigned int);

void vec_mtvscr (vector signed int);
void vec_mtvscr (vector unsigned int);
void vec_mtvscr (vector bool int);
void vec_mtvscr (vector signed short);
void vec_mtvscr (vector unsigned short);
void vec_mtvscr (vector bool short);
void vec_mtvscr (vector pixel);
void vec_mtvscr (vector signed char);
void vec_mtvscr (vector unsigned char);
void vec_mtvscr (vector bool char);

vector unsigned short vec_mule (vector unsigned char,
                                vector unsigned char);
vector signed short vec_mule (vector signed char,
                              vector signed char);
vector unsigned int vec_mule (vector unsigned short,
                              vector unsigned short);
vector signed int vec_mule (vector signed short, vector signed short);

vector signed int vec_vmulesh (vector signed short,
                               vector signed short);

vector unsigned int vec_vmuleuh (vector unsigned short,
                                 vector unsigned short);

vector signed short vec_vmulesb (vector signed char,
                                 vector signed char);

vector unsigned short vec_vmuleub (vector unsigned char,
                                   vector unsigned char);

vector unsigned short vec_mulo (vector unsigned char,
                                vector unsigned char);
vector signed short vec_mulo (vector signed char, vector signed char);
vector unsigned int vec_mulo (vector unsigned short,
                              vector unsigned short);
vector signed int vec_mulo (vector signed short, vector signed short);

vector signed int vec_vmulosh (vector signed short,
                               vector signed short);

vector unsigned int vec_vmulouh (vector unsigned short,
                                 vector unsigned short);

vector signed short vec_vmulosb (vector signed char,
                                 vector signed char);
```

```
vector unsigned short vec_vmuloub (vector unsigned char,
                                   vector unsigned char);

vector float vec_nmsub (vector float, vector float, vector float);

vector float vec_nor (vector float, vector float);
vector signed int vec_nor (vector signed int, vector signed int);
vector unsigned int vec_nor (vector unsigned int, vector unsigned int);
vector bool int vec_nor (vector bool int, vector bool int);
vector signed short vec_nor (vector signed short, vector signed short);
vector unsigned short vec_nor (vector unsigned short,
                               vector unsigned short);
vector bool short vec_nor (vector bool short, vector bool short);
vector signed char vec_nor (vector signed char, vector signed char);
vector unsigned char vec_nor (vector unsigned char,
                              vector unsigned char);
vector bool char vec_nor (vector bool char, vector bool char);

vector float vec_or (vector float, vector float);
vector float vec_or (vector float, vector bool int);
vector float vec_or (vector bool int, vector float);
vector bool int vec_or (vector bool int, vector bool int);
vector signed int vec_or (vector bool int, vector signed int);
vector signed int vec_or (vector signed int, vector bool int);
vector signed int vec_or (vector signed int, vector signed int);
vector unsigned int vec_or (vector bool int, vector unsigned int);
vector unsigned int vec_or (vector unsigned int, vector bool int);
vector unsigned int vec_or (vector unsigned int, vector unsigned int);
vector bool short vec_or (vector bool short, vector bool short);
vector signed short vec_or (vector bool short, vector signed short);
vector signed short vec_or (vector signed short, vector bool short);
vector signed short vec_or (vector signed short, vector signed short);
vector unsigned short vec_or (vector bool short, vector unsigned short);
vector unsigned short vec_or (vector unsigned short, vector bool short);
vector unsigned short vec_or (vector unsigned short,
                              vector unsigned short);
vector signed char vec_or (vector bool char, vector signed char);
vector bool char vec_or (vector bool char, vector bool char);
vector signed char vec_or (vector signed char, vector bool char);
vector signed char vec_or (vector signed char, vector signed char);
vector unsigned char vec_or (vector bool char, vector unsigned char);
vector unsigned char vec_or (vector unsigned char, vector bool char);
vector unsigned char vec_or (vector unsigned char,
                             vector unsigned char);

vector signed char vec_pack (vector signed short, vector signed short);
vector unsigned char vec_pack (vector unsigned short,
                               vector unsigned short);
vector bool char vec_pack (vector bool short, vector bool short);
vector signed short vec_pack (vector signed int, vector signed int);
vector unsigned short vec_pack (vector unsigned int,
                                vector unsigned int);
vector bool short vec_pack (vector bool int, vector bool int);

vector bool short vec_vpkuwum (vector bool int, vector bool int);
vector signed short vec_vpkuwum (vector signed int, vector signed int);
vector unsigned short vec_vpkuwum (vector unsigned int,
                                   vector unsigned int);
```

```
vector bool char vec_vpkuhum (vector bool short, vector bool short);
vector signed char vec_vpkuhum (vector signed short,
                                vector signed short);
vector unsigned char vec_vpkuhum (vector unsigned short,
                                  vector unsigned short);

vector pixel vec_packpx (vector unsigned int, vector unsigned int);

vector unsigned char vec_packs (vector unsigned short,
                                vector unsigned short);
vector signed char vec_packs (vector signed short, vector signed short);
vector unsigned short vec_packs (vector unsigned int,
                                 vector unsigned int);
vector signed short vec_packs (vector signed int, vector signed int);

vector signed short vec_vpkswss (vector signed int, vector signed int);

vector unsigned short vec_vpkuwus (vector unsigned int,
                                   vector unsigned int);

vector signed char vec_vpkshss (vector signed short,
                                vector signed short);

vector unsigned char vec_vpkuhus (vector unsigned short,
                                  vector unsigned short);

vector unsigned char vec_packsu (vector unsigned short,
                                 vector unsigned short);
vector unsigned char vec_packsu (vector signed short,
                                 vector signed short);
vector unsigned short vec_packsu (vector unsigned int,
                                  vector unsigned int);
vector unsigned short vec_packsu (vector signed int, vector signed int);

vector unsigned short vec_vpkswus (vector signed int,
                                   vector signed int);

vector unsigned char vec_vpkshus (vector signed short,
                                  vector signed short);

vector float vec_perm (vector float,
                       vector float,
                       vector unsigned char);
vector signed int vec_perm (vector signed int,
                            vector signed int,
                            vector unsigned char);
vector unsigned int vec_perm (vector unsigned int,
                              vector unsigned int,
                              vector unsigned char);
vector bool int vec_perm (vector bool int,
                          vector bool int,
                          vector unsigned char);
vector signed short vec_perm (vector signed short,
                              vector signed short,
                              vector unsigned char);
vector unsigned short vec_perm (vector unsigned short,
                                vector unsigned short,
```

```
                                vector unsigned char);
vector bool short vec_perm (vector bool short,
                            vector bool short,
                            vector unsigned char);
vector pixel vec_perm (vector pixel,
                       vector pixel,
                       vector unsigned char);
vector signed char vec_perm (vector signed char,
                             vector signed char,
                             vector unsigned char);
vector unsigned char vec_perm (vector unsigned char,
                               vector unsigned char,
                               vector unsigned char);
vector bool char vec_perm (vector bool char,
                           vector bool char,
                           vector unsigned char);

vector float vec_re (vector float);

vector signed char vec_rl (vector signed char,
                           vector unsigned char);
vector unsigned char vec_rl (vector unsigned char,
                             vector unsigned char);
vector signed short vec_rl (vector signed short, vector unsigned short);
vector unsigned short vec_rl (vector unsigned short,
                              vector unsigned short);
vector signed int vec_rl (vector signed int, vector unsigned int);
vector unsigned int vec_rl (vector unsigned int, vector unsigned int);

vector signed int vec_vrlw (vector signed int, vector unsigned int);
vector unsigned int vec_vrlw (vector unsigned int, vector unsigned int);

vector signed short vec_vrlh (vector signed short,
                              vector unsigned short);
vector unsigned short vec_vrlh (vector unsigned short,
                                vector unsigned short);

vector signed char vec_vrlb (vector signed char, vector unsigned char);
vector unsigned char vec_vrlb (vector unsigned char,
                               vector unsigned char);

vector float vec_round (vector float);

vector float vec_recip (vector float, vector float);

vector float vec_rsqrt (vector float);

vector float vec_rsqrte (vector float);

vector float vec_sel (vector float, vector float, vector bool int);
vector float vec_sel (vector float, vector float, vector unsigned int);
vector signed int vec_sel (vector signed int,
                           vector signed int,
                           vector bool int);
vector signed int vec_sel (vector signed int,
                           vector signed int,
                           vector unsigned int);
vector unsigned int vec_sel (vector unsigned int,
```

```
                              vector unsigned int,
                              vector bool int);
vector unsigned int vec_sel (vector unsigned int,
                              vector unsigned int,
                              vector unsigned int);
vector bool int vec_sel (vector bool int,
                         vector bool int,
                         vector bool int);
vector bool int vec_sel (vector bool int,
                         vector bool int,
                         vector unsigned int);
vector signed short vec_sel (vector signed short,
                             vector signed short,
                             vector bool short);
vector signed short vec_sel (vector signed short,
                             vector signed short,
                             vector unsigned short);
vector unsigned short vec_sel (vector unsigned short,
                               vector unsigned short,
                               vector bool short);
vector unsigned short vec_sel (vector unsigned short,
                               vector unsigned short,
                               vector unsigned short);
vector bool short vec_sel (vector bool short,
                           vector bool short,
                           vector bool short);
vector bool short vec_sel (vector bool short,
                           vector bool short,
                           vector unsigned short);
vector signed char vec_sel (vector signed char,
                            vector signed char,
                            vector bool char);
vector signed char vec_sel (vector signed char,
                            vector signed char,
                            vector unsigned char);
vector unsigned char vec_sel (vector unsigned char,
                              vector unsigned char,
                              vector bool char);
vector unsigned char vec_sel (vector unsigned char,
                              vector unsigned char,
                              vector unsigned char);
vector bool char vec_sel (vector bool char,
                          vector bool char,
                          vector bool char);
vector bool char vec_sel (vector bool char,
                          vector bool char,
                          vector unsigned char);

vector signed char vec_sl (vector signed char,
                           vector unsigned char);
vector unsigned char vec_sl (vector unsigned char,
                             vector unsigned char);
vector signed short vec_sl (vector signed short, vector unsigned short);
vector unsigned short vec_sl (vector unsigned short,
                              vector unsigned short);
vector signed int vec_sl (vector signed int, vector unsigned int);
vector unsigned int vec_sl (vector unsigned int, vector unsigned int);
```

```
vector signed int vec_vslw (vector signed int, vector unsigned int);
vector unsigned int vec_vslw (vector unsigned int, vector unsigned int);

vector signed short vec_vslh (vector signed short,
                              vector unsigned short);
vector unsigned short vec_vslh (vector unsigned short,
                                vector unsigned short);

vector signed char vec_vslb (vector signed char, vector unsigned char);
vector unsigned char vec_vslb (vector unsigned char,
                               vector unsigned char);

vector float vec_sld (vector float, vector float, const int);
vector signed int vec_sld (vector signed int,
                           vector signed int,
                           const int);
vector unsigned int vec_sld (vector unsigned int,
                             vector unsigned int,
                             const int);
vector bool int vec_sld (vector bool int,
                         vector bool int,
                         const int);
vector signed short vec_sld (vector signed short,
                             vector signed short,
                             const int);
vector unsigned short vec_sld (vector unsigned short,
                               vector unsigned short,
                               const int);
vector bool short vec_sld (vector bool short,
                           vector bool short,
                           const int);
vector pixel vec_sld (vector pixel,
                      vector pixel,
                      const int);
vector signed char vec_sld (vector signed char,
                            vector signed char,
                            const int);
vector unsigned char vec_sld (vector unsigned char,
                              vector unsigned char,
                              const int);
vector bool char vec_sld (vector bool char,
                          vector bool char,
                          const int);

vector signed int vec_sll (vector signed int,
                           vector unsigned int);
vector signed int vec_sll (vector signed int,
                           vector unsigned short);
vector signed int vec_sll (vector signed int,
                           vector unsigned char);
vector unsigned int vec_sll (vector unsigned int,
                             vector unsigned int);
vector unsigned int vec_sll (vector unsigned int,
                             vector unsigned short);
vector unsigned int vec_sll (vector unsigned int,
                             vector unsigned char);
vector bool int vec_sll (vector bool int,
                         vector unsigned int);
```

```
vector bool int vec_sll (vector bool int,
                         vector unsigned short);
vector bool int vec_sll (vector bool int,
                         vector unsigned char);
vector signed short vec_sll (vector signed short,
                             vector unsigned int);
vector signed short vec_sll (vector signed short,
                             vector unsigned short);
vector signed short vec_sll (vector signed short,
                             vector unsigned char);
vector unsigned short vec_sll (vector unsigned short,
                               vector unsigned int);
vector unsigned short vec_sll (vector unsigned short,
                               vector unsigned short);
vector unsigned short vec_sll (vector unsigned short,
                               vector unsigned char);
vector bool short vec_sll (vector bool short, vector unsigned int);
vector bool short vec_sll (vector bool short, vector unsigned short);
vector bool short vec_sll (vector bool short, vector unsigned char);
vector pixel vec_sll (vector pixel, vector unsigned int);
vector pixel vec_sll (vector pixel, vector unsigned short);
vector pixel vec_sll (vector pixel, vector unsigned char);
vector signed char vec_sll (vector signed char, vector unsigned int);
vector signed char vec_sll (vector signed char, vector unsigned short);
vector signed char vec_sll (vector signed char, vector unsigned char);
vector unsigned char vec_sll (vector unsigned char,
                              vector unsigned int);
vector unsigned char vec_sll (vector unsigned char,
                              vector unsigned short);
vector unsigned char vec_sll (vector unsigned char,
                              vector unsigned char);
vector bool char vec_sll (vector bool char, vector unsigned int);
vector bool char vec_sll (vector bool char, vector unsigned short);
vector bool char vec_sll (vector bool char, vector unsigned char);

vector float vec_slo (vector float, vector signed char);
vector float vec_slo (vector float, vector unsigned char);
vector signed int vec_slo (vector signed int, vector signed char);
vector signed int vec_slo (vector signed int, vector unsigned char);
vector unsigned int vec_slo (vector unsigned int, vector signed char);
vector unsigned int vec_slo (vector unsigned int, vector unsigned char);
vector signed short vec_slo (vector signed short, vector signed char);
vector signed short vec_slo (vector signed short, vector unsigned char);
vector unsigned short vec_slo (vector unsigned short,
                               vector signed char);
vector unsigned short vec_slo (vector unsigned short,
                               vector unsigned char);
vector pixel vec_slo (vector pixel, vector signed char);
vector pixel vec_slo (vector pixel, vector unsigned char);
vector signed char vec_slo (vector signed char, vector signed char);
vector signed char vec_slo (vector signed char, vector unsigned char);
vector unsigned char vec_slo (vector unsigned char, vector signed char);
vector unsigned char vec_slo (vector unsigned char,
                              vector unsigned char);

vector signed char vec_splat (vector signed char, const int);
vector unsigned char vec_splat (vector unsigned char, const int);
vector bool char vec_splat (vector bool char, const int);
```

```
vector signed short vec_splat (vector signed short, const int);
vector unsigned short vec_splat (vector unsigned short, const int);
vector bool short vec_splat (vector bool short, const int);
vector pixel vec_splat (vector pixel, const int);
vector float vec_splat (vector float, const int);
vector signed int vec_splat (vector signed int, const int);
vector unsigned int vec_splat (vector unsigned int, const int);
vector bool int vec_splat (vector bool int, const int);

vector float vec_vspltw (vector float, const int);
vector signed int vec_vspltw (vector signed int, const int);
vector unsigned int vec_vspltw (vector unsigned int, const int);
vector bool int vec_vspltw (vector bool int, const int);

vector bool short vec_vsplth (vector bool short, const int);
vector signed short vec_vsplth (vector signed short, const int);
vector unsigned short vec_vsplth (vector unsigned short, const int);
vector pixel vec_vsplth (vector pixel, const int);

vector signed char vec_vspltb (vector signed char, const int);
vector unsigned char vec_vspltb (vector unsigned char, const int);
vector bool char vec_vspltb (vector bool char, const int);

vector signed char vec_splat_s8 (const int);

vector signed short vec_splat_s16 (const int);

vector signed int vec_splat_s32 (const int);

vector unsigned char vec_splat_u8 (const int);

vector unsigned short vec_splat_u16 (const int);

vector unsigned int vec_splat_u32 (const int);

vector signed char vec_sr (vector signed char, vector unsigned char);
vector unsigned char vec_sr (vector unsigned char,
                             vector unsigned char);
vector signed short vec_sr (vector signed short,
                            vector unsigned short);
vector unsigned short vec_sr (vector unsigned short,
                              vector unsigned short);
vector signed int vec_sr (vector signed int, vector unsigned int);
vector unsigned int vec_sr (vector unsigned int, vector unsigned int);

vector signed int vec_vsrw (vector signed int, vector unsigned int);
vector unsigned int vec_vsrw (vector unsigned int, vector unsigned int);

vector signed short vec_vsrh (vector signed short,
                              vector unsigned short);
vector unsigned short vec_vsrh (vector unsigned short,
                                vector unsigned short);

vector signed char vec_vsrb (vector signed char, vector unsigned char);
vector unsigned char vec_vsrb (vector unsigned char,
                               vector unsigned char);

vector signed char vec_sra (vector signed char, vector unsigned char);
```

```
vector unsigned char vec_sra (vector unsigned char,
                              vector unsigned char);
vector signed short vec_sra (vector signed short,
                             vector unsigned short);
vector unsigned short vec_sra (vector unsigned short,
                               vector unsigned short);
vector signed int vec_sra (vector signed int, vector unsigned int);
vector unsigned int vec_sra (vector unsigned int, vector unsigned int);

vector signed int vec_vsraw (vector signed int, vector unsigned int);
vector unsigned int vec_vsraw (vector unsigned int,
                               vector unsigned int);

vector signed short vec_vsrah (vector signed short,
                               vector unsigned short);
vector unsigned short vec_vsrah (vector unsigned short,
                                 vector unsigned short);

vector signed char vec_vsrab (vector signed char, vector unsigned char);
vector unsigned char vec_vsrab (vector unsigned char,
                                vector unsigned char);

vector signed int vec_srl (vector signed int, vector unsigned int);
vector signed int vec_srl (vector signed int, vector unsigned short);
vector signed int vec_srl (vector signed int, vector unsigned char);
vector unsigned int vec_srl (vector unsigned int, vector unsigned int);
vector unsigned int vec_srl (vector unsigned int,
                             vector unsigned short);
vector unsigned int vec_srl (vector unsigned int, vector unsigned char);
vector bool int vec_srl (vector bool int, vector unsigned int);
vector bool int vec_srl (vector bool int, vector unsigned short);
vector bool int vec_srl (vector bool int, vector unsigned char);
vector signed short vec_srl (vector signed short, vector unsigned int);
vector signed short vec_srl (vector signed short,
                             vector unsigned short);
vector signed short vec_srl (vector signed short, vector unsigned char);
vector unsigned short vec_srl (vector unsigned short,
                               vector unsigned int);
vector unsigned short vec_srl (vector unsigned short,
                               vector unsigned short);
vector unsigned short vec_srl (vector unsigned short,
                               vector unsigned char);
vector bool short vec_srl (vector bool short, vector unsigned int);
vector bool short vec_srl (vector bool short, vector unsigned short);
vector bool short vec_srl (vector bool short, vector unsigned char);
vector pixel vec_srl (vector pixel, vector unsigned int);
vector pixel vec_srl (vector pixel, vector unsigned short);
vector pixel vec_srl (vector pixel, vector unsigned char);
vector signed char vec_srl (vector signed char, vector unsigned int);
vector signed char vec_srl (vector signed char, vector unsigned short);
vector signed char vec_srl (vector signed char, vector unsigned char);
vector unsigned char vec_srl (vector unsigned char,
                              vector unsigned int);
vector unsigned char vec_srl (vector unsigned char,
                              vector unsigned short);
vector unsigned char vec_srl (vector unsigned char,
                              vector unsigned char);
vector bool char vec_srl (vector bool char, vector unsigned int);
```

```
vector bool char vec_srl (vector bool char, vector unsigned short);
vector bool char vec_srl (vector bool char, vector unsigned char);

vector float vec_sro (vector float, vector signed char);
vector float vec_sro (vector float, vector unsigned char);
vector signed int vec_sro (vector signed int, vector signed char);
vector signed int vec_sro (vector signed int, vector unsigned char);
vector unsigned int vec_sro (vector unsigned int, vector signed char);
vector unsigned int vec_sro (vector unsigned int, vector unsigned char);
vector signed short vec_sro (vector signed short, vector signed char);
vector signed short vec_sro (vector signed short, vector unsigned char);
vector unsigned short vec_sro (vector unsigned short,
                               vector signed char);
vector unsigned short vec_sro (vector unsigned short,
                               vector unsigned char);
vector pixel vec_sro (vector pixel, vector signed char);
vector pixel vec_sro (vector pixel, vector unsigned char);
vector signed char vec_sro (vector signed char, vector signed char);
vector signed char vec_sro (vector signed char, vector unsigned char);
vector unsigned char vec_sro (vector unsigned char, vector signed char);
vector unsigned char vec_sro (vector unsigned char,
                              vector unsigned char);

void vec_st (vector float, int, vector float *);
void vec_st (vector float, int, float *);
void vec_st (vector signed int, int, vector signed int *);
void vec_st (vector signed int, int, int *);
void vec_st (vector unsigned int, int, vector unsigned int *);
void vec_st (vector unsigned int, int, unsigned int *);
void vec_st (vector bool int, int, vector bool int *);
void vec_st (vector bool int, int, unsigned int *);
void vec_st (vector bool int, int, int *);
void vec_st (vector signed short, int, vector signed short *);
void vec_st (vector signed short, int, short *);
void vec_st (vector unsigned short, int, vector unsigned short *);
void vec_st (vector unsigned short, int, unsigned short *);
void vec_st (vector bool short, int, vector bool short *);
void vec_st (vector bool short, int, unsigned short *);
void vec_st (vector pixel, int, vector pixel *);
void vec_st (vector pixel, int, unsigned short *);
void vec_st (vector pixel, int, short *);
void vec_st (vector bool short, int, short *);
void vec_st (vector signed char, int, vector signed char *);
void vec_st (vector signed char, int, signed char *);
void vec_st (vector unsigned char, int, vector unsigned char *);
void vec_st (vector unsigned char, int, unsigned char *);
void vec_st (vector bool char, int, vector bool char *);
void vec_st (vector bool char, int, unsigned char *);
void vec_st (vector bool char, int, signed char *);

void vec_ste (vector signed char, int, signed char *);
void vec_ste (vector unsigned char, int, unsigned char *);
void vec_ste (vector bool char, int, signed char *);
void vec_ste (vector bool char, int, unsigned char *);
void vec_ste (vector signed short, int, short *);
void vec_ste (vector unsigned short, int, unsigned short *);
void vec_ste (vector bool short, int, short *);
void vec_ste (vector bool short, int, unsigned short *);
```

```
void vec_ste (vector pixel, int, short *);
void vec_ste (vector pixel, int, unsigned short *);
void vec_ste (vector float, int, float *);
void vec_ste (vector signed int, int, int *);
void vec_ste (vector unsigned int, int, unsigned int *);
void vec_ste (vector bool int, int, int *);
void vec_ste (vector bool int, int, unsigned int *);

void vec_stvewx (vector float, int, float *);
void vec_stvewx (vector signed int, int, int *);
void vec_stvewx (vector unsigned int, int, unsigned int *);
void vec_stvewx (vector bool int, int, int *);
void vec_stvewx (vector bool int, int, unsigned int *);

void vec_stvehx (vector signed short, int, short *);
void vec_stvehx (vector unsigned short, int, unsigned short *);
void vec_stvehx (vector bool short, int, short *);
void vec_stvehx (vector bool short, int, unsigned short *);
void vec_stvehx (vector pixel, int, short *);
void vec_stvehx (vector pixel, int, unsigned short *);

void vec_stvebx (vector signed char, int, signed char *);
void vec_stvebx (vector unsigned char, int, unsigned char *);
void vec_stvebx (vector bool char, int, signed char *);
void vec_stvebx (vector bool char, int, unsigned char *);

void vec_stl (vector float, int, vector float *);
void vec_stl (vector float, int, float *);
void vec_stl (vector signed int, int, vector signed int *);
void vec_stl (vector signed int, int, int *);
void vec_stl (vector unsigned int, int, vector unsigned int *);
void vec_stl (vector unsigned int, int, unsigned int *);
void vec_stl (vector bool int, int, vector bool int *);
void vec_stl (vector bool int, int, unsigned int *);
void vec_stl (vector bool int, int, int *);
void vec_stl (vector signed short, int, vector signed short *);
void vec_stl (vector signed short, int, short *);
void vec_stl (vector unsigned short, int, vector unsigned short *);
void vec_stl (vector unsigned short, int, unsigned short *);
void vec_stl (vector bool short, int, vector bool short *);
void vec_stl (vector bool short, int, unsigned short *);
void vec_stl (vector bool short, int, short *);
void vec_stl (vector pixel, int, vector pixel *);
void vec_stl (vector pixel, int, unsigned short *);
void vec_stl (vector pixel, int, short *);
void vec_stl (vector signed char, int, vector signed char *);
void vec_stl (vector signed char, int, signed char *);
void vec_stl (vector unsigned char, int, vector unsigned char *);
void vec_stl (vector unsigned char, int, unsigned char *);
void vec_stl (vector bool char, int, vector bool char *);
void vec_stl (vector bool char, int, unsigned char *);
void vec_stl (vector bool char, int, signed char *);

vector signed char vec_sub (vector bool char, vector signed char);
vector signed char vec_sub (vector signed char, vector bool char);
vector signed char vec_sub (vector signed char, vector signed char);
vector unsigned char vec_sub (vector bool char, vector unsigned char);
vector unsigned char vec_sub (vector unsigned char, vector bool char);
```

```
vector unsigned char vec_sub (vector unsigned char,
                              vector unsigned char);
vector signed short vec_sub (vector bool short, vector signed short);
vector signed short vec_sub (vector signed short, vector bool short);
vector signed short vec_sub (vector signed short, vector signed short);
vector unsigned short vec_sub (vector bool short,
                               vector unsigned short);
vector unsigned short vec_sub (vector unsigned short,
                               vector bool short);
vector unsigned short vec_sub (vector unsigned short,
                               vector unsigned short);
vector signed int vec_sub (vector bool int, vector signed int);
vector signed int vec_sub (vector signed int, vector bool int);
vector signed int vec_sub (vector signed int, vector signed int);
vector unsigned int vec_sub (vector bool int, vector unsigned int);
vector unsigned int vec_sub (vector unsigned int, vector bool int);
vector unsigned int vec_sub (vector unsigned int, vector unsigned int);
vector float vec_sub (vector float, vector float);

vector float vec_vsubfp (vector float, vector float);

vector signed int vec_vsubuwm (vector bool int, vector signed int);
vector signed int vec_vsubuwm (vector signed int, vector bool int);
vector signed int vec_vsubuwm (vector signed int, vector signed int);
vector unsigned int vec_vsubuwm (vector bool int, vector unsigned int);
vector unsigned int vec_vsubuwm (vector unsigned int, vector bool int);
vector unsigned int vec_vsubuwm (vector unsigned int,
                                 vector unsigned int);

vector signed short vec_vsubuhm (vector bool short,
                                 vector signed short);
vector signed short vec_vsubuhm (vector signed short,
                                 vector bool short);
vector signed short vec_vsubuhm (vector signed short,
                                 vector signed short);
vector unsigned short vec_vsubuhm (vector bool short,
                                   vector unsigned short);
vector unsigned short vec_vsubuhm (vector unsigned short,
                                   vector bool short);
vector unsigned short vec_vsubuhm (vector unsigned short,
                                   vector unsigned short);

vector signed char vec_vsububm (vector bool char, vector signed char);
vector signed char vec_vsububm (vector signed char, vector bool char);
vector signed char vec_vsububm (vector signed char, vector signed char);
vector unsigned char vec_vsububm (vector bool char,
                                  vector unsigned char);
vector unsigned char vec_vsububm (vector unsigned char,
                                  vector bool char);
vector unsigned char vec_vsububm (vector unsigned char,
                                  vector unsigned char);

vector unsigned int vec_subc (vector unsigned int, vector unsigned int);

vector unsigned char vec_subs (vector bool char, vector unsigned char);
vector unsigned char vec_subs (vector unsigned char, vector bool char);
vector unsigned char vec_subs (vector unsigned char,
                               vector unsigned char);
```

```
vector signed char vec_subs (vector bool char, vector signed char);
vector signed char vec_subs (vector signed char, vector bool char);
vector signed char vec_subs (vector signed char, vector signed char);
vector unsigned short vec_subs (vector bool short,
                                vector unsigned short);
vector unsigned short vec_subs (vector unsigned short,
                                vector bool short);
vector unsigned short vec_subs (vector unsigned short,
                                vector unsigned short);
vector signed short vec_subs (vector bool short, vector signed short);
vector signed short vec_subs (vector signed short, vector bool short);
vector signed short vec_subs (vector signed short, vector signed short);
vector unsigned int vec_subs (vector bool int, vector unsigned int);
vector unsigned int vec_subs (vector unsigned int, vector bool int);
vector unsigned int vec_subs (vector unsigned int, vector unsigned int);
vector signed int vec_subs (vector bool int, vector signed int);
vector signed int vec_subs (vector signed int, vector bool int);
vector signed int vec_subs (vector signed int, vector signed int);

vector signed int vec_vsubsws (vector bool int, vector signed int);
vector signed int vec_vsubsws (vector signed int, vector bool int);
vector signed int vec_vsubsws (vector signed int, vector signed int);

vector unsigned int vec_vsubuws (vector bool int, vector unsigned int);
vector unsigned int vec_vsubuws (vector unsigned int, vector bool int);
vector unsigned int vec_vsubuws (vector unsigned int,
                                 vector unsigned int);

vector signed short vec_vsubshs (vector bool short,
                                 vector signed short);
vector signed short vec_vsubshs (vector signed short,
                                 vector bool short);
vector signed short vec_vsubshs (vector signed short,
                                 vector signed short);

vector unsigned short vec_vsubuhs (vector bool short,
                                   vector unsigned short);
vector unsigned short vec_vsubuhs (vector unsigned short,
                                   vector bool short);
vector unsigned short vec_vsubuhs (vector unsigned short,
                                   vector unsigned short);

vector signed char vec_vsubsbs (vector bool char, vector signed char);
vector signed char vec_vsubsbs (vector signed char, vector bool char);
vector signed char vec_vsubsbs (vector signed char, vector signed char);

vector unsigned char vec_vsububs (vector bool char,
                                  vector unsigned char);
vector unsigned char vec_vsububs (vector unsigned char,
                                  vector bool char);
vector unsigned char vec_vsububs (vector unsigned char,
                                  vector unsigned char);

vector unsigned int vec_sum4s (vector unsigned char,
                               vector unsigned int);
vector signed int vec_sum4s (vector signed char, vector signed int);
vector signed int vec_sum4s (vector signed short, vector signed int);
```

```
vector signed int vec_vsum4shs (vector signed short, vector signed int);

vector signed int vec_vsum4sbs (vector signed char, vector signed int);

vector unsigned int vec_vsum4ubs (vector unsigned char,
                                  vector unsigned int);

vector signed int vec_sum2s (vector signed int, vector signed int);

vector signed int vec_sums (vector signed int, vector signed int);

vector float vec_trunc (vector float);

vector signed short vec_unpackh (vector signed char);
vector bool short vec_unpackh (vector bool char);
vector signed int vec_unpackh (vector signed short);
vector bool int vec_unpackh (vector bool short);
vector unsigned int vec_unpackh (vector pixel);

vector bool int vec_vupkhsh (vector bool short);
vector signed int vec_vupkhsh (vector signed short);

vector unsigned int vec_vupkhpx (vector pixel);

vector bool short vec_vupkhsb (vector bool char);
vector signed short vec_vupkhsb (vector signed char);

vector signed short vec_unpackl (vector signed char);
vector bool short vec_unpackl (vector bool char);
vector unsigned int vec_unpackl (vector pixel);
vector signed int vec_unpackl (vector signed short);
vector bool int vec_unpackl (vector bool short);

vector unsigned int vec_vupklpx (vector pixel);

vector bool int vec_vupklsh (vector bool short);
vector signed int vec_vupklsh (vector signed short);

vector bool short vec_vupklsb (vector bool char);
vector signed short vec_vupklsb (vector signed char);

vector float vec_xor (vector float, vector float);
vector float vec_xor (vector float, vector bool int);
vector float vec_xor (vector bool int, vector float);
vector bool int vec_xor (vector bool int, vector bool int);
vector signed int vec_xor (vector bool int, vector signed int);
vector signed int vec_xor (vector signed int, vector bool int);
vector signed int vec_xor (vector signed int, vector signed int);
vector unsigned int vec_xor (vector bool int, vector unsigned int);
vector unsigned int vec_xor (vector unsigned int, vector bool int);
vector unsigned int vec_xor (vector unsigned int, vector unsigned int);
vector bool short vec_xor (vector bool short, vector bool short);
vector signed short vec_xor (vector bool short, vector signed short);
vector signed short vec_xor (vector signed short, vector bool short);
vector signed short vec_xor (vector signed short, vector signed short);
vector unsigned short vec_xor (vector bool short,
                               vector unsigned short);
vector unsigned short vec_xor (vector unsigned short,
```

```
                                vector bool short);
vector unsigned short vec_xor (vector unsigned short,
                               vector unsigned short);
vector signed char vec_xor (vector bool char, vector signed char);
vector bool char vec_xor (vector bool char, vector bool char);
vector signed char vec_xor (vector signed char, vector bool char);
vector signed char vec_xor (vector signed char, vector signed char);
vector unsigned char vec_xor (vector bool char, vector unsigned char);
vector unsigned char vec_xor (vector unsigned char, vector bool char);
vector unsigned char vec_xor (vector unsigned char,
                              vector unsigned char);

int vec_all_eq (vector signed char, vector bool char);
int vec_all_eq (vector signed char, vector signed char);
int vec_all_eq (vector unsigned char, vector bool char);
int vec_all_eq (vector unsigned char, vector unsigned char);
int vec_all_eq (vector bool char, vector bool char);
int vec_all_eq (vector bool char, vector unsigned char);
int vec_all_eq (vector bool char, vector signed char);
int vec_all_eq (vector signed short, vector bool short);
int vec_all_eq (vector signed short, vector signed short);
int vec_all_eq (vector unsigned short, vector bool short);
int vec_all_eq (vector unsigned short, vector unsigned short);
int vec_all_eq (vector bool short, vector bool short);
int vec_all_eq (vector bool short, vector unsigned short);
int vec_all_eq (vector bool short, vector signed short);
int vec_all_eq (vector pixel, vector pixel);
int vec_all_eq (vector signed int, vector bool int);
int vec_all_eq (vector signed int, vector signed int);
int vec_all_eq (vector unsigned int, vector bool int);
int vec_all_eq (vector unsigned int, vector unsigned int);
int vec_all_eq (vector bool int, vector bool int);
int vec_all_eq (vector bool int, vector unsigned int);
int vec_all_eq (vector bool int, vector signed int);
int vec_all_eq (vector float, vector float);

int vec_all_ge (vector bool char, vector unsigned char);
int vec_all_ge (vector unsigned char, vector bool char);
int vec_all_ge (vector unsigned char, vector unsigned char);
int vec_all_ge (vector bool char, vector signed char);
int vec_all_ge (vector signed char, vector bool char);
int vec_all_ge (vector signed char, vector signed char);
int vec_all_ge (vector bool short, vector unsigned short);
int vec_all_ge (vector unsigned short, vector bool short);
int vec_all_ge (vector unsigned short, vector unsigned short);
int vec_all_ge (vector signed short, vector signed short);
int vec_all_ge (vector bool short, vector signed short);
int vec_all_ge (vector signed short, vector bool short);
int vec_all_ge (vector bool int, vector unsigned int);
int vec_all_ge (vector unsigned int, vector bool int);
int vec_all_ge (vector unsigned int, vector unsigned int);
int vec_all_ge (vector bool int, vector signed int);
int vec_all_ge (vector signed int, vector bool int);
int vec_all_ge (vector signed int, vector signed int);
int vec_all_ge (vector float, vector float);

int vec_all_gt (vector bool char, vector unsigned char);
int vec_all_gt (vector unsigned char, vector bool char);
```

```
int vec_all_gt (vector unsigned char, vector unsigned char);
int vec_all_gt (vector bool char, vector signed char);
int vec_all_gt (vector signed char, vector bool char);
int vec_all_gt (vector signed char, vector signed char);
int vec_all_gt (vector bool short, vector unsigned short);
int vec_all_gt (vector unsigned short, vector bool short);
int vec_all_gt (vector unsigned short, vector unsigned short);
int vec_all_gt (vector bool short, vector signed short);
int vec_all_gt (vector signed short, vector bool short);
int vec_all_gt (vector signed short, vector signed short);
int vec_all_gt (vector bool int, vector unsigned int);
int vec_all_gt (vector unsigned int, vector bool int);
int vec_all_gt (vector unsigned int, vector unsigned int);
int vec_all_gt (vector bool int, vector signed int);
int vec_all_gt (vector signed int, vector bool int);
int vec_all_gt (vector signed int, vector signed int);
int vec_all_gt (vector float, vector float);

int vec_all_in (vector float, vector float);

int vec_all_le (vector bool char, vector unsigned char);
int vec_all_le (vector unsigned char, vector bool char);
int vec_all_le (vector unsigned char, vector unsigned char);
int vec_all_le (vector bool char, vector signed char);
int vec_all_le (vector signed char, vector bool char);
int vec_all_le (vector signed char, vector signed char);
int vec_all_le (vector bool short, vector unsigned short);
int vec_all_le (vector unsigned short, vector bool short);
int vec_all_le (vector unsigned short, vector unsigned short);
int vec_all_le (vector bool short, vector signed short);
int vec_all_le (vector signed short, vector bool short);
int vec_all_le (vector signed short, vector signed short);
int vec_all_le (vector bool int, vector unsigned int);
int vec_all_le (vector unsigned int, vector bool int);
int vec_all_le (vector unsigned int, vector unsigned int);
int vec_all_le (vector bool int, vector signed int);
int vec_all_le (vector signed int, vector bool int);
int vec_all_le (vector signed int, vector signed int);
int vec_all_le (vector float, vector float);

int vec_all_lt (vector bool char, vector unsigned char);
int vec_all_lt (vector unsigned char, vector bool char);
int vec_all_lt (vector unsigned char, vector unsigned char);
int vec_all_lt (vector bool char, vector signed char);
int vec_all_lt (vector signed char, vector bool char);
int vec_all_lt (vector signed char, vector signed char);
int vec_all_lt (vector bool short, vector unsigned short);
int vec_all_lt (vector unsigned short, vector bool short);
int vec_all_lt (vector unsigned short, vector unsigned short);
int vec_all_lt (vector bool short, vector signed short);
int vec_all_lt (vector signed short, vector bool short);
int vec_all_lt (vector signed short, vector signed short);
int vec_all_lt (vector bool int, vector unsigned int);
int vec_all_lt (vector unsigned int, vector bool int);
int vec_all_lt (vector unsigned int, vector unsigned int);
int vec_all_lt (vector bool int, vector signed int);
int vec_all_lt (vector signed int, vector bool int);
int vec_all_lt (vector signed int, vector signed int);
```

```
int vec_all_lt (vector float, vector float);

int vec_all_nan (vector float);

int vec_all_ne (vector signed char, vector bool char);
int vec_all_ne (vector signed char, vector signed char);
int vec_all_ne (vector unsigned char, vector bool char);
int vec_all_ne (vector unsigned char, vector unsigned char);
int vec_all_ne (vector bool char, vector bool char);
int vec_all_ne (vector bool char, vector unsigned char);
int vec_all_ne (vector bool char, vector signed char);
int vec_all_ne (vector signed short, vector bool short);
int vec_all_ne (vector signed short, vector signed short);
int vec_all_ne (vector unsigned short, vector bool short);
int vec_all_ne (vector unsigned short, vector unsigned short);
int vec_all_ne (vector bool short, vector bool short);
int vec_all_ne (vector bool short, vector unsigned short);
int vec_all_ne (vector bool short, vector signed short);
int vec_all_ne (vector pixel, vector pixel);
int vec_all_ne (vector signed int, vector bool int);
int vec_all_ne (vector signed int, vector signed int);
int vec_all_ne (vector unsigned int, vector bool int);
int vec_all_ne (vector unsigned int, vector unsigned int);
int vec_all_ne (vector bool int, vector bool int);
int vec_all_ne (vector bool int, vector unsigned int);
int vec_all_ne (vector bool int, vector signed int);
int vec_all_ne (vector float, vector float);

int vec_all_nge (vector float, vector float);

int vec_all_ngt (vector float, vector float);

int vec_all_nle (vector float, vector float);

int vec_all_nlt (vector float, vector float);

int vec_all_numeric (vector float);

int vec_any_eq (vector signed char, vector bool char);
int vec_any_eq (vector signed char, vector signed char);
int vec_any_eq (vector unsigned char, vector bool char);
int vec_any_eq (vector unsigned char, vector unsigned char);
int vec_any_eq (vector bool char, vector bool char);
int vec_any_eq (vector bool char, vector unsigned char);
int vec_any_eq (vector bool char, vector signed char);
int vec_any_eq (vector signed short, vector bool short);
int vec_any_eq (vector signed short, vector signed short);
int vec_any_eq (vector unsigned short, vector bool short);
int vec_any_eq (vector unsigned short, vector unsigned short);
int vec_any_eq (vector bool short, vector bool short);
int vec_any_eq (vector bool short, vector unsigned short);
int vec_any_eq (vector bool short, vector signed short);
int vec_any_eq (vector pixel, vector pixel);
int vec_any_eq (vector signed int, vector bool int);
int vec_any_eq (vector signed int, vector signed int);
int vec_any_eq (vector unsigned int, vector bool int);
int vec_any_eq (vector unsigned int, vector unsigned int);
int vec_any_eq (vector bool int, vector bool int);
```

```
int vec_any_eq (vector bool int, vector unsigned int);
int vec_any_eq (vector bool int, vector signed int);
int vec_any_eq (vector float, vector float);

int vec_any_ge (vector signed char, vector bool char);
int vec_any_ge (vector unsigned char, vector bool char);
int vec_any_ge (vector unsigned char, vector unsigned char);
int vec_any_ge (vector signed char, vector signed char);
int vec_any_ge (vector bool char, vector unsigned char);
int vec_any_ge (vector bool char, vector signed char);
int vec_any_ge (vector unsigned short, vector bool short);
int vec_any_ge (vector unsigned short, vector unsigned short);
int vec_any_ge (vector signed short, vector signed short);
int vec_any_ge (vector signed short, vector bool short);
int vec_any_ge (vector bool short, vector unsigned short);
int vec_any_ge (vector bool short, vector signed short);
int vec_any_ge (vector signed int, vector bool int);
int vec_any_ge (vector unsigned int, vector bool int);
int vec_any_ge (vector unsigned int, vector unsigned int);
int vec_any_ge (vector signed int, vector signed int);
int vec_any_ge (vector bool int, vector unsigned int);
int vec_any_ge (vector bool int, vector signed int);
int vec_any_ge (vector float, vector float);

int vec_any_gt (vector bool char, vector unsigned char);
int vec_any_gt (vector unsigned char, vector bool char);
int vec_any_gt (vector unsigned char, vector unsigned char);
int vec_any_gt (vector bool char, vector signed char);
int vec_any_gt (vector signed char, vector bool char);
int vec_any_gt (vector signed char, vector signed char);
int vec_any_gt (vector bool short, vector unsigned short);
int vec_any_gt (vector unsigned short, vector bool short);
int vec_any_gt (vector unsigned short, vector unsigned short);
int vec_any_gt (vector bool short, vector signed short);
int vec_any_gt (vector signed short, vector bool short);
int vec_any_gt (vector signed short, vector signed short);
int vec_any_gt (vector bool int, vector unsigned int);
int vec_any_gt (vector unsigned int, vector bool int);
int vec_any_gt (vector unsigned int, vector unsigned int);
int vec_any_gt (vector bool int, vector signed int);
int vec_any_gt (vector signed int, vector bool int);
int vec_any_gt (vector signed int, vector signed int);
int vec_any_gt (vector float, vector float);

int vec_any_le (vector bool char, vector unsigned char);
int vec_any_le (vector unsigned char, vector bool char);
int vec_any_le (vector unsigned char, vector unsigned char);
int vec_any_le (vector bool char, vector signed char);
int vec_any_le (vector signed char, vector bool char);
int vec_any_le (vector signed char, vector signed char);
int vec_any_le (vector bool short, vector unsigned short);
int vec_any_le (vector unsigned short, vector bool short);
int vec_any_le (vector unsigned short, vector unsigned short);
int vec_any_le (vector bool short, vector signed short);
int vec_any_le (vector signed short, vector bool short);
int vec_any_le (vector signed short, vector signed short);
int vec_any_le (vector bool int, vector unsigned int);
int vec_any_le (vector unsigned int, vector bool int);
```

```
int vec_any_le (vector unsigned int, vector unsigned int);
int vec_any_le (vector bool int, vector signed int);
int vec_any_le (vector signed int, vector bool int);
int vec_any_le (vector signed int, vector signed int);
int vec_any_le (vector float, vector float);

int vec_any_lt (vector bool char, vector unsigned char);
int vec_any_lt (vector unsigned char, vector bool char);
int vec_any_lt (vector unsigned char, vector unsigned char);
int vec_any_lt (vector bool char, vector signed char);
int vec_any_lt (vector signed char, vector bool char);
int vec_any_lt (vector signed char, vector signed char);
int vec_any_lt (vector bool short, vector unsigned short);
int vec_any_lt (vector unsigned short, vector bool short);
int vec_any_lt (vector unsigned short, vector unsigned short);
int vec_any_lt (vector bool short, vector signed short);
int vec_any_lt (vector signed short, vector bool short);
int vec_any_lt (vector signed short, vector signed short);
int vec_any_lt (vector bool int, vector unsigned int);
int vec_any_lt (vector unsigned int, vector bool int);
int vec_any_lt (vector unsigned int, vector unsigned int);
int vec_any_lt (vector bool int, vector signed int);
int vec_any_lt (vector signed int, vector bool int);
int vec_any_lt (vector signed int, vector signed int);
int vec_any_lt (vector float, vector float);

int vec_any_nan (vector float);

int vec_any_ne (vector signed char, vector bool char);
int vec_any_ne (vector signed char, vector signed char);
int vec_any_ne (vector unsigned char, vector bool char);
int vec_any_ne (vector unsigned char, vector unsigned char);
int vec_any_ne (vector bool char, vector bool char);
int vec_any_ne (vector bool char, vector unsigned char);
int vec_any_ne (vector bool char, vector signed char);
int vec_any_ne (vector signed short, vector bool short);
int vec_any_ne (vector signed short, vector signed short);
int vec_any_ne (vector unsigned short, vector bool short);
int vec_any_ne (vector unsigned short, vector unsigned short);
int vec_any_ne (vector bool short, vector bool short);
int vec_any_ne (vector bool short, vector unsigned short);
int vec_any_ne (vector bool short, vector signed short);
int vec_any_ne (vector pixel, vector pixel);
int vec_any_ne (vector signed int, vector bool int);
int vec_any_ne (vector signed int, vector signed int);
int vec_any_ne (vector unsigned int, vector bool int);
int vec_any_ne (vector unsigned int, vector unsigned int);
int vec_any_ne (vector bool int, vector bool int);
int vec_any_ne (vector bool int, vector unsigned int);
int vec_any_ne (vector bool int, vector signed int);
int vec_any_ne (vector float, vector float);

int vec_any_nge (vector float, vector float);

int vec_any_ngt (vector float, vector float);

int vec_any_nle (vector float, vector float);
```

```
int vec_any_nlt (vector float, vector float);

int vec_any_numeric (vector float);

int vec_any_out (vector float, vector float);
```

If the vector/scalar (VSX) instruction set is available, the following additional functions are available:

```
vector double vec_abs (vector double);
vector double vec_add (vector double, vector double);
vector double vec_and (vector double, vector double);
vector double vec_and (vector double, vector bool long);
vector double vec_and (vector bool long, vector double);
vector double vec_andc (vector double, vector double);
vector double vec_andc (vector double, vector bool long);
vector double vec_andc (vector bool long, vector double);
vector double vec_ceil (vector double);
vector bool long vec_cmpeq (vector double, vector double);
vector bool long vec_cmpge (vector double, vector double);
vector bool long vec_cmpgt (vector double, vector double);
vector bool long vec_cmple (vector double, vector double);
vector bool long vec_cmplt (vector double, vector double);
vector float vec_div (vector float, vector float);
vector double vec_div (vector double, vector double);
vector double vec_floor (vector double);
vector double vec_ld (int, const vector double *);
vector double vec_ld (int, const double *);
vector double vec_ldl (int, const vector double *);
vector double vec_ldl (int, const double *);
vector unsigned char vec_lvsl (int, const volatile double *);
vector unsigned char vec_lvsr (int, const volatile double *);
vector double vec_madd (vector double, vector double, vector double);
vector double vec_max (vector double, vector double);
vector double vec_min (vector double, vector double);
vector float vec_msub (vector float, vector float, vector float);
vector double vec_msub (vector double, vector double, vector double);
vector float vec_mul (vector float, vector float);
vector double vec_mul (vector double, vector double);
vector float vec_nearbyint (vector float);
vector double vec_nearbyint (vector double);
vector float vec_nmadd (vector float, vector float, vector float);
vector double vec_nmadd (vector double, vector double, vector double);
vector double vec_nmsub (vector double, vector double, vector double);
vector double vec_nor (vector double, vector double);
vector double vec_or (vector double, vector double);
vector double vec_or (vector double, vector bool long);
vector double vec_or (vector bool long, vector double);
vector double vec_perm (vector double,
                        vector double,
                        vector unsigned char);
vector double vec_rint (vector double);
vector double vec_recip (vector double, vector double);
vector double vec_rsqrt (vector double);
vector double vec_rsqrte (vector double);
vector double vec_sel (vector double, vector double, vector bool long);
vector double vec_sel (vector double, vector double, vector unsigned long);
vector double vec_sub (vector double, vector double);
vector float vec_sqrt (vector float);
```

```
vector double vec_sqrt (vector double);
void vec_st (vector double, int, vector double *);
void vec_st (vector double, int, double *);
vector double vec_trunc (vector double);
vector double vec_xor (vector double, vector double);
vector double vec_xor (vector double, vector bool long);
vector double vec_xor (vector bool long, vector double);
int vec_all_eq (vector double, vector double);
int vec_all_ge (vector double, vector double);
int vec_all_gt (vector double, vector double);
int vec_all_le (vector double, vector double);
int vec_all_lt (vector double, vector double);
int vec_all_nan (vector double);
int vec_all_ne (vector double, vector double);
int vec_all_nge (vector double, vector double);
int vec_all_ngt (vector double, vector double);
int vec_all_nle (vector double, vector double);
int vec_all_nlt (vector double, vector double);
int vec_all_numeric (vector double);
int vec_any_eq (vector double, vector double);
int vec_any_ge (vector double, vector double);
int vec_any_gt (vector double, vector double);
int vec_any_le (vector double, vector double);
int vec_any_lt (vector double, vector double);
int vec_any_nan (vector double);
int vec_any_ne (vector double, vector double);
int vec_any_nge (vector double, vector double);
int vec_any_ngt (vector double, vector double);
int vec_any_nle (vector double, vector double);
int vec_any_nlt (vector double, vector double);
int vec_any_numeric (vector double);

vector double vec_vsx_ld (int, const vector double *);
vector double vec_vsx_ld (int, const double *);
vector float vec_vsx_ld (int, const vector float *);
vector float vec_vsx_ld (int, const float *);
vector bool int vec_vsx_ld (int, const vector bool int *);
vector signed int vec_vsx_ld (int, const vector signed int *);
vector signed int vec_vsx_ld (int, const int *);
vector signed int vec_vsx_ld (int, const long *);
vector unsigned int vec_vsx_ld (int, const vector unsigned int *);
vector unsigned int vec_vsx_ld (int, const unsigned int *);
vector unsigned int vec_vsx_ld (int, const unsigned long *);
vector bool short vec_vsx_ld (int, const vector bool short *);
vector pixel vec_vsx_ld (int, const vector pixel *);
vector signed short vec_vsx_ld (int, const vector signed short *);
vector signed short vec_vsx_ld (int, const short *);
vector unsigned short vec_vsx_ld (int, const vector unsigned short *);
vector unsigned short vec_vsx_ld (int, const unsigned short *);
vector bool char vec_vsx_ld (int, const vector bool char *);
vector signed char vec_vsx_ld (int, const vector signed char *);
vector signed char vec_vsx_ld (int, const signed char *);
vector unsigned char vec_vsx_ld (int, const vector unsigned char *);
vector unsigned char vec_vsx_ld (int, const unsigned char *);

void vec_vsx_st (vector double, int, vector double *);
void vec_vsx_st (vector double, int, double *);
void vec_vsx_st (vector float, int, vector float *);
```

```
void vec_vsx_st (vector float, int, float *);
void vec_vsx_st (vector signed int, int, vector signed int *);
void vec_vsx_st (vector signed int, int, int *);
void vec_vsx_st (vector unsigned int, int, vector unsigned int *);
void vec_vsx_st (vector unsigned int, int, unsigned int *);
void vec_vsx_st (vector bool int, int, vector bool int *);
void vec_vsx_st (vector bool int, int, unsigned int *);
void vec_vsx_st (vector bool int, int, int *);
void vec_vsx_st (vector signed short, int, vector signed short *);
void vec_vsx_st (vector signed short, int, short *);
void vec_vsx_st (vector unsigned short, int, vector unsigned short *);
void vec_vsx_st (vector unsigned short, int, unsigned short *);
void vec_vsx_st (vector bool short, int, vector bool short *);
void vec_vsx_st (vector bool short, int, unsigned short *);
void vec_vsx_st (vector pixel, int, vector pixel *);
void vec_vsx_st (vector pixel, int, unsigned short *);
void vec_vsx_st (vector pixel, int, short *);
void vec_vsx_st (vector bool short, int, short *);
void vec_vsx_st (vector signed char, int, vector signed char *);
void vec_vsx_st (vector signed char, int, signed char *);
void vec_vsx_st (vector unsigned char, int, vector unsigned char *);
void vec_vsx_st (vector unsigned char, int, unsigned char *);
void vec_vsx_st (vector bool char, int, vector bool char *);
void vec_vsx_st (vector bool char, int, unsigned char *);
void vec_vsx_st (vector bool char, int, signed char *);
```

Note that the '`vec_ld`' and '`vec_st`' builtins will always generate the Altivec '`LVX`' and '`STVX`' instructions even if the VSX instruction set is available. The '`vec_vsx_ld`' and '`vec_vsx_st`' builtins will always generate the VSX '`LXVD2X`', '`LXVW4X`', '`STXVD2X`', and '`STXVW4X`' instructions.

GCC provides a few other builtins on Powerpc to access certain instructions:

```
float __builtin_recipdivf (float, float);
float __builtin_rsqrtf (float);
double __builtin_recipdiv (double, double);
double __builtin_rsqrt (double);
long __builtin_bpermd (long, long);
int __builtin_bswap16 (int);
```

The `vec_rsqrt`, `__builtin_rsqrt`, and `__builtin_rsqrtf` functions generate multiple instructions to implement the reciprocal sqrt functionality using reciprocal sqrt estimate instructions.

The `__builtin_recipdiv`, and `__builtin_recipdivf` functions generate multiple instructions to implement division using the reciprocal estimate instructions.

6.55.14 RX Built-in Functions

GCC supports some of the RX instructions which cannot be expressed in the C programming language via the use of built-in functions. The following functions are supported:

`void __builtin_rx_brk (`*void*`)` [Built-in Function]
Generates the `brk` machine instruction.

`void __builtin_rx_clrpsw (`*int*`)` [Built-in Function]
Generates the `clrpsw` machine instruction to clear the specified bit in the processor status word.

`void __builtin_rx_int (`*int*`)` [Built-in Function]
Generates the `int` machine instruction to generate an interrupt with the specified value.

`void __builtin_rx_machi (`*int, int*`)` [Built-in Function]
Generates the `machi` machine instruction to add the result of multiplying the top 16-bits of the two arguments into the accumulator.

`void __builtin_rx_maclo (`*int, int*`)` [Built-in Function]
Generates the `maclo` machine instruction to add the result of multiplying the bottom 16-bits of the two arguments into the accumulator.

`void __builtin_rx_mulhi (`*int, int*`)` [Built-in Function]
Generates the `mulhi` machine instruction to place the result of multiplying the top 16-bits of the two arguments into the accumulator.

`void __builtin_rx_mullo (`*int, int*`)` [Built-in Function]
Generates the `mullo` machine instruction to place the result of multiplying the bottom 16-bits of the two arguments into the accumulator.

`int __builtin_rx_mvfachi (`*void*`)` [Built-in Function]
Generates the `mvfachi` machine instruction to read the top 32-bits of the accumulator.

`int __builtin_rx_mvfacmi (`*void*`)` [Built-in Function]
Generates the `mvfacmi` machine instruction to read the middle 32-bits of the accumulator.

`int __builtin_rx_mvfc (`*int*`)` [Built-in Function]
Generates the `mvfc` machine instruction which reads the control register specified in its argument and returns its value.

`void __builtin_rx_mvtachi (`*int*`)` [Built-in Function]
Generates the `mvtachi` machine instruction to set the top 32-bits of the accumulator.

`void __builtin_rx_mvtaclo (`*int*`)` [Built-in Function]
Generates the `mvtaclo` machine instruction to set the bottom 32-bits of the accumulator.

`void __builtin_rx_mvtc (`*int reg, int val*`)` [Built-in Function]
Generates the `mvtc` machine instruction which sets control register number `reg` to `val`.

`void __builtin_rx_mvtipl (`*int*`)` [Built-in Function]
Generates the `mvtipl` machine instruction set the interrupt priority level.

`void __builtin_rx_racw (`*int*`)` [Built-in Function]
Generates the `racw` machine instruction to round the accumulator according to the specified mode.

`int __builtin_rx_revw (`*int*`)` [Built-in Function]
Generates the `revw` machine instruction which swaps the bytes in the argument so that bits 0–7 now occupy bits 8–15 and vice versa, and also bits 16–23 occupy bits 24–31 and vice versa.

`void __builtin_rx_rmpa (`*void*`)` [Built-in Function]
Generates the `rmpa` machine instruction which initiates a repeated multiply and accumulate sequence.

`void __builtin_rx_round (`*float*`)` [Built-in Function]
Generates the `round` machine instruction which returns the floating point argument rounded according to the current rounding mode set in the floating point status word register.

`int __builtin_rx_sat (`*int*`)` [Built-in Function]
Generates the `sat` machine instruction which returns the saturated value of the argument.

`void __builtin_rx_setpsw (`*int*`)` [Built-in Function]
Generates the `setpsw` machine instruction to set the specified bit in the processor status word.

`void __builtin_rx_wait (`*void*`)` [Built-in Function]
Generates the `wait` machine instruction.

6.55.15 SPARC VIS Built-in Functions

GCC supports SIMD operations on the SPARC using both the generic vector extensions (see Section 6.49 [Vector Extensions], page 423) as well as built-in functions for the SPARC Visual Instruction Set (VIS). When you use the '`-mvis`' switch, the VIS extension is exposed as the following built-in functions:

```
typedef int v1si __attribute__ ((vector_size (4)));
typedef int v2si __attribute__ ((vector_size (8)));
typedef short v4hi __attribute__ ((vector_size (8)));
typedef short v2hi __attribute__ ((vector_size (4)));
typedef unsigned char v8qi __attribute__ ((vector_size (8)));
typedef unsigned char v4qi __attribute__ ((vector_size (4)));

void __builtin_vis_write_gsr (int64_t);
int64_t __builtin_vis_read_gsr (void);

void * __builtin_vis_alignaddr (void *, long);
void * __builtin_vis_alignaddrl (void *, long);
int64_t __builtin_vis_faligndatadi (int64_t, int64_t);
v2si __builtin_vis_faligndatav2si (v2si, v2si);
v4hi __builtin_vis_faligndatav4hi (v4si, v4si);
v8qi __builtin_vis_faligndatav8qi (v8qi, v8qi);

v4hi __builtin_vis_fexpand (v4qi);

v4hi __builtin_vis_fmul8x16 (v4qi, v4hi);
v4hi __builtin_vis_fmul8x16au (v4qi, v2hi);
v4hi __builtin_vis_fmul8x16al (v4qi, v2hi);
v4hi __builtin_vis_fmul8sux16 (v8qi, v4hi);
v4hi __builtin_vis_fmul8ulx16 (v8qi, v4hi);
v2si __builtin_vis_fmuld8sux16 (v4qi, v2hi);
v2si __builtin_vis_fmuld8ulx16 (v4qi, v2hi);

v4qi __builtin_vis_fpack16 (v4hi);
v8qi __builtin_vis_fpack32 (v2si, v8qi);
```

```
v2hi __builtin_vis_fpackfix (v2si);
v8qi __builtin_vis_fpmerge (v4qi, v4qi);

int64_t __builtin_vis_pdist (v8qi, v8qi, int64_t);

long __builtin_vis_edge8 (void *, void *);
long __builtin_vis_edge8l (void *, void *);
long __builtin_vis_edge16 (void *, void *);
long __builtin_vis_edge16l (void *, void *);
long __builtin_vis_edge32 (void *, void *);
long __builtin_vis_edge32l (void *, void *);

long __builtin_vis_fcmple16 (v4hi, v4hi);
long __builtin_vis_fcmple32 (v2si, v2si);
long __builtin_vis_fcmpne16 (v4hi, v4hi);
long __builtin_vis_fcmpne32 (v2si, v2si);
long __builtin_vis_fcmpgt16 (v4hi, v4hi);
long __builtin_vis_fcmpgt32 (v2si, v2si);
long __builtin_vis_fcmpeq16 (v4hi, v4hi);
long __builtin_vis_fcmpeq32 (v2si, v2si);

v4hi __builtin_vis_fpadd16 (v4hi, v4hi);
v2hi __builtin_vis_fpadd16s (v2hi, v2hi);
v2si __builtin_vis_fpadd32 (v2si, v2si);
v1si __builtin_vis_fpadd32s (v1si, v1si);
v4hi __builtin_vis_fpsub16 (v4hi, v4hi);
v2hi __builtin_vis_fpsub16s (v2hi, v2hi);
v2si __builtin_vis_fpsub32 (v2si, v2si);
v1si __builtin_vis_fpsub32s (v1si, v1si);

long __builtin_vis_array8 (long, long);
long __builtin_vis_array16 (long, long);
long __builtin_vis_array32 (long, long);
```

When you use the '`-mvis2`' switch, the VIS version 2.0 built-in functions also become available:

```
long __builtin_vis_bmask (long, long);
int64_t __builtin_vis_bshuffledi (int64_t, int64_t);
v2si __builtin_vis_bshufflev2si (v2si, v2si);
v4hi __builtin_vis_bshufflev2si (v4hi, v4hi);
v8qi __builtin_vis_bshufflev2si (v8qi, v8qi);

long __builtin_vis_edge8n (void *, void *);
long __builtin_vis_edge8ln (void *, void *);
long __builtin_vis_edge16n (void *, void *);
long __builtin_vis_edge16ln (void *, void *);
long __builtin_vis_edge32n (void *, void *);
long __builtin_vis_edge32ln (void *, void *);
```

When you use the '`-mvis3`' switch, the VIS version 3.0 built-in functions also become available:

```
void __builtin_vis_cmask8 (long);
void __builtin_vis_cmask16 (long);
void __builtin_vis_cmask32 (long);

v4hi __builtin_vis_fchksm16 (v4hi, v4hi);

v4hi __builtin_vis_fsll16 (v4hi, v4hi);
```

```
v4hi __builtin_vis_fslas16 (v4hi, v4hi);
v4hi __builtin_vis_fsrl16 (v4hi, v4hi);
v4hi __builtin_vis_fsra16 (v4hi, v4hi);
v2si __builtin_vis_fsll16 (v2si, v2si);
v2si __builtin_vis_fslas16 (v2si, v2si);
v2si __builtin_vis_fsrl16 (v2si, v2si);
v2si __builtin_vis_fsra16 (v2si, v2si);

long __builtin_vis_pdistn (v8qi, v8qi);

v4hi __builtin_vis_fmean16 (v4hi, v4hi);

int64_t __builtin_vis_fpadd64 (int64_t, int64_t);
int64_t __builtin_vis_fpsub64 (int64_t, int64_t);

v4hi __builtin_vis_fpadds16 (v4hi, v4hi);
v2hi __builtin_vis_fpadds16s (v2hi, v2hi);
v4hi __builtin_vis_fpsubs16 (v4hi, v4hi);
v2hi __builtin_vis_fpsubs16s (v2hi, v2hi);
v2si __builtin_vis_fpadds32 (v2si, v2si);
v1si __builtin_vis_fpadds32s (v1si, v1si);
v2si __builtin_vis_fpsubs32 (v2si, v2si);
v1si __builtin_vis_fpsubs32s (v1si, v1si);

long __builtin_vis_fucmple8 (v8qi, v8qi);
long __builtin_vis_fucmpne8 (v8qi, v8qi);
long __builtin_vis_fucmpgt8 (v8qi, v8qi);
long __builtin_vis_fucmpeq8 (v8qi, v8qi);

float __builtin_vis_fhadds (float, float);
double __builtin_vis_fhaddd (double, double);
float __builtin_vis_fhsubs (float, float);
double __builtin_vis_fhsubd (double, double);
float __builtin_vis_fnhadds (float, float);
double __builtin_vis_fnhaddd (double, double);

int64_t __builtin_vis_umulxhi (int64_t, int64_t);
int64_t __builtin_vis_xmulx (int64_t, int64_t);
int64_t __builtin_vis_xmulxhi (int64_t, int64_t);
```

6.55.16 SPU Built-in Functions

GCC provides extensions for the SPU processor as described in the Sony/Toshiba/IBM SPU Language Extensions Specification, which can be found at `http://cell.scei.co.jp/` or `http://www.ibm.com/developerworks/power/cell/`. GCC's implementation differs in several ways.

- The optional extension of specifying vector constants in parentheses is not supported.
- A vector initializer requires no cast if the vector constant is of the same type as the variable it is initializing.
- If `signed` or `unsigned` is omitted, the signedness of the vector type is the default signedness of the base type. The default varies depending on the operating system, so a portable program should always specify the signedness.
- By default, the keyword `__vector` is added. The macro `vector` is defined in `<spu_intrinsics.h>` and can be undefined.
- GCC allows using a `typedef` name as the type specifier for a vector type.

- For C, overloaded functions are implemented with macros so the following does not work:

  ```
  spu_add ((vector signed int){1, 2, 3, 4}, foo);
  ```

  Since `spu_add` is a macro, the vector constant in the example is treated as four separate arguments. Wrap the entire argument in parentheses for this to work.
- The extended version of `__builtin_expect` is not supported.

Note: Only the interface described in the aforementioned specification is supported. Internally, GCC uses built-in functions to implement the required functionality, but these are not supported and are subject to change without notice.

6.55.17 TI C6X Built-in Functions

GCC provides intrinsics to access certain instructions of the TI C6X processors. These intrinsics, listed below, are available after inclusion of the `c6x_intrinsics.h` header file. They map directly to C6X instructions.

```
int _sadd (int, int)
int _ssub (int, int)
int _sadd2 (int, int)
int _ssub2 (int, int)
long long _mpy2 (int, int)
long long _smpy2 (int, int)
int _add4 (int, int)
int _sub4 (int, int)
int _saddu4 (int, int)

int _smpy (int, int)
int _smpyh (int, int)
int _smpyhl (int, int)
int _smpylh (int, int)

int _sshl (int, int)
int _subc (int, int)

int _avg2 (int, int)
int _avgu4 (int, int)

int _clrr (int, int)
int _extr (int, int)
int _extru (int, int)
int _abs (int)
int _abs2 (int)
```

6.55.18 TILE-Gx Built-in Functions

GCC provides intrinsics to access every instruction of the TILE-Gx processor. The intrinsics are of the form:

```
unsigned long long __insn_op (...)
```

Where *op* is the name of the instruction. Refer to the ISA manual for the complete list of instructions.

GCC also provides intrinsics to directly access the network registers. The intrinsics are:

```
unsigned long long __tile_idn0_receive (void)
unsigned long long __tile_idn1_receive (void)
unsigned long long __tile_udn0_receive (void)
unsigned long long __tile_udn1_receive (void)
unsigned long long __tile_udn2_receive (void)
unsigned long long __tile_udn3_receive (void)
void __tile_idn_send (unsigned long long)
void __tile_udn_send (unsigned long long)
```

The intrinsic `void __tile_network_barrier (void)` is used to guarantee that no network operatons before it will be reordered with those after it.

6.55.19 TILEPro Built-in Functions

GCC provides intrinsics to access every instruction of the TILEPro processor. The intrinsics are of the form:

```
unsigned __insn_op (...)
```

Where *op* is the name of the instruction. Refer to the ISA manual for the complete list of instructions.

GCC also provides intrinsics to directly access the network registers. The intrinsics are:

```
unsigned __tile_idn0_receive (void)
unsigned __tile_idn1_receive (void)
unsigned __tile_sn_receive (void)
unsigned __tile_udn0_receive (void)
unsigned __tile_udn1_receive (void)
unsigned __tile_udn2_receive (void)
unsigned __tile_udn3_receive (void)
void __tile_idn_send (unsigned)
void __tile_sn_send (unsigned)
void __tile_udn_send (unsigned)
```

The intrinsic `void __tile_network_barrier (void)` is used to guarantee that no network operatons before it will be reordered with those after it.

6.56 Format Checks Specific to Particular Target Machines

For some target machines, GCC supports additional options to the format attribute (see Section 6.30 [Declaring Attributes of Functions], page 332).

6.56.1 Solaris Format Checks

Solaris targets support the `cmn_err` (or `__cmn_err__`) format check. `cmn_err` accepts a subset of the standard `printf` conversions, and the two-argument `%b` conversion for displaying bit-fields. See the Solaris man page for `cmn_err` for more information.

6.56.2 Darwin Format Checks

Darwin targets support the `CFString` (or `__CFString__`) in the format attribute context. Declarations made with such attribution will be parsed for correct syntax and format ar-

gument types. However, parsing of the format string itself is currently undefined and will not be carried out by this version of the compiler.

Additionally, `CFStringRefs` (defined by the `CoreFoundation` headers) may also be used as format arguments. Note that the relevant headers are only likely to be available on Darwin (OSX) installations. On such installations, the XCode and system documentation provide descriptions of `CFString`, `CFStringRefs` and associated functions.

6.57 Pragmas Accepted by GCC

GCC supports several types of pragmas, primarily in order to compile code originally written for other compilers. Note that in general we do not recommend the use of pragmas; See Section 6.30 [Function Attributes], page 332, for further explanation.

6.57.1 ARM Pragmas

The ARM target defines pragmas for controlling the default addition of `long_call` and `short_call` attributes to functions. See Section 6.30 [Function Attributes], page 332, for information about the effects of these attributes.

`long_calls`
: Set all subsequent functions to have the `long_call` attribute.

`no_long_calls`
: Set all subsequent functions to have the `short_call` attribute.

`long_calls_off`
: Do not affect the `long_call` or `short_call` attributes of subsequent functions.

6.57.2 M32C Pragmas

`GCC memregs` *`number`*
: Overrides the command-line option `-memregs=` for the current file. Use with care! This pragma must be before any function in the file, and mixing different memregs values in different objects may make them incompatible. This pragma is useful when a performance-critical function uses a memreg for temporary values, as it may allow you to reduce the number of memregs used.

`ADDRESS` *`name address`*
: For any declared symbols matching *name*, this does three things to that symbol: it forces the symbol to be located at the given address (a number), it forces the symbol to be volatile, and it changes the symbol's scope to be static. This pragma exists for compatibility with other compilers, but note that the common `1234H` numeric syntax is not supported (use `0x1234` instead). Example:

```
#pragma ADDRESS port3 0x103
char port3;
```

6.57.3 MeP Pragmas

`custom io_volatile (on|off)`
: Overrides the command line option `-mio-volatile` for the current file. Note that for compatibility with future GCC releases, this option should only be used once before any `io` variables in each file.

`GCC coprocessor available` *registers*

Specifies which coprocessor registers are available to the register allocator. *registers* may be a single register, register range separated by ellipses, or comma-separated list of those. Example:

```
#pragma GCC coprocessor available $c0...$c10, $c28
```

`GCC coprocessor call_saved` *registers*

Specifies which coprocessor registers are to be saved and restored by any function using them. *registers* may be a single register, register range separated by ellipses, or comma-separated list of those. Example:

```
#pragma GCC coprocessor call_saved $c4...$c6, $c31
```

`GCC coprocessor subclass '(A|B|C|D)' =` *registers*

Creates and defines a register class. These register classes can be used by inline `asm` constructs. *registers* may be a single register, register range separated by ellipses, or comma-separated list of those. Example:

```
#pragma GCC coprocessor subclass 'B' = $c2, $c4, $c6

asm ("cpfoo %0" : "=B" (x));
```

`GCC disinterrupt` *name* `,` *name* `...`

For the named functions, the compiler adds code to disable interrupts for the duration of those functions. Any functions so named, which are not encountered in the source, cause a warning that the pragma was not used. Examples:

```
#pragma disinterrupt foo
#pragma disinterrupt bar, grill
int foo () { ... }
```

`GCC call` *name* `,` *name* `...`

For the named functions, the compiler always uses a register-indirect call model when calling the named functions. Examples:

```
extern int foo ();
#pragma call foo
```

6.57.4 RS/6000 and PowerPC Pragmas

The RS/6000 and PowerPC targets define one pragma for controlling whether or not the `longcall` attribute is added to function declarations by default. This pragma overrides the '`-mlongcall`' option, but not the `longcall` and `shortcall` attributes. See Section 3.17.34 [RS/6000 and PowerPC Options], page 249, for more information about when long calls are and are not necessary.

`longcall (1)`

Apply the `longcall` attribute to all subsequent function declarations.

`longcall (0)`

Do not apply the `longcall` attribute to subsequent function declarations.

6.57.5 Darwin Pragmas

The following pragmas are available for all architectures running the Darwin operating system. These are useful for compatibility with other Mac OS compilers.

`mark` *tokens*`...`
: This pragma is accepted, but has no effect.

`options align=`*alignment*
: This pragma sets the alignment of fields in structures. The values of *alignment* may be `mac68k`, to emulate m68k alignment, or `power`, to emulate PowerPC alignment. Uses of this pragma nest properly; to restore the previous setting, use `reset` for the *alignment*.

`segment` *tokens*`...`
: This pragma is accepted, but has no effect.

`unused (`*var* `[, `*var*`]...)`
: This pragma declares variables to be possibly unused. GCC will not produce warnings for the listed variables. The effect is similar to that of the `unused` attribute, except that this pragma may appear anywhere within the variables' scopes.

6.57.6 Solaris Pragmas

The Solaris target supports `#pragma redefine_extname` (see Section 6.57.7 [Symbol-Renaming Pragmas], page 612). It also supports additional `#pragma` directives for compatibility with the system compiler.

`align` *alignment* `(`*variable* `[, `*variable*`]...)`
: Increase the minimum alignment of each *variable* to *alignment*. This is the same as GCC's `aligned` attribute see Section 6.36 [Variable Attributes], page 365). Macro expansion occurs on the arguments to this pragma when compiling C and Objective-C. It does not currently occur when compiling C++, but this is a bug which may be fixed in a future release.

`fini (`*function* `[, `*function*`]...)`
: This pragma causes each listed *function* to be called after main, or during shared module unloading, by adding a call to the `.fini` section.

`init (`*function* `[, `*function*`]...)`
: This pragma causes each listed *function* to be called during initialization (before `main`) or during shared module loading, by adding a call to the `.init` section.

6.57.7 Symbol-Renaming Pragmas

For compatibility with the Solaris and Tru64 UNIX system headers, GCC supports two `#pragma` directives which change the name used in assembly for a given declaration. `#pragma extern_prefix` is only available on platforms whose system headers need it. To get this effect on all platforms supported by GCC, use the asm labels extension (see Section 6.43 [Asm Labels], page 417).

`redefine_extname` *oldname* *newname*
: This pragma gives the C function *oldname* the assembly symbol *newname*. The preprocessor macro `__PRAGMA_REDEFINE_EXTNAME` will be defined if this pragma is available (currently on all platforms).

`extern_prefix` *string*

This pragma causes all subsequent external function and variable declarations to have *string* prepended to their assembly symbols. This effect may be terminated with another `extern_prefix` pragma whose argument is an empty string. The preprocessor macro `__PRAGMA_EXTERN_PREFIX` will be defined if this pragma is available (currently only on Tru64 UNIX).

These pragmas and the asm labels extension interact in a complicated manner. Here are some corner cases you may want to be aware of.

1. Both pragmas silently apply only to declarations with external linkage. Asm labels do not have this restriction.
2. In C++, both pragmas silently apply only to declarations with "C" linkage. Again, asm labels do not have this restriction.
3. If any of the three ways of changing the assembly name of a declaration is applied to a declaration whose assembly name has already been determined (either by a previous use of one of these features, or because the compiler needed the assembly name in order to generate code), and the new name is different, a warning issues and the name does not change.
4. The *oldname* used by `#pragma redefine_extname` is always the C-language name.
5. If `#pragma extern_prefix` is in effect, and a declaration occurs with an asm label attached, the prefix is silently ignored for that declaration.
6. If `#pragma extern_prefix` and `#pragma redefine_extname` apply to the same declaration, whichever triggered first wins, and a warning issues if they contradict each other. (We would like to have `#pragma redefine_extname` always win, for consistency with asm labels, but if `#pragma extern_prefix` triggers first we have no way of knowing that that happened.)

6.57.8 Structure-Packing Pragmas

For compatibility with Microsoft Windows compilers, GCC supports a set of `#pragma` directives which change the maximum alignment of members of structures (other than zero-width bitfields), unions, and classes subsequently defined. The *n* value below always is required to be a small power of two and specifies the new alignment in bytes.

1. `#pragma pack(n)` simply sets the new alignment.
2. `#pragma pack()` sets the alignment to the one that was in effect when compilation started (see also command-line option '`-fpack-struct[=n]`' see Section 3.18 [Code Gen Options], page 285).
3. `#pragma pack(push[,n])` pushes the current alignment setting on an internal stack and then optionally sets the new alignment.
4. `#pragma pack(pop)` restores the alignment setting to the one saved at the top of the internal stack (and removes that stack entry). Note that `#pragma pack([n])` does not influence this internal stack; thus it is possible to have `#pragma pack(push)` followed by multiple `#pragma pack(n)` instances and finalized by a single `#pragma pack(pop)`.

Some targets, e.g. i386 and powerpc, support the `ms_struct #pragma` which lays out a structure as the documented `__attribute__ ((ms_struct))`.

1. `#pragma ms_struct on` turns on the layout for structures declared.
2. `#pragma ms_struct off` turns off the layout for structures declared.
3. `#pragma ms_struct reset` goes back to the default layout.

6.57.9 Weak Pragmas

For compatibility with SVR4, GCC supports a set of `#pragma` directives for declaring symbols to be weak, and defining weak aliases.

`#pragma weak` *`symbol`*

This pragma declares *symbol* to be weak, as if the declaration had the attribute of the same name. The pragma may appear before or after the declaration of *symbol*. It is not an error for *symbol* to never be defined at all.

`#pragma weak` *`symbol1 = symbol2`*

This pragma declares *symbol1* to be a weak alias of *symbol2*. It is an error if *symbol2* is not defined in the current translation unit.

6.57.10 Diagnostic Pragmas

GCC allows the user to selectively enable or disable certain types of diagnostics, and change the kind of the diagnostic. For example, a project's policy might require that all sources compile with '`-Werror`' but certain files might have exceptions allowing specific types of warnings. Or, a project might selectively enable diagnostics and treat them as errors depending on which preprocessor macros are defined.

`#pragma GCC diagnostic` *`kind option`*

Modifies the disposition of a diagnostic. Note that not all diagnostics are modifiable; at the moment only warnings (normally controlled by '`-W...`') can be controlled, and not all of them. Use '`-fdiagnostics-show-option`' to determine which diagnostics are controllable and which option controls them.

kind is '`error`' to treat this diagnostic as an error, '`warning`' to treat it like a warning (even if '`-Werror`' is in effect), or '`ignored`' if the diagnostic is to be ignored. *option* is a double quoted string which matches the command-line option.

```
#pragma GCC diagnostic warning "-Wformat"
#pragma GCC diagnostic error "-Wformat"
#pragma GCC diagnostic ignored "-Wformat"
```

Note that these pragmas override any command-line options. GCC keeps track of the location of each pragma, and issues diagnostics according to the state as of that point in the source file. Thus, pragmas occurring after a line do not affect diagnostics caused by that line.

`#pragma GCC diagnostic push`
`#pragma GCC diagnostic pop`

Causes GCC to remember the state of the diagnostics as of each `push`, and restore to that point at each `pop`. If a `pop` has no matching `push`, the command line options are restored.

```
#pragma GCC diagnostic error "-Wuninitialized"
```

```
  foo(a); /* error is given for this one */
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wuninitialized"
  foo(b); /* no diagnostic for this one */
#pragma GCC diagnostic pop
  foo(c); /* error is given for this one */
#pragma GCC diagnostic pop
  foo(d); /* depends on command line options */
```

GCC also offers a simple mechanism for printing messages during compilation.

`#pragma message` *`string`*

Prints *string* as a compiler message on compilation. The message is informational only, and is neither a compilation warning nor an error.

```
#pragma message "Compiling " __FILE__ "..."
```

string may be parenthesized, and is printed with location information. For example,

```
#define DO_PRAGMA(x) _Pragma (#x)
#define TODO(x) DO_PRAGMA(message ("TODO - " #x))

TODO(Remember to fix this)
```

prints '`/tmp/file.c:4: note: #pragma message: TODO - Remember to fix this`'.

6.57.11 Visibility Pragmas

`#pragma GCC visibility push(`*`visibility`*`)`
`#pragma GCC visibility pop`

This pragma allows the user to set the visibility for multiple declarations without having to give each a visibility attribute See Section 6.30 [Function Attributes], page 332, for more information about visibility and the attribute syntax.

In C++, '`#pragma GCC visibility`' affects only namespace-scope declarations. Class members and template specializations are not affected; if you want to override the visibility for a particular member or instantiation, you must use an attribute.

6.57.12 Push/Pop Macro Pragmas

For compatibility with Microsoft Windows compilers, GCC supports '`#pragma push_macro("`*`macro_name`*`")`' and '`#pragma pop_macro("`*`macro_name`*`")`'.

`#pragma push_macro("`*`macro_name`*`")`

This pragma saves the value of the macro named as *macro_name* to the top of the stack for this macro.

`#pragma pop_macro("`*`macro_name`*`")`

This pragma sets the value of the macro named as *macro_name* to the value on top of the stack for this macro. If the stack for *macro_name* is empty, the value of the macro remains unchanged.

For example:

```
#define X  1
#pragma push_macro("X")
#undef X
#define X -1
#pragma pop_macro("X")
int x [X];
```

In this example, the definition of X as 1 is saved by `#pragma push_macro` and restored by `#pragma pop_macro`.

6.57.13 Function Specific Option Pragmas

`#pragma GCC target ("`*string*`"...)`

This pragma allows you to set target specific options for functions defined later in the source file. One or more strings can be specified. Each function that is defined after this point will be as if `attribute((target("STRING")))` was specified for that function. The parenthesis around the options is optional. See Section 6.30 [Function Attributes], page 332, for more information about the `target` attribute and the attribute syntax.

The `#pragma GCC target` attribute is not implemented in GCC versions earlier than 4.4 for the i386/x86_64 and 4.6 for the PowerPC backends. At present, it is not implemented for other backends.

`#pragma GCC optimize ("`*string*`"...)`

This pragma allows you to set global optimization options for functions defined later in the source file. One or more strings can be specified. Each function that is defined after this point will be as if `attribute((optimize("STRING")))` was specified for that function. The parenthesis around the options is optional. See Section 6.30 [Function Attributes], page 332, for more information about the `optimize` attribute and the attribute syntax.

The '`#pragma GCC optimize`' pragma is not implemented in GCC versions earlier than 4.4.

`#pragma GCC push_options`
`#pragma GCC pop_options`

These pragmas maintain a stack of the current target and optimization options. It is intended for include files where you temporarily want to switch to using a different '`#pragma GCC target`' or '`#pragma GCC optimize`' and then to pop back to the previous options.

The '`#pragma GCC push_options`' and '`#pragma GCC pop_options`' pragmas are not implemented in GCC versions earlier than 4.4.

`#pragma GCC reset_options`

This pragma clears the current `#pragma GCC target` and `#pragma GCC optimize` to use the default switches as specified on the command line.

The '`#pragma GCC reset_options`' pragma is not implemented in GCC versions earlier than 4.4.

6.58 Unnamed struct/union fields within structs/unions

As permitted by ISO C11 and for compatibility with other compilers, GCC allows you to define a structure or union that contains, as fields, structures and unions without names. For example:

```
struct {
  int a;
  union {
    int b;
    float c;
  };
  int d;
} foo;
```

In this example, the user would be able to access members of the unnamed union with code like '`foo.b`'. Note that only unnamed structs and unions are allowed, you may not have, for example, an unnamed `int`.

You must never create such structures that cause ambiguous field definitions. For example, this structure:

```
struct {
  int a;
  struct {
    int a;
  };
} foo;
```

It is ambiguous which `a` is being referred to with '`foo.a`'. The compiler gives errors for such constructs.

Unless '`-fms-extensions`' is used, the unnamed field must be a structure or union definition without a tag (for example, '`struct { int a; };`'). If '`-fms-extensions`' is used, the field may also be a definition with a tag such as '`struct foo { int a; };`', a reference to a previously defined structure or union such as '`struct foo;`', or a reference to a `typedef` name for a previously defined structure or union type.

The option '`-fplan9-extensions`' enables '`-fms-extensions`' as well as two other extensions. First, a pointer to a structure is automatically converted to a pointer to an anonymous field for assignments and function calls. For example:

```
struct s1 { int a; };
struct s2 { struct s1; };
extern void f1 (struct s1 *);
void f2 (struct s2 *p) { f1 (p); }
```

In the call to `f1` inside `f2`, the pointer `p` is converted into a pointer to the anonymous field.

Second, when the type of an anonymous field is a `typedef` for a `struct` or `union`, code may refer to the field using the name of the `typedef`.

```
typedef struct { int a; } s1;
struct s2 { s1; };
s1 f1 (struct s2 *p) { return p->s1; }
```

These usages are only permitted when they are not ambiguous.

6.59 Thread-Local Storage

Thread-local storage (TLS) is a mechanism by which variables are allocated such that there is one instance of the variable per extant thread. The run-time model GCC uses to implement this originates in the IA-64 processor-specific ABI, but has since been migrated to other processors as well. It requires significant support from the linker (`ld`), dynamic linker (`ld.so`), and system libraries ('`libc.so`' and '`libpthread.so`'), so it is not available everywhere.

At the user level, the extension is visible with a new storage class keyword: `__thread`. For example:

```
__thread int i;
extern __thread struct state s;
static __thread char *p;
```

The `__thread` specifier may be used alone, with the `extern` or `static` specifiers, but with no other storage class specifier. When used with `extern` or `static`, `__thread` must appear immediately after the other storage class specifier.

The `__thread` specifier may be applied to any global, file-scoped static, function-scoped static, or static data member of a class. It may not be applied to block-scoped automatic or non-static data member.

When the address-of operator is applied to a thread-local variable, it is evaluated at run-time and returns the address of the current thread's instance of that variable. An address so obtained may be used by any thread. When a thread terminates, any pointers to thread-local variables in that thread become invalid.

No static initialization may refer to the address of a thread-local variable.

In C++, if an initializer is present for a thread-local variable, it must be a *constant-expression*, as defined in 5.19.2 of the ANSI/ISO C++ standard.

See ELF Handling For Thread-Local Storage (`http://www.akkadia.org/drepper/tls.pdf`) for a detailed explanation of the four thread-local storage addressing models, and how the run-time is expected to function.

6.59.1 ISO/IEC 9899:1999 Edits for Thread-Local Storage

The following are a set of changes to ISO/IEC 9899:1999 (aka C99) that document the exact semantics of the language extension.

- *5.1.2 Execution environments*

  Add new text after paragraph 1

  > Within either execution environment, a *thread* is a flow of control within a program. It is implementation defined whether or not there may be more than one thread associated with a program. It is implementation defined how threads beyond the first are created, the name and type of the function called at thread startup, and how threads may be terminated. However, objects with thread storage duration shall be initialized before thread startup.

- *6.2.4 Storage durations of objects*

  Add new text before paragraph 3

> An object whose identifier is declared with the storage-class specifier `__thread` has *thread storage duration*. Its lifetime is the entire execution of the thread, and its stored value is initialized only once, prior to thread startup.

- *6.4.1 Keywords*

  Add `__thread`.

- *6.7.1 Storage-class specifiers*

  Add `__thread` to the list of storage class specifiers in paragraph 1.

  Change paragraph 2 to

  > With the exception of `__thread`, at most one storage-class specifier may be given [...]. The `__thread` specifier may be used alone, or immediately following `extern` or `static`.

  Add new text after paragraph 6

  > The declaration of an identifier for a variable that has block scope that specifies `__thread` shall also specify either `extern` or `static`.
  >
  > The `__thread` specifier shall be used only with variables.

6.59.2 ISO/IEC 14882:1998 Edits for Thread-Local Storage

The following are a set of changes to ISO/IEC 14882:1998 (aka C++98) that document the exact semantics of the language extension.

- **[intro.execution]**

  New text after paragraph 4

  > A *thread* is a flow of control within the abstract machine. It is implementation defined whether or not there may be more than one thread.

  New text after paragraph 7

  > It is unspecified whether additional action must be taken to ensure when and whether side effects are visible to other threads.

- **[lex.key]**

  Add `__thread`.

- **[basic.start.main]**

  Add after paragraph 5

  > The thread that begins execution at the `main` function is called the *main thread*. It is implementation defined how functions beginning threads other than the main thread are designated or typed. A function so designated, as well as the `main` function, is called a *thread startup function*. It is implementation defined what happens if a thread startup function returns. It is implementation defined what happens to other threads when any thread calls `exit`.

- **[basic.start.init]**

  Add after paragraph 4

  > The storage for an object of thread storage duration shall be statically initialized before the first statement of the thread startup function. An object of thread storage duration shall not require dynamic initialization.

- **[basic.start.term]**

  Add after paragraph 3

  > The type of an object with thread storage duration shall not have a non-trivial destructor, nor shall it be an array type whose elements (directly or indirectly) have non-trivial destructors.

- **[basic.stc]**

  Add "thread storage duration" to the list in paragraph 1.

  Change paragraph 2

  > Thread, static, and automatic storage durations are associated with objects introduced by declarations [. . .].

  Add `__thread` to the list of specifiers in paragraph 3.

- **[basic.stc.thread]**

  New section before **[basic.stc.static]**

  > The keyword `__thread` applied to a non-local object gives the object thread storage duration.
  >
  > A local variable or class data member declared both `static` and `__thread` gives the variable or member thread storage duration.

- **[basic.stc.static]**

  Change paragraph 1

  > All objects which have neither thread storage duration, dynamic storage duration nor are local [. . .].

- **[dcl.stc]**

  Add `__thread` to the list in paragraph 1.

  Change paragraph 1

  > With the exception of `__thread`, at most one *storage-class-specifier* shall appear in a given *decl-specifier-seq*. The `__thread` specifier may be used alone, or immediately following the `extern` or `static` specifiers. [. . .]

  Add after paragraph 5

  > The `__thread` specifier can be applied only to the names of objects and to anonymous unions.

- **[class.mem]**

  Add after paragraph 6

  > Non-`static` members shall not be `__thread`.

6.60 Binary constants using the '`0b`' prefix

Integer constants can be written as binary constants, consisting of a sequence of '`0`' and '`1`' digits, prefixed by '`0b`' or '`0B`'. This is particularly useful in environments that operate a lot on the bit-level (like microcontrollers).

The following statements are identical:

```
i =       42;
i =     0x2a;
```

```
i =      052;
i = 0b101010;
```

The type of these constants follows the same rules as for octal or hexadecimal integer constants, so suffixes like ‘`L`’ or ‘`UL`’ can be applied.

7 Extensions to the C++ Language

The GNU compiler provides these extensions to the C++ language (and you can also use most of the C language extensions in your C++ programs). If you want to write code that checks whether these features are available, you can test for the GNU compiler the same way as for C programs: check for a predefined macro `__GNUC__`. You can also use `__GNUG__` to test specifically for GNU C++ (see Section "Predefined Macros" in *The GNU C Preprocessor*).

7.1 When is a Volatile C++ Object Accessed?

The C++ standard differs from the C standard in its treatment of volatile objects. It fails to specify what constitutes a volatile access, except to say that C++ should behave in a similar manner to C with respect to volatiles, where possible. However, the different lvalueness of expressions between C and C++ complicate the behavior. G++ behaves the same as GCC for volatile access, See Chapter 6 [Volatiles], page 309, for a description of GCC's behavior.

The C and C++ language specifications differ when an object is accessed in a void context:

```
volatile int *src = somevalue;
*src;
```

The C++ standard specifies that such expressions do not undergo lvalue to rvalue conversion, and that the type of the dereferenced object may be incomplete. The C++ standard does not specify explicitly that it is lvalue to rvalue conversion which is responsible for causing an access. There is reason to believe that it is, because otherwise certain simple expressions become undefined. However, because it would surprise most programmers, G++ treats dereferencing a pointer to volatile object of complete type as GCC would do for an equivalent type in C. When the object has incomplete type, G++ issues a warning; if you wish to force an error, you must force a conversion to rvalue with, for instance, a static cast.

When using a reference to volatile, G++ does not treat equivalent expressions as accesses to volatiles, but instead issues a warning that no volatile is accessed. The rationale for this is that otherwise it becomes difficult to determine where volatile access occur, and not possible to ignore the return value from functions returning volatile references. Again, if you wish to force a read, cast the reference to an rvalue.

G++ implements the same behavior as GCC does when assigning to a volatile object – there is no reread of the assigned-to object, the assigned rvalue is reused. Note that in C++ assignment expressions are lvalues, and if used as an lvalue, the volatile object will be referred to. For instance, *vref* will refer to *vobj*, as expected, in the following example:

```
volatile int vobj;
volatile int &vref = vobj = something;
```

7.2 Restricting Pointer Aliasing

As with the C front end, G++ understands the C99 feature of restricted pointers, specified with the `__restrict__`, or `__restrict` type qualifier. Because you cannot compile C++ by specifying the '`-std=c99`' language flag, `restrict` is not a keyword in C++.

In addition to allowing restricted pointers, you can specify restricted references, which indicate that the reference is not aliased in the local context.

```
void fn (int *__restrict__ rptr, int &__restrict__ rref)
{
  /* ... */
}
```

In the body of `fn`, *rptr* points to an unaliased integer and *rref* refers to a (different) unaliased integer.

You may also specify whether a member function's *this* pointer is unaliased by using `__restrict__` as a member function qualifier.

```
void T::fn () __restrict__
{
  /* ... */
}
```

Within the body of `T::fn`, *this* will have the effective definition `T *__restrict__ const this`. Notice that the interpretation of a `__restrict__` member function qualifier is different to that of `const` or `volatile` qualifier, in that it is applied to the pointer rather than the object. This is consistent with other compilers which implement restricted pointers.

As with all outermost parameter qualifiers, `__restrict__` is ignored in function definition matching. This means you only need to specify `__restrict__` in a function definition, rather than in a function prototype as well.

7.3 Vague Linkage

There are several constructs in C++ which require space in the object file but are not clearly tied to a single translation unit. We say that these constructs have "vague linkage". Typically such constructs are emitted wherever they are needed, though sometimes we can be more clever.

Inline Functions

: Inline functions are typically defined in a header file which can be included in many different compilations. Hopefully they can usually be inlined, but sometimes an out-of-line copy is necessary, if the address of the function is taken or if inlining fails. In general, we emit an out-of-line copy in all translation units where one is needed. As an exception, we only emit inline virtual functions with the vtable, since it will always require a copy.

  Local static variables and string constants used in an inline function are also considered to have vague linkage, since they must be shared between all inlined and out-of-line instances of the function.

VTables

: C++ virtual functions are implemented in most compilers using a lookup table, known as a vtable. The vtable contains pointers to the virtual functions provided by a class, and each object of the class contains a pointer to its vtable (or vtables, in some multiple-inheritance situations). If the class declares any non-inline, non-pure virtual functions, the first one is chosen as the "key method" for the class, and the vtable is only emitted in the translation unit where the key method is defined.

  Note: If the chosen key method is later defined as inline, the vtable will still be emitted in every translation unit which defines it. Make sure that any inline virtuals are declared inline in the class body, even if they are not defined there.

`type_info` objects
: C++ requires information about types to be written out in order to implement '`dynamic_cast`', '`typeid`' and exception handling. For polymorphic classes (classes with virtual functions), the '`type_info`' object is written out along with the vtable so that '`dynamic_cast`' can determine the dynamic type of a class object at runtime. For all other types, we write out the '`type_info`' object when it is used: when applying '`typeid`' to an expression, throwing an object, or referring to a type in a catch clause or exception specification.

Template Instantiations
: Most everything in this section also applies to template instantiations, but there are other options as well. See Section 7.5 [Where's the Template?], page 626.

When used with GNU ld version 2.8 or later on an ELF system such as GNU/Linux or Solaris 2, or on Microsoft Windows, duplicate copies of these constructs will be discarded at link time. This is known as COMDAT support.

On targets that don't support COMDAT, but do support weak symbols, GCC will use them. This way one copy will override all the others, but the unused copies will still take up space in the executable.

For targets which do not support either COMDAT or weak symbols, most entities with vague linkage will be emitted as local symbols to avoid duplicate definition errors from the linker. This will not happen for local statics in inlines, however, as having multiple copies will almost certainly break things.

See Section 7.4 [Declarations and Definitions in One Header], page 625, for another way to control placement of these constructs.

7.4 #pragma interface and implementation

`#pragma interface` and `#pragma implementation` provide the user with a way of explicitly directing the compiler to emit entities with vague linkage (and debugging information) in a particular translation unit.

Note: As of GCC 2.7.2, these `#pragma`s are not useful in most cases, because of COMDAT support and the "key method" heuristic mentioned in Section 7.3 [Vague Linkage], page 624. Using them can actually cause your program to grow due to unnecessary out-of-line copies of inline functions. Currently (3.4) the only benefit of these `#pragma`s is reduced duplication of debugging information, and that should be addressed soon on DWARF 2 targets with the use of COMDAT groups.

`#pragma interface`
`#pragma interface "`*`subdir/objects`*`.h"`
: Use this directive in *header files* that define object classes, to save space in most of the object files that use those classes. Normally, local copies of certain information (backup copies of inline member functions, debugging information, and the internal tables that implement virtual functions) must be kept in each object file that includes class definitions. You can use this pragma to avoid such duplication. When a header file containing '`#pragma interface`' is included in a compilation, this auxiliary information will not be generated (unless the main

input source file itself uses '`#pragma implementation`'). Instead, the object files will contain references to be resolved at link time.

The second form of this directive is useful for the case where you have multiple headers with the same name in different directories. If you use this form, you must specify the same string to '`#pragma implementation`'.

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

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

If you use '`#pragma implementation`' with no argument, it applies to an include file with the same basename[1] as your source file. For example, in '`allclass.cc`', giving just '`#pragma implementation`' by itself is equivalent to '`#pragma implementation "allclass.h"`'.

In versions of GNU C++ prior to 2.6.0 '`allclass.h`' was treated as an implementation file whenever you would include it from '`allclass.cc`' even if you never specified '`#pragma implementation`'. This was deemed to be more trouble than it was worth, however, and disabled.

Use the string argument if you want a single implementation file to include code from multiple header files. (You must also use '`#include`' to include the header file; '`#pragma implementation`' only specifies how to use the file—it doesn't actually include it.)

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

'`#pragma implementation`' and '`#pragma interface`' also have an effect on function inlining.

If you define a class in a header file marked with '`#pragma interface`', the effect on an inline function defined in that class is similar to an explicit `extern` declaration—the compiler emits no code at all to define an independent version of the function. Its definition is used only for inlining with its callers.

Conversely, when you include the same header file in a main source file that declares it as '`#pragma implementation`', the compiler emits code for the function itself; this defines a version of the function that can be found via pointers (or by callers compiled without inlining). If all calls to the function can be inlined, you can avoid emitting the function by compiling with '`-fno-implement-inlines`'. If any calls were not inlined, you will get linker errors.

7.5 Where's the Template?

C++ templates are the first language feature to require more intelligence from the environment than one usually finds on a UNIX system. Somehow the compiler and linker have to

[1] A file's *basename* was the name stripped of all leading path information and of trailing suffixes, such as '`.h`' or '`.C`' or '`.cc`'.

make sure that each template instance occurs exactly once in the executable if it is needed, and not at all otherwise. There are two basic approaches to this problem, which are referred to as the Borland model and the Cfront model.

Borland model
: Borland C++ solved the template instantiation problem by adding the code equivalent of common blocks to their linker; the compiler emits template instances in each translation unit that uses them, and the linker collapses them together. The advantage of this model is that the linker only has to consider the object files themselves; there is no external complexity to worry about. This disadvantage is that compilation time is increased because the template code is being compiled repeatedly. Code written for this model tends to include definitions of all templates in the header file, since they must be seen to be instantiated.

Cfront model
: The AT&T C++ translator, Cfront, solved the template instantiation problem by creating the notion of a template repository, an automatically maintained place where template instances are stored. A more modern version of the repository works as follows: As individual object files are built, the compiler places any template definitions and instantiations encountered in the repository. At link time, the link wrapper adds in the objects in the repository and compiles any needed instances that were not previously emitted. The advantages of this model are more optimal compilation speed and the ability to use the system linker; to implement the Borland model a compiler vendor also needs to replace the linker. The disadvantages are vastly increased complexity, and thus potential for error; for some code this can be just as transparent, but in practice it can been very difficult to build multiple programs in one directory and one program in multiple directories. Code written for this model tends to separate definitions of non-inline member templates into a separate file, which should be compiled separately.

When used with GNU ld version 2.8 or later on an ELF system such as GNU/Linux or Solaris 2, or on Microsoft Windows, G++ supports the Borland model. On other systems, G++ implements neither automatic model.

A future version of G++ will support a hybrid model whereby the compiler will emit any instantiations for which the template definition is included in the compile, and store template definitions and instantiation context information into the object file for the rest. The link wrapper will extract that information as necessary and invoke the compiler to produce the remaining instantiations. The linker will then combine duplicate instantiations.

In the mean time, you have the following options for dealing with template instantiations:

1. Compile your template-using code with '`-frepo`'. The compiler will generate files with the extension '`.rpo`' listing all of the template instantiations used in the corresponding object files which could be instantiated there; the link wrapper, '`collect2`', will then update the '`.rpo`' files to tell the compiler where to place those instantiations and rebuild any affected object files. The link-time overhead is negligible after the first pass, as the compiler will continue to place the instantiations in the same files.

This is your best option for application code written for the Borland model, as it will just work. Code written for the Cfront model will need to be modified so that the template definitions are available at one or more points of instantiation; usually this is as simple as adding `#include <tmethods.cc>` to the end of each template header.

For library code, if you want the library to provide all of the template instantiations it needs, just try to link all of its object files together; the link will fail, but cause the instantiations to be generated as a side effect. Be warned, however, that this may cause conflicts if multiple libraries try to provide the same instantiations. For greater control, use explicit instantiation as described in the next option.

2. Compile your code with '`-fno-implicit-templates`' to disable the implicit generation of template instances, and explicitly instantiate all the ones you use. This approach requires more knowledge of exactly which instances you need than do the others, but it's less mysterious and allows greater control. You can scatter the explicit instantiations throughout your program, perhaps putting them in the translation units where the instances are used or the translation units that define the templates themselves; you can put all of the explicit instantiations you need into one big file; or you can create small files like

```
#include "Foo.h"
#include "Foo.cc"

template class Foo<int>;
template ostream& operator <<
                (ostream&, const Foo<int>&);
```

for each of the instances you need, and create a template instantiation library from those.

If you are using Cfront-model code, you can probably get away with not using '`-fno-implicit-templates`' when compiling files that don't '`#include`' the member template definitions.

If you use one big file to do the instantiations, you may want to compile it without '`-fno-implicit-templates`' so you get all of the instances required by your explicit instantiations (but not by any other files) without having to specify them as well.

G++ has extended the template instantiation syntax given in the ISO standard to allow forward declaration of explicit instantiations (with `extern`), instantiation of the compiler support data for a template class (i.e. the vtable) without instantiating any of its members (with `inline`), and instantiation of only the static data members of a template class, without the support data or member functions (with (`static`):

```
extern template int max (int, int);
inline template class Foo<int>;
static template class Foo<int>;
```

3. Do nothing. Pretend G++ does implement automatic instantiation management. Code written for the Borland model will work fine, but each translation unit will contain instances of each of the templates it uses. In a large program, this can lead to an unacceptable amount of code duplication.

7.6 Extracting the function pointer from a bound pointer to member function

In C++, pointer to member functions (PMFs) are implemented using a wide pointer of sorts to handle all the possible call mechanisms; the PMF needs to store information about how to adjust the '`this`' pointer, and if the function pointed to is virtual, where to find the vtable, and where in the vtable to look for the member function. If you are using PMFs in an inner loop, you should really reconsider that decision. If that is not an option, you can extract the pointer to the function that would be called for a given object/PMF pair and call it directly inside the inner loop, to save a bit of time.

Note that you will still be paying the penalty for the call through a function pointer; on most modern architectures, such a call defeats the branch prediction features of the CPU. This is also true of normal virtual function calls.

The syntax for this extension is

```
extern A a;
extern int (A::*fp)();
typedef int (*fptr)(A *);

fptr p = (fptr)(a.*fp);
```

For PMF constants (i.e. expressions of the form '`&Klasse::Member`'), no object is needed to obtain the address of the function. They can be converted to function pointers directly:

```
fptr p1 = (fptr)(&A::foo);
```

You must specify '`-Wno-pmf-conversions`' to use this extension.

7.7 C++-Specific Variable, Function, and Type Attributes

Some attributes only make sense for C++ programs.

`init_priority` (*priority*)

In Standard C++, objects defined at namespace scope are guaranteed to be initialized in an order in strict accordance with that of their definitions *in a given translation unit*. No guarantee is made for initializations across translation units. However, GNU C++ allows users to control the order of initialization of objects defined at namespace scope with the `init_priority` attribute by specifying a relative *priority*, a constant integral expression currently bounded between 101 and 65535 inclusive. Lower numbers indicate a higher priority.

In the following example, `A` would normally be created before `B`, but the `init_priority` attribute has reversed that order:

```
Some_Class  A  __attribute__ ((init_priority (2000)));
Some_Class  B  __attribute__ ((init_priority (543)));
```

Note that the particular values of *priority* do not matter; only their relative ordering.

`java_interface`

This type attribute informs C++ that the class is a Java interface. It may only be applied to classes declared within an `extern "Java"` block. Calls to methods declared in this interface will be dispatched using GCJ's interface table mechanism, instead of regular virtual table dispatch.

See also Section 7.8 [Namespace Association], page 630.

7.8 Namespace Association

Caution: The semantics of this extension are not fully defined. Users should refrain from using this extension as its semantics may change subtly over time. It is possible that this extension will be removed in future versions of G++.

A using-directive with `__attribute ((strong))` is stronger than a normal using-directive in two ways:

- Templates from the used namespace can be specialized and explicitly instantiated as though they were members of the using namespace.
- The using namespace is considered an associated namespace of all templates in the used namespace for purposes of argument-dependent name lookup.

The used namespace must be nested within the using namespace so that normal unqualified lookup works properly.

This is useful for composing a namespace transparently from implementation namespaces. For example:

```
namespace std {
  namespace debug {
    template <class T> struct A { };
  }
  using namespace debug __attribute ((__strong__));
  template <> struct A<int> { };   // ok to specialize

  template <class T> void f (A<T>);
}

int main()
{
  f (std::A<float>());             // lookup finds std::f
  f (std::A<int>());
}
```

7.9 Type Traits

The C++ front-end implements syntactic extensions that allow to determine at compile time various characteristics of a type (or of a pair of types).

`__has_nothrow_assign (type)`
: If `type` is const qualified or is a reference type then the trait is false. Otherwise if `__has_trivial_assign (type)` is true then the trait is true, else if `type` is a cv class or union type with copy assignment operators that are known not to throw an exception then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__has_nothrow_copy (type)`
: If `__has_trivial_copy (type)` is true then the trait is true, else if `type` is a cv class or union type with copy constructors that are known not to throw an exception then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__has_nothrow_constructor (type)`
: If `__has_trivial_constructor (type)` is true then the trait is true, else if `type` is a cv class or union type (or array thereof) with a default constructor that is known not to throw an exception then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__has_trivial_assign (type)`
: If `type` is const qualified or is a reference type then the trait is false. Otherwise if `__is_pod (type)` is true then the trait is true, else if `type` is a cv class or union type with a trivial copy assignment ([class.copy]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__has_trivial_copy (type)`
: If `__is_pod (type)` is true or `type` is a reference type then the trait is true, else if `type` is a cv class or union type with a trivial copy constructor ([class.copy]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__has_trivial_constructor (type)`
: If `__is_pod (type)` is true then the trait is true, else if `type` is a cv class or union type (or array thereof) with a trivial default constructor ([class.ctor]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__has_trivial_destructor (type)`
: If `__is_pod (type)` is true or `type` is a reference type then the trait is true, else if `type` is a cv class or union type (or array thereof) with a trivial destructor ([class.dtor]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__has_virtual_destructor (type)`
: If `type` is a class type with a virtual destructor ([class.dtor]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_abstract (type)`
: If `type` is an abstract class ([class.abstract]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_base_of (base_type, derived_type)`
: If `base_type` is a base class of `derived_type` ([class.derived]) then the trait is true, otherwise it is false. Top-level cv qualifications of `base_type` and `derived_type` are ignored. For the purposes of this trait, a class type is considered is own base. Requires: if `__is_class (base_type)` and `__is_class (derived_type)` are true and `base_type` and `derived_type` are not the same type (disregarding cv-qualifiers), `derived_type` shall be a complete type. Diagnostic is produced if this requirement is not met.

`__is_class (type)`
: If `type` is a cv class type, and not a union type ([basic.compound]) the trait is true, else it is false.

`__is_empty (type)`
: If `__is_class (type)` is false then the trait is false. Otherwise `type` is considered empty if and only if: `type` has no non-static data members, or all non-static data members, if any, are bit-fields of length 0, and `type` has no virtual members, and `type` has no virtual base classes, and `type` has no base classes `base_type` for which `__is_empty (base_type)` is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_enum (type)`
: If `type` is a cv enumeration type ([basic.compound]) the trait is true, else it is false.

`__is_literal_type (type)`
: If `type` is a literal type ([basic.types]) the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_pod (type)`
: If `type` is a cv POD type ([basic.types]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_polymorphic (type)`
: If `type` is a polymorphic class ([class.virtual]) then the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_standard_layout (type)`
: If `type` is a standard-layout type ([basic.types]) the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_trivial (type)`
: If `type` is a trivial type ([basic.types]) the trait is true, else it is false. Requires: `type` shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

`__is_union (type)`
: If `type` is a cv union type ([basic.compound]) the trait is true, else it is false.

`__underlying_type (type)`
: The underlying type of `type`. Requires: `type` shall be an enumeration type ([dcl.enum]).

7.10 Java Exceptions

The Java language uses a slightly different exception handling model from C++. Normally, GNU C++ will automatically detect when you are writing C++ code that uses Java

exceptions, and handle them appropriately. However, if C++ code only needs to execute destructors when Java exceptions are thrown through it, GCC will guess incorrectly. Sample problematic code is:

```
struct S { ~S(); };
extern void bar();     // is written in Java, and may throw exceptions
void foo()
{
  S s;
  bar();
}
```

The usual effect of an incorrect guess is a link failure, complaining of a missing routine called '`__gxx_personality_v0`'.

You can inform the compiler that Java exceptions are to be used in a translation unit, irrespective of what it might think, by writing '`#pragma GCC java_exceptions`' at the head of the file. This '`#pragma`' must appear before any functions that throw or catch exceptions, or run destructors when exceptions are thrown through them.

You cannot mix Java and C++ exceptions in the same translation unit. It is believed to be safe to throw a C++ exception from one file through another file compiled for the Java exception model, or vice versa, but there may be bugs in this area.

7.11 Deprecated Features

In the past, the GNU C++ compiler was extended to experiment with new features, at a time when the C++ language was still evolving. Now that the C++ standard is complete, some of those features are superseded by superior alternatives. Using the old features might cause a warning in some cases that the feature will be dropped in the future. In other cases, the feature might be gone already.

While the list below is not exhaustive, it documents some of the options that are now deprecated:

`-fexternal-templates`
`-falt-external-templates`
: These are two of the many ways for G++ to implement template instantiation. See Section 7.5 [Template Instantiation], page 626. The C++ standard clearly defines how template definitions have to be organized across implementation units. G++ has an implicit instantiation mechanism that should work just fine for standard-conforming code.

`-fstrict-prototype`
`-fno-strict-prototype`
: Previously it was possible to use an empty prototype parameter list to indicate an unspecified number of parameters (like C), rather than no parameters, as C++ demands. This feature has been removed, except where it is required for backwards compatibility. See Section 7.12 [Backwards Compatibility], page 634.

G++ allows a virtual function returning '`void *`' to be overridden by one returning a different pointer type. This extension to the covariant return type rules is now deprecated and will be removed from a future version.

The G++ minimum and maximum operators ('<?' and '>?') and their compound forms ('<?=') and '>?=') have been deprecated and are now removed from G++. Code using these operators should be modified to use `std::min` and `std::max` instead.

The named return value extension has been deprecated, and is now removed from G++.

The use of initializer lists with new expressions has been deprecated, and is now removed from G++.

Floating and complex non-type template parameters have been deprecated, and are now removed from G++.

The implicit typename extension has been deprecated and is now removed from G++.

The use of default arguments in function pointers, function typedefs and other places where they are not permitted by the standard is deprecated and will be removed from a future version of G++.

G++ allows floating-point literals to appear in integral constant expressions, e.g. ' `enum E { e = int(2.2 * 3.7) }` ' This extension is deprecated and will be removed from a future version.

G++ allows static data members of const floating-point type to be declared with an initializer in a class definition. The standard only allows initializers for static members of const integral types and const enumeration types so this extension has been deprecated and will be removed from a future version.

7.12 Backwards Compatibility

Now that there is a definitive ISO standard C++, G++ has a specification to adhere to. The C++ language evolved over time, and features that used to be acceptable in previous drafts of the standard, such as the ARM [Annotated C++ Reference Manual], are no longer accepted. In order to allow compilation of C++ written to such drafts, G++ contains some backwards compatibilities. *All such backwards compatibility features are liable to disappear in future versions of G++.* They should be considered deprecated. See Section 7.11 [Deprecated Features], page 633.

`For scope` If a variable is declared at for scope, it used to remain in scope until the end of the scope which contained the for statement (rather than just within the for scope). G++ retains this, but issues a warning, if such a variable is accessed outside the for scope.

`Implicit C language`
Old C system header files did not contain an `extern "C" {...}` scope to set the language. On such systems, all header files are implicitly scoped inside a C language scope. Also, an empty prototype `()` will be treated as an unspecified number of arguments, rather than no arguments, as C++ demands.

8 GNU Objective-C features

This document is meant to describe some of the GNU Objective-C features. It is not intended to teach you Objective-C. There are several resources on the Internet that present the language.

8.1 GNU Objective-C runtime API

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

The GNU Objective-C runtime provides an API that allows you to interact with the Objective-C runtime system, querying the live runtime structures and even manipulating them. This allows you for example to inspect and navigate classes, methods and protocols; to define new classes or new methods, and even to modify existing classes or protocols.

If you are using a "Foundation" library such as GNUstep-Base, this library will provide you with a rich set of functionality to do most of the inspection tasks, and you probably will only need direct access to the GNU Objective-C runtime API to define new classes or methods.

8.1.1 Modern GNU Objective-C runtime API

The GNU Objective-C runtime provides an API which is similar to the one provided by the "Objective-C 2.0" Apple/NeXT Objective-C runtime. The API is documented in the public header files of the GNU Objective-C runtime:

- '`objc/objc.h`': this is the basic Objective-C header file, defining the basic Objective-C types such as `id`, `Class` and `BOOL`. You have to include this header to do almost anything with Objective-C.
- '`objc/runtime.h`': this header declares most of the public runtime API functions allowing you to inspect and manipulate the Objective-C runtime data structures. These functions are fairly standardized across Objective-C runtimes and are almost identical to the Apple/NeXT Objective-C runtime ones. It does not declare functions in some specialized areas (constructing and forwarding message invocations, threading) which are in the other headers below. You have to include '`objc/objc.h`' and '`objc/runtime.h`' to use any of the functions, such as `class_getName()`, declared in '`objc/runtime.h`'.
- '`objc/message.h`': this header declares public functions used to construct, deconstruct and forward message invocations. Because messaging is done in quite a different way on different runtimes, functions in this header are specific to the GNU Objective-C runtime implementation.
- '`objc/objc-exception.h`': this header declares some public functions related to Objective-C exceptions. For example functions in this header allow you to throw an Objective-C exception from plain C/C++ code.
- '`objc/objc-sync.h`': this header declares some public functions related to the Objective-C `@synchronized()` syntax, allowing you to emulate an Objective-C `@synchronized()` block in plain C/C++ code.

- '`objc/thr.h`': this header declares a public runtime API threading layer that is only provided by the GNU Objective-C runtime. It declares functions such as `objc_mutex_lock()`, which provide a platform-independent set of threading functions.

The header files contain detailed documentation for each function in the GNU Objective-C runtime API.

8.1.2 Traditional GNU Objective-C runtime API

The GNU Objective-C runtime used to provide a different API, which we call the "traditional" GNU Objective-C runtime API. Functions belonging to this API are easy to recognize because they use a different naming convention, such as `class_get_super_class()` (traditional API) instead of `class_getSuperclass()` (modern API). Software using this API includes the file '`objc/objc-api.h`' where it is declared.

Starting with GCC 4.7.0, the traditional GNU runtime API is no longer available.

8.2 `+load`: Executing code before main

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

The GNU Objective-C runtime provides a way that allows you to execute code before the execution of the program enters the `main` function. The code is executed on a per-class and a per-category basis, through a special class method `+load`.

This facility is very useful if you want to initialize global variables which can be accessed by the program directly, without sending a message to the class first. The usual way to initialize global variables, in the `+initialize` method, might not be useful because `+initialize` is only called when the first message is sent to a class object, which in some cases could be too late.

Suppose for example you have a `FileStream` class that declares `Stdin`, `Stdout` and `Stderr` as global variables, like below:

```
FileStream *Stdin = nil;
FileStream *Stdout = nil;
FileStream *Stderr = nil;

@implementation FileStream

+ (void)initialize
{
    Stdin = [[FileStream new] initWithFd:0];
    Stdout = [[FileStream new] initWithFd:1];
    Stderr = [[FileStream new] initWithFd:2];
}

/* Other methods here */
@end
```

In this example, the initialization of `Stdin`, `Stdout` and `Stderr` in `+initialize` occurs too late. The programmer can send a message to one of these objects before the variables are actually initialized, thus sending messages to the `nil` object. The `+initialize` method which actually initializes the global variables is not invoked until the first message is sent

to the class object. The solution would require these variables to be initialized just before entering `main`.

The correct solution of the above problem is to use the `+load` method instead of `+initialize`:

```
@implementation FileStream

+ (void)load
{
    Stdin = [[FileStream new] initWithFd:0];
    Stdout = [[FileStream new] initWithFd:1];
    Stderr = [[FileStream new] initWithFd:2];
}

/* Other methods here */
@end
```

The `+load` is a method that is not overridden by categories. If a class and a category of it both implement `+load`, both methods are invoked. This allows some additional initializations to be performed in a category.

This mechanism is not intended to be a replacement for `+initialize`. You should be aware of its limitations when you decide to use it instead of `+initialize`.

8.2.1 What you can and what you cannot do in `+load`

`+load` is to be used only as a last resort. Because it is executed very early, most of the Objective-C runtime machinery will not be ready when `+load` is executed; hence `+load` works best for executing C code that is independent on the Objective-C runtime.

The `+load` implementation in the GNU runtime guarantees you the following things:

- you can write whatever C code you like;
- you can allocate and send messages to objects whose class is implemented in the same file;
- the `+load` implementation of all super classes of a class are executed before the `+load` of that class is executed;
- the `+load` implementation of a class is executed before the `+load` implementation of any category.

In particular, the following things, even if they can work in a particular case, are not guaranteed:

- allocation of or sending messages to arbitrary objects;
- allocation of or sending messages to objects whose classes have a category implemented in the same file;
- sending messages to Objective-C constant strings (`@"this is a constant string"`);

You should make no assumptions about receiving `+load` in sibling classes when you write `+load` of a class. The order in which sibling classes receive `+load` is not guaranteed.

The order in which `+load` and `+initialize` are called could be problematic if this matters. If you don't allocate objects inside `+load`, it is guaranteed that `+load` is called before `+initialize`. If you create an object inside `+load` the `+initialize` method of object's

class is invoked even if `+load` was not invoked. Note if you explicitly call `+load` on a class, `+initialize` will be called first. To avoid possible problems try to implement only one of these methods.

The `+load` method is also invoked when a bundle is dynamically loaded into your running program. This happens automatically without any intervening operation from you. When you write bundles and you need to write `+load` you can safely create and send messages to objects whose classes already exist in the running program. The same restrictions as above apply to classes defined in bundle.

8.3 Type encoding

This is an advanced section. Type encodings are used extensively by the compiler and by the runtime, but you generally do not need to know about them to use Objective-C.

The Objective-C compiler generates type encodings for all the types. These type encodings are used at runtime to find out information about selectors and methods and about objects and classes.

The types are encoded in the following way:

`_Bool`	`B`
`char`	`c`
`unsigned char`	`C`
`short`	`s`
`unsigned short`	`S`
`int`	`i`
`unsigned int`	`I`
`long`	`l`
`unsigned long`	`L`
`long long`	`q`
`unsigned long long`	`Q`
`float`	`f`
`double`	`d`
`long double`	`D`
`void`	`v`
`id`	`@`
`Class`	`#`
`SEL`	`:`
`char*`	`*`
`enum`	an `enum` is encoded exactly as the integer type that the compiler uses for it, which depends on the enumeration values. Often the compiler users `unsigned int`, which is then encoded as `I`.
unknown type	`?`
Complex types	`j` followed by the inner type. For example `_Complex double` is encoded as "jd".
bit-fields	`b` followed by the starting position of the bit-field, the type of the bit-field and the size of the bit-field (the bit-fields encoding was changed from the NeXT's compiler encoding, see below)

The encoding of bit-fields has changed to allow bit-fields to be properly handled by the runtime functions that compute sizes and alignments of types that contain bit-fields. The previous encoding contained only the size of the bit-field. Using only this information it is not possible to reliably compute the size occupied by the bit-field. This is very important in the presence of the Boehm's garbage collector because the objects are allocated using the typed memory facility available in this collector. The typed memory allocation requires information about where the pointers are located inside the object.

The position in the bit-field is the position, counting in bits, of the bit closest to the beginning of the structure.

The non-atomic types are encoded as follows:

pointers	'`^`' followed by the pointed type.
arrays	'`[`' followed by the number of elements in the array followed by the type of the elements followed by '`]`'
structures	'`{`' followed by the name of the structure (or '`?`' if the structure is unnamed), the '`=`' sign, the type of the members and by '`}`'
unions	'`(`' followed by the name of the structure (or '`?`' if the union is unnamed), the '`=`' sign, the type of the members followed by '`)`'
vectors	'`![`' followed by the vector_size (the number of bytes composing the vector) followed by a comma, followed by the alignment (in bytes) of the vector, followed by the type of the elements followed by '`]`'

Here are some types and their encodings, as they are generated by the compiler on an i386 machine:

Objective-C type	Compiler encoding
`int a[10];`	`[10i]`
`struct { int i; float f[3]; int a:3; int b:2; char c; }`	`{?=i[3f]b128i3b131i2c}`
`int a __attribute__ ((vector_size(16)));`	`![16,16i]` (alignment would depend on the machine)

In addition to the types the compiler also encodes the type specifiers. The table below describes the encoding of the current Objective-C type specifiers:

Specifier	Encoding
`const`	`r`
`in`	`n`
`inout`	`N`
`out`	`o`
`bycopy`	`O`

byref	R
oneway	V

The type specifiers are encoded just before the type. Unlike types however, the type specifiers are only encoded when they appear in method argument types.

Note how `const` interacts with pointers:

Objective-C type	Compiler encoding
`const int`	`ri`
`const int*`	`^ri`
`int *const`	`r^i`

`const int*` is a pointer to a `const int`, and so is encoded as `^ri`. `int* const`, instead, is a `const` pointer to an `int`, and so is encoded as `r^i`.

Finally, there is a complication when encoding `const char *` versus `char * const`. Because `char *` is encoded as `*` and not as `^c`, there is no way to express the fact that `r` applies to the pointer or to the pointee.

Hence, it is assumed as a convention that `r*` means `const char *` (since it is what is most often meant), and there is no way to encode `char *const`. `char *const` would simply be encoded as `*`, and the `const` is lost.

8.3.1 Legacy type encoding

Unfortunately, historically GCC used to have a number of bugs in its encoding code. The NeXT runtime expects GCC to emit type encodings in this historical format (compatible with GCC-3.3), so when using the NeXT runtime, GCC will introduce on purpose a number of incorrect encodings:

- the read-only qualifier of the pointee gets emitted before the '^'. The read-only qualifier of the pointer itself gets ignored, unless it is a typedef. Also, the 'r' is only emitted for the outermost type.
- 32-bit longs are encoded as 'l' or 'L', but not always. For typedefs, the compiler uses 'i' or 'I' instead if encoding a struct field or a pointer.
- `enum`s are always encoded as 'i' (int) even if they are actually unsigned or long.

In addition to that, the NeXT runtime uses a different encoding for bitfields. It encodes them as `b` followed by the size, without a bit offset or the underlying field type.

8.3.2 `@encode`

GNU Objective-C supports the `@encode` syntax that allows you to create a type encoding from a C/Objective-C type. For example, `@encode(int)` is compiled by the compiler into `"i"`.

`@encode` does not support type qualifiers other than `const`. For example, `@encode(const char*)` is valid and is compiled into `"r*"`, while `@encode(bycopy char *)` is invalid and will cause a compilation error.

8.3.3 Method signatures

This section documents the encoding of method types, which is rarely needed to use Objective-C. You should skip it at a first reading; the runtime provides functions that will work on methods and can walk through the list of parameters and interpret them for you. These functions are part of the public "API" and are the preferred way to interact with method signatures from user code.

But if you need to debug a problem with method signatures and need to know how they are implemented (i.e., the "ABI"), read on.

Methods have their "signature" encoded and made available to the runtime. The "signature" encodes all the information required to dynamically build invocations of the method at runtime: return type and arguments.

The "signature" is a null-terminated string, composed of the following:

- The return type, including type qualifiers. For example, a method returning `int` would have `i` here.
- The total size (in bytes) required to pass all the parameters. This includes the two hidden parameters (the object `self` and the method selector `_cmd`).
- Each argument, with the type encoding, followed by the offset (in bytes) of the argument in the list of parameters.

For example, a method with no arguments and returning `int` would have the signature `i8@0:4` if the size of a pointer is 4. The signature is interpreted as follows: the `i` is the return type (an `int`), the `8` is the total size of the parameters in bytes (two pointers each of size 4), the `@0` is the first parameter (an object at byte offset `0`) and `:4` is the second parameter (a `SEL` at byte offset `4`).

You can easily find more examples by running the "strings" program on an Objective-C object file compiled by GCC. You'll see a lot of strings that look very much like `i8@0:4`. They are signatures of Objective-C methods.

8.4 Garbage Collection

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

Support for garbage collection with the GNU runtime has been added by using a powerful conservative garbage collector, known as the Boehm-Demers-Weiser conservative garbage collector.

To enable the support for it you have to configure the compiler using an additional argument, '`--enable-objc-gc`'. This will build the boehm-gc library, and build an additional runtime library which has several enhancements to support the garbage collector. The new library has a new name, '`libobjc_gc.a`' to not conflict with the non-garbage-collected library.

When the garbage collector is used, the objects are allocated using the so-called typed memory allocation mechanism available in the Boehm-Demers-Weiser collector. This mode requires precise information on where pointers are located inside objects. This information is computed once per class, immediately after the class has been initialized.

There is a new runtime function `class_ivar_set_gcinvisible()` which can be used to declare a so-called *weak pointer* reference. Such a pointer is basically hidden for the garbage collector; this can be useful in certain situations, especially when you want to keep track of the allocated objects, yet allow them to be collected. This kind of pointers can only be members of objects, you cannot declare a global pointer as a weak reference. Every type which is a pointer type can be declared a weak pointer, including `id`, `Class` and `SEL`.

Here is an example of how to use this feature. Suppose you want to implement a class whose instances hold a weak pointer reference; the following class does this:

```
@interface WeakPointer : Object
{
    const void* weakPointer;
}

- initWithPointer:(const void*)p;
- (const void*)weakPointer;
@end


@implementation WeakPointer

+ (void)initialize
{
  if (self == objc_lookUpClass ("WeakPointer"))
    class_ivar_set_gcinvisible (self, "weakPointer", YES);
}

- initWithPointer:(const void*)p
{
  weakPointer = p;
  return self;
}

- (const void*)weakPointer
{
  return weakPointer;
}

@end
```

Weak pointers are supported through a new type character specifier represented by the '`!`' character. The `class_ivar_set_gcinvisible()` function adds or removes this specifier to the string type description of the instance variable named as argument.

8.5 Constant string objects

GNU Objective-C provides constant string objects that are generated directly by the compiler. You declare a constant string object by prefixing a C constant string with the character '`@`':

```
id myString = @"this is a constant string object";
```

The constant string objects are by default instances of the `NXConstantString` class which is provided by the GNU Objective-C runtime. To get the definition of this class you must include the '`objc/NXConstStr.h`' header file.

User defined libraries may want to implement their own constant string class. To be able to support them, the GNU Objective-C compiler provides a new command line options '`-fconstant-string-class=`*class-name*'. The provided class should adhere to a strict structure, the same as `NXConstantString`'s structure:

```
@interface MyConstantStringClass
{
  Class isa;
  char *c_string;
  unsigned int len;
}
@end
```

`NXConstantString` inherits from `Object`; user class libraries may choose to inherit the customized constant string class from a different class than `Object`. There is no requirement in the methods the constant string class has to implement, but the final ivar layout of the class must be the compatible with the given structure.

When the compiler creates the statically allocated constant string object, the `c_string` field will be filled by the compiler with the string; the `length` field will be filled by the compiler with the string length; the `isa` pointer will be filled with `NULL` by the compiler, and it will later be fixed up automatically at runtime by the GNU Objective-C runtime library to point to the class which was set by the '`-fconstant-string-class`' option when the object file is loaded (if you wonder how it works behind the scenes, the name of the class to use, and the list of static objects to fixup, are stored by the compiler in the object file in a place where the GNU runtime library will find them at runtime).

As a result, when a file is compiled with the '`-fconstant-string-class`' option, all the constant string objects will be instances of the class specified as argument to this option. It is possible to have multiple compilation units referring to different constant string classes, neither the compiler nor the linker impose any restrictions in doing this.

8.6 compatibility_alias

The keyword `@compatibility_alias` allows you to define a class name as equivalent to another class name. For example:

```
@compatibility_alias WOApplication GSWApplication;
```

tells the compiler that each time it encounters `WOApplication` as a class name, it should replace it with `GSWApplication` (that is, `WOApplication` is just an alias for `GSWApplication`).

There are some constraints on how this can be used—

- `WOApplication` (the alias) must not be an existing class;
- `GSWApplication` (the real class) must be an existing class.

8.7 Exceptions

GNU Objective-C provides exception support built into the language, as in the following example:

```
@try {
  ...
```

```
    @throw expr;
  ...
}
@catch (AnObjCClass *exc) {
  ...
    @throw expr;
  ...
    @throw;
  ...
}
@catch (AnotherClass *exc) {
  ...
}
@catch (id allOthers) {
  ...
}
@finally {
  ...
    @throw expr;
  ...
}
```

The `@throw` statement may appear anywhere in an Objective-C or Objective-C++ program; when used inside of a `@catch` block, the `@throw` may appear without an argument (as shown above), in which case the object caught by the `@catch` will be rethrown.

Note that only (pointers to) Objective-C objects may be thrown and caught using this scheme. When an object is thrown, it will be caught by the nearest `@catch` clause capable of handling objects of that type, analogously to how `catch` blocks work in C++ and Java. A `@catch(id ...)` clause (as shown above) may also be provided to catch any and all Objective-C exceptions not caught by previous `@catch` clauses (if any).

The `@finally` clause, if present, will be executed upon exit from the immediately preceding `@try ... @catch` section. This will happen regardless of whether any exceptions are thrown, caught or rethrown inside the `@try ... @catch` section, analogously to the behavior of the `finally` clause in Java.

There are several caveats to using the new exception mechanism:

- The '`-fobjc-exceptions`' command line option must be used when compiling Objective-C files that use exceptions.
- With the GNU runtime, exceptions are always implemented as "native" exceptions and it is recommended that the '`-fexceptions`' and '`-shared-libgcc`' options are used when linking.
- With the NeXT runtime, although currently designed to be binary compatible with `NS_HANDLER`-style idioms provided by the `NSException` class, the new exceptions can only be used on Mac OS X 10.3 (Panther) and later systems, due to additional functionality needed in the NeXT Objective-C runtime.
- As mentioned above, the new exceptions do not support handling types other than Objective-C objects. Furthermore, when used from Objective-C++, the Objective-C exception model does not interoperate with C++ exceptions at this time. This means you cannot `@throw` an exception from Objective-C and `catch` it in C++, or vice versa (i.e., `throw ... @catch`).

8.8 Synchronization

GNU Objective-C provides support for synchronized blocks:

```
@synchronized (ObjCClass *guard) {
  ...
}
```

Upon entering the `@synchronized` block, a thread of execution shall first check whether a lock has been placed on the corresponding `guard` object by another thread. If it has, the current thread shall wait until the other thread relinquishes its lock. Once `guard` becomes available, the current thread will place its own lock on it, execute the code contained in the `@synchronized` block, and finally relinquish the lock (thereby making `guard` available to other threads).

Unlike Java, Objective-C does not allow for entire methods to be marked `@synchronized`. Note that throwing exceptions out of `@synchronized` blocks is allowed, and will cause the guarding object to be unlocked properly.

Because of the interactions between synchronization and exception handling, you can only use `@synchronized` when compiling with exceptions enabled, that is with the command line option '`-fobjc-exceptions`'.

8.9 Fast enumeration

8.9.1 Using fast enumeration

GNU Objective-C provides support for the fast enumeration syntax:

```
id array = ...;
id object;

for (object in array)
{
  /* Do something with 'object' */
}
```

`array` needs to be an Objective-C object (usually a collection object, for example an array, a dictionary or a set) which implements the "Fast Enumeration Protocol" (see below). If you are using a Foundation library such as GNUstep Base or Apple Cocoa Foundation, all collection objects in the library implement this protocol and can be used in this way.

The code above would iterate over all objects in `array`. For each of them, it assigns it to `object`, then executes the `Do something with 'object'` statements.

Here is a fully worked-out example using a Foundation library (which provides the implementation of `NSArray`, `NSString` and `NSLog`):

```
NSArray *array = [NSArray arrayWithObjects: @"1", @"2", @"3", nil];
NSString *object;

for (object in array)
  NSLog (@"Iterating over %@", object);
```

8.9.2 c99-like fast enumeration syntax

A c99-like declaration syntax is also allowed:

```
id array = ...;
```

```
for (id object in array)
{
  /* Do something with 'object'  */
}
```

this is completely equivalent to:

```
id array = ...;

{
  id object;
  for (object in array)
  {
    /* Do something with 'object'  */
  }
}
```

but can save some typing.

Note that the option '`-std=c99`' is not required to allow this syntax in Objective-C.

8.9.3 Fast enumeration details

Here is a more technical description with the gory details. Consider the code

```
for (object expression in collection expression)
{
  statements
}
```

here is what happens when you run it:

- *`collection expression`* is evaluated exactly once and the result is used as the collection object to iterate over. This means it is safe to write code such as `for (object in [NSDictionary keyEnumerator]) ...`.
- the iteration is implemented by the compiler by repeatedly getting batches of objects from the collection object using the fast enumeration protocol (see below), then iterating over all objects in the batch. This is faster than a normal enumeration where objects are retrieved one by one (hence the name "fast enumeration").
- if there are no objects in the collection, then *`object expression`* is set to `nil` and the loop immediately terminates.
- if there are objects in the collection, then for each object in the collection (in the order they are returned) *`object expression`* is set to the object, then *`statements`* are executed.
- *`statements`* can contain `break` and `continue` commands, which will abort the iteration or skip to the next loop iteration as expected.
- when the iteration ends because there are no more objects to iterate over, *`object expression`* is set to `nil`. This allows you to determine whether the iteration finished because a `break` command was used (in which case *`object expression`* will remain set to the last object that was iterated over) or because it iterated over all the objects (in which case *`object expression`* will be set to `nil`).
- *`statements`* must not make any changes to the collection object; if they do, it is a hard error and the fast enumeration terminates by invoking `objc_enumerationMutation`, a runtime function that normally aborts the program but which can be customized by Foundation libraries via `objc_set_mutation_handler` to do something different, such as raising an exception.

8.9.4 Fast enumeration protocol

If you want your own collection object to be usable with fast enumeration, you need to have it implement the method

```
- (unsigned long) countByEnumeratingWithState: (NSFastEnumerationState *)state
                                      objects: (id *)objects
                                        count: (unsigned long)len;
```

where `NSFastEnumerationState` must be defined in your code as follows:

```
typedef struct
{
  unsigned long state;
  id            *itemsPtr;
  unsigned long *mutationsPtr;
  unsigned long extra[5];
} NSFastEnumerationState;
```

If no `NSFastEnumerationState` is defined in your code, the compiler will automatically replace `NSFastEnumerationState *` with `struct __objcFastEnumerationState *`, where that type is silently defined by the compiler in an identical way. This can be confusing and we recommend that you define `NSFastEnumerationState` (as shown above) instead.

The method is called repeatedly during a fast enumeration to retrieve batches of objects. Each invocation of the method should retrieve the next batch of objects.

The return value of the method is the number of objects in the current batch; this should not exceed `len`, which is the maximum size of a batch as requested by the caller. The batch itself is returned in the `itemsPtr` field of the `NSFastEnumerationState` struct.

To help with returning the objects, the `objects` array is a C array preallocated by the caller (on the stack) of size `len`. In many cases you can put the objects you want to return in that `objects` array, then do `itemsPtr = objects`. But you don't have to; if your collection already has the objects to return in some form of C array, it could return them from there instead.

The `state` and `extra` fields of the `NSFastEnumerationState` structure allows your collection object to keep track of the state of the enumeration. In a simple array implementation, `state` may keep track of the index of the last object that was returned, and `extra` may be unused.

The `mutationsPtr` field of the `NSFastEnumerationState` is used to keep track of mutations. It should point to a number; before working on each object, the fast enumeration loop will check that this number has not changed. If it has, a mutation has happened and the fast enumeration will abort. So, `mutationsPtr` could be set to point to some sort of version number of your collection, which is increased by one every time there is a change (for example when an object is added or removed). Or, if you are content with less strict mutation checks, it could point to the number of objects in your collection or some other value that can be checked to perform an approximate check that the collection has not been mutated.

Finally, note how we declared the `len` argument and the return value to be of type `unsigned long`. They could also be declared to be of type `unsigned int` and everything would still work.

8.10 Messaging with the GNU Objective-C runtime

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

The implementation of messaging in the GNU Objective-C runtime is designed to be portable, and so is based on standard C.

Sending a message in the GNU Objective-C runtime is composed of two separate steps. First, there is a call to the lookup function, `objc_msg_lookup ()` (or, in the case of messages to super, `objc_msg_lookup_super ()`). This runtime function takes as argument the receiver and the selector of the method to be called; it returns the `IMP`, that is a pointer to the function implementing the method. The second step of method invocation consists of casting this pointer function to the appropriate function pointer type, and calling the function pointed to it with the right arguments.

For example, when the compiler encounters a method invocation such as `[object init]`, it compiles it into a call to `objc_msg_lookup (object, @selector(init))` followed by a cast of the returned value to the appropriate function pointer type, and then it calls it.

8.10.1 Dynamically registering methods

If `objc_msg_lookup()` does not find a suitable method implementation, because the receiver does not implement the required method, it tries to see if the class can dynamically register the method.

To do so, the runtime checks if the class of the receiver implements the method

```
+ (BOOL) resolveInstanceMethod: (SEL)selector;
```

in the case of an instance method, or

```
+ (BOOL) resolveClassMethod: (SEL)selector;
```

in the case of a class method. If the class implements it, the runtime invokes it, passing as argument the selector of the original method, and if it returns `YES`, the runtime tries the lookup again, which could now succeed if a matching method was added dynamically by `+resolveInstanceMethod:` or `+resolveClassMethod:`.

This allows classes to dynamically register methods (by adding them to the class using `class_addMethod`) when they are first called. To do so, a class should implement `+resolveInstanceMethod:` (or, depending on the case, `+resolveClassMethod:`) and have it recognize the selectors of methods that can be registered dynamically at runtime, register them, and return `YES`. It should return `NO` for methods that it does not dynamically registered at runtime.

If `+resolveInstanceMethod:` (or `+resolveClassMethod:`) is not implemented or returns `NO`, the runtime then tries the forwarding hook.

Support for `+resolveInstanceMethod:` and `resolveClassMethod:` was added to the GNU Objective-C runtime in GCC version 4.6.

8.10.2 Forwarding hook

The GNU Objective-C runtime provides a hook, called `__objc_msg_forward2`, which is called by `objc_msg_lookup()` when it can't find a method implementation in the runtime tables and after calling `+resolveInstanceMethod:` and `+resolveClassMethod:` has been attempted and did not succeed in dynamically registering the method.

To configure the hook, you set the global variable `__objc_msg_foward2` to a function with the same argument and return types of `objc_msg_lookup()`. When `objc_msg_lookup()` can not find a method implementation, it invokes the hook function you provided to get a method implementation to return. So, in practice `__objc_msg_forward2` allows you to extend `objc_msg_lookup()` by adding some custom code that is called to do a further lookup when no standard method implementation can be found using the normal lookup.

This hook is generally reserved for "Foundation" libraries such as GNUstep Base, which use it to implement their high-level method forwarding API, typically based around the `forwardInvocation:` method. So, unless you are implementing your own "Foundation" library, you should not set this hook.

In a typical forwarding implementation, the `__objc_msg_forward2` hook function determines the argument and return type of the method that is being looked up, and then creates a function that takes these arguments and has that return type, and returns it to the caller. Creating this function is non-trivial and is typically performed using a dedicated library such as `libffi`.

The forwarding method implementation thus created is returned by `objc_msg_lookup()` and is executed as if it was a normal method implementation. When the forwarding method implementation is called, it is usually expected to pack all arguments into some sort of object (typically, an `NSInvocation` in a "Foundation" library), and hand it over to the programmer (`forwardInvocation:`) who is then allowed to manipulate the method invocation using a high-level API provided by the "Foundation" library. For example, the programmer may want to examine the method invocation arguments and name and potentially change them before forwarding the method invocation to one or more local objects (`performInvocation:`) or even to remote objects (by using Distributed Objects or some other mechanism). When all this completes, the return value is passed back and must be returned correctly to the original caller.

Note that the GNU Objective-C runtime currently provides no support for method forwarding or method invocations other than the `__objc_msg_forward2` hook.

If the forwarding hook does not exist or returns `NULL`, the runtime currently attempts forwarding using an older, deprecated API, and if that fails, it aborts the program. In future versions of the GNU Objective-C runtime, the runtime will immediately abort.

9 Binary Compatibility

Binary compatibility encompasses several related concepts:

application binary interface (ABI)
: The set of runtime conventions followed by all of the tools that deal with binary representations of a program, including compilers, assemblers, linkers, and language runtime support. Some ABIs are formal with a written specification, possibly designed by multiple interested parties. Others are simply the way things are actually done by a particular set of tools.

ABI conformance
: A compiler conforms to an ABI if it generates code that follows all of the specifications enumerated by that ABI. A library conforms to an ABI if it is implemented according to that ABI. An application conforms to an ABI if it is built using tools that conform to that ABI and does not contain source code that specifically changes behavior specified by the ABI.

calling conventions
: Calling conventions are a subset of an ABI that specify of how arguments are passed and function results are returned.

interoperability
: Different sets of tools are interoperable if they generate files that can be used in the same program. The set of tools includes compilers, assemblers, linkers, libraries, header files, startup files, and debuggers. Binaries produced by different sets of tools are not interoperable unless they implement the same ABI. This applies to different versions of the same tools as well as tools from different vendors.

intercallability
: Whether a function in a binary built by one set of tools can call a function in a binary built by a different set of tools is a subset of interoperability.

implementation-defined features
: Language standards include lists of implementation-defined features whose behavior can vary from one implementation to another. Some of these features are normally covered by a platform's ABI and others are not. The features that are not covered by an ABI generally affect how a program behaves, but not intercallability.

compatibility
: Conformance to the same ABI and the same behavior of implementation-defined features are both relevant for compatibility.

The application binary interface implemented by a C or C++ compiler affects code generation and runtime support for:

- size and alignment of data types
- layout of structured types
- calling conventions

- register usage conventions
- interfaces for runtime arithmetic support
- object file formats

In addition, the application binary interface implemented by a C++ compiler affects code generation and runtime support for:

- name mangling
- exception handling
- invoking constructors and destructors
- layout, alignment, and padding of classes
- layout and alignment of virtual tables

Some GCC compilation options cause the compiler to generate code that does not conform to the platform's default ABI. Other options cause different program behavior for implementation-defined features that are not covered by an ABI. These options are provided for consistency with other compilers that do not follow the platform's default ABI or the usual behavior of implementation-defined features for the platform. Be very careful about using such options.

Most platforms have a well-defined ABI that covers C code, but ABIs that cover C++ functionality are not yet common.

Starting with GCC 3.2, GCC binary conventions for C++ are based on a written, vendor-neutral C++ ABI that was designed to be specific to 64-bit Itanium but also includes generic specifications that apply to any platform. This C++ ABI is also implemented by other compiler vendors on some platforms, notably GNU/Linux and BSD systems. We have tried hard to provide a stable ABI that will be compatible with future GCC releases, but it is possible that we will encounter problems that make this difficult. Such problems could include different interpretations of the C++ ABI by different vendors, bugs in the ABI, or bugs in the implementation of the ABI in different compilers. GCC's '`-Wabi`' switch warns when G++ generates code that is probably not compatible with the C++ ABI.

The C++ library used with a C++ compiler includes the Standard C++ Library, with functionality defined in the C++ Standard, plus language runtime support. The runtime support is included in a C++ ABI, but there is no formal ABI for the Standard C++ Library. Two implementations of that library are interoperable if one follows the de-facto ABI of the other and if they are both built with the same compiler, or with compilers that conform to the same ABI for C++ compiler and runtime support.

When G++ and another C++ compiler conform to the same C++ ABI, but the implementations of the Standard C++ Library that they normally use do not follow the same ABI for the Standard C++ Library, object files built with those compilers can be used in the same program only if they use the same C++ library. This requires specifying the location of the C++ library header files when invoking the compiler whose usual library is not being used. The location of GCC's C++ header files depends on how the GCC build was configured, but can be seen by using the G++ '`-v`' option. With default configuration options for G++ 3.3 the compile line for a different C++ compiler needs to include

```
-Igcc_install_directory/include/c++/3.3
```

Similarly, compiling code with G++ that must use a C++ library other than the GNU C++ library requires specifying the location of the header files for that other library.

The most straightforward way to link a program to use a particular C++ library is to use a C++ driver that specifies that C++ library by default. The `g++` driver, for example, tells the linker where to find GCC's C++ library ('`libstdc++`') plus the other libraries and startup files it needs, in the proper order.

If a program must use a different C++ library and it's not possible to do the final link using a C++ driver that uses that library by default, it is necessary to tell `g++` the location and name of that library. It might also be necessary to specify different startup files and other runtime support libraries, and to suppress the use of GCC's support libraries with one or more of the options '`-nostdlib`', '`-nostartfiles`', and '`-nodefaultlibs`'.

10 gcov—a Test Coverage Program

gcov is a tool you can use in conjunction with GCC to test code coverage in your programs.

10.1 Introduction to gcov

gcov is a test coverage program. Use it in concert with GCC to analyze your programs to help create more efficient, faster running code and to discover untested parts of your program. You can use gcov as a profiling tool to help discover where your optimization efforts will best affect your code. You can also use gcov along with the other profiling tool, gprof, to assess which parts of your code use the greatest amount of computing time.

Profiling tools help you analyze your code's performance. Using a profiler such as gcov or gprof, you can find out some basic performance statistics, such as:

- how often each line of code executes
- what lines of code are actually executed
- how much computing time each section of code uses

Once you know these things about how your code works when compiled, you can look at each module to see which modules should be optimized. gcov helps you determine where to work on optimization.

Software developers also use coverage testing in concert with testsuites, to make sure software is actually good enough for a release. Testsuites can verify that a program works as expected; a coverage program tests to see how much of the program is exercised by the testsuite. Developers can then determine what kinds of test cases need to be added to the testsuites to create both better testing and a better final product.

You should compile your code without optimization if you plan to use gcov because the optimization, by combining some lines of code into one function, may not give you as much information as you need to look for 'hot spots' where the code is using a great deal of computer time. Likewise, because gcov accumulates statistics by line (at the lowest resolution), it works best with a programming style that places only one statement on each line. If you use complicated macros that expand to loops or to other control structures, the statistics are less helpful—they only report on the line where the macro call appears. If your complex macros behave like functions, you can replace them with inline functions to solve this problem.

gcov creates a logfile called '*sourcefile*.gcov' which indicates how many times each line of a source file '*sourcefile*.c' has executed. You can use these logfiles along with gprof to aid in fine-tuning the performance of your programs. gprof gives timing information you can use along with the information you get from gcov.

gcov works only on code compiled with GCC. It is not compatible with any other profiling or test coverage mechanism.

10.2 Invoking gcov

```
gcov [options] files
```

gcov accepts the following options:

`-h`
`--help` Display help about using `gcov` (on the standard output), and exit without doing any further processing.

`-v`
`--version`
Display the `gcov` version number (on the standard output), and exit without doing any further processing.

`-a`
`--all-blocks`
Write individual execution counts for every basic block. Normally gcov outputs execution counts only for the main blocks of a line. With this option you can determine if blocks within a single line are not being executed.

`-b`
`--branch-probabilities`
Write branch frequencies to the output file, and write branch summary info to the standard output. This option allows you to see how often each branch in your program was taken. Unconditional branches will not be shown, unless the '`-u`' option is given.

`-c`
`--branch-counts`
Write branch frequencies as the number of branches taken, rather than the percentage of branches taken.

`-n`
`--no-output`
Do not create the `gcov` output file.

`-l`
`--long-file-names`
Create long file names for included source files. For example, if the header file '`x.h`' contains code, and was included in the file '`a.c`', then running `gcov` on the file '`a.c`' will produce an output file called '`a.c##x.h.gcov`' instead of '`x.h.gcov`'. This can be useful if '`x.h`' is included in multiple source files and you want to see the individual contributions. If you use the '`-p`' option, both the including and included file names will be complete path names.

`-p`
`--preserve-paths`
Preserve complete path information in the names of generated '`.gcov`' files. Without this option, just the filename component is used. With this option, all directories are used, with '`/`' characters translated to '`#`' characters, '`.`' directory components removed and unremoveable '`..`' components renamed to '`^`'. This is useful if sourcefiles are in several different directories.

`-r`
`--relative-only`
: Only output information about source files with a relative pathname (after source prefix elision). Absolute paths are usually system header files and coverage of any inline functions therein is normally uninteresting.

`-f`
`--function-summaries`
: Output summaries for each function in addition to the file level summary.

`-o` *directory|file*
`--object-directory` *directory*
`--object-file` *file*
: Specify either the directory containing the gcov data files, or the object path name. The '`.gcno`', and '`.gcda`' data files are searched for using this option. If a directory is specified, the data files are in that directory and named after the input file name, without its extension. If a file is specified here, the data files are named after that file, without its extension.

`-s` *directory*
`--source-prefix` *directory*
: A prefix for source file names to remove when generating the output coverage files. This option is useful when building in a separate directory, and the pathname to the source directory is not wanted when determining the output file names. Note that this prefix detection is applied before determining whether the source file is absolute.

`-u`
`--unconditional-branches`
: When branch probabilities are given, include those of unconditional branches. Unconditional branches are normally not interesting.

`-d`
`--display-progress`
: Display the progress on the standard output.

`gcov` should be run with the current directory the same as that when you invoked the compiler. Otherwise it will not be able to locate the source files. `gcov` produces files called '*mangledname*`.gcov`' in the current directory. These contain the coverage information of the source file they correspond to. One '`.gcov`' file is produced for each source (or header) file containing code, which was compiled to produce the data files. The *mangledname* part of the output file name is usually simply the source file name, but can be something more complicated if the '`-l`' or '`-p`' options are given. Refer to those options for details.

If you invoke `gcov` with multiple input files, the contributions from each input file are summed. Typically you would invoke it with the same list of files as the final link of your executable.

The '`.gcov`' files contain the '`:`' separated fields along with program source code. The format is

```
execution_count:line_number:source line text
```

Additional block information may succeed each line, when requested by command line option. The *execution_count* is '`-`' for lines containing no code. Unexecuted lines are marked '`#####`' or '`====`', depending on whether they are reachable by non-exceptional paths or only exceptional paths such as C++ exception handlers, respectively.

Some lines of information at the start have *line_number* of zero. These preamble lines are of the form

```
-:0:tag:value
```

The ordering and number of these preamble lines will be augmented as `gcov` development progresses — do not rely on them remaining unchanged. Use *tag* to locate a particular preamble line.

The additional block information is of the form

```
tag information
```

The *information* is human readable, but designed to be simple enough for machine parsing too.

When printing percentages, 0% and 100% are only printed when the values are *exactly* 0% and 100% respectively. Other values which would conventionally be rounded to 0% or 100% are instead printed as the nearest non-boundary value.

When using `gcov`, you must first compile your program with two special GCC options: '`-fprofile-arcs -ftest-coverage`'. This tells the compiler to generate additional information needed by gcov (basically a flow graph of the program) and also includes additional code in the object files for generating the extra profiling information needed by gcov. These additional files are placed in the directory where the object file is located.

Running the program will cause profile output to be generated. For each source file compiled with '`-fprofile-arcs`', an accompanying '`.gcda`' file will be placed in the object file directory.

Running `gcov` with your program's source file names as arguments will now produce a listing of the code along with frequency of execution for each line. For example, if your program is called '`tmp.c`', this is what you see when you use the basic `gcov` facility:

```
$ gcc -fprofile-arcs -ftest-coverage tmp.c
$ a.out
$ gcov tmp.c
90.00% of 10 source lines executed in file tmp.c
Creating tmp.c.gcov.
```

The file '`tmp.c.gcov`' contains output from `gcov`. Here is a sample:

```
        -:    0:Source:tmp.c
        -:    0:Graph:tmp.gcno
        -:    0:Data:tmp.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:#include <stdio.h>
        -:    2:
        -:    3:int main (void)
        1:    4:{
        1:    5:  int i, total;
        -:    6:
        1:    7:  total = 0;
        -:    8:
       11:    9:  for (i = 0; i < 10; i++)
```

```
   10:   10:    total += i;
    -:   11:
    1:   12:  if (total != 45)
#####:   13:    printf ("Failure\n");
    -:   14:  else
    1:   15:    printf ("Success\n");
    1:   16:  return 0;
    -:   17:}
```

When you use the '`-a`' option, you will get individual block counts, and the output looks like this:

```
    -:    0:Source:tmp.c
    -:    0:Graph:tmp.gcno
    -:    0:Data:tmp.gcda
    -:    0:Runs:1
    -:    0:Programs:1
    -:    1:#include <stdio.h>
    -:    2:
    -:    3:int main (void)
    1:    4:{
    1:    4-block  0
    1:    5:  int i, total;
    -:    6:
    1:    7:  total = 0;
    -:    8:
   11:    9:  for (i = 0; i < 10; i++)
   11:    9-block  0
   10:   10:    total += i;
   10:   10-block  0
    -:   11:
    1:   12:  if (total != 45)
    1:   12-block  0
#####:   13:    printf ("Failure\n");
$$$$$:   13-block  0
    -:   14:  else
    1:   15:    printf ("Success\n");
    1:   15-block  0
    1:   16:  return 0;
    1:   16-block  0
    -:   17:}
```

In this mode, each basic block is only shown on one line – the last line of the block. A multi-line block will only contribute to the execution count of that last line, and other lines will not be shown to contain code, unless previous blocks end on those lines. The total execution count of a line is shown and subsequent lines show the execution counts for individual blocks that end on that line. After each block, the branch and call counts of the block will be shown, if the '`-b`' option is given.

Because of the way GCC instruments calls, a call count can be shown after a line with no individual blocks. As you can see, line 13 contains a basic block that was not executed.

When you use the '`-b`' option, your output looks like this:

```
$ gcov -b tmp.c
90.00% of 10 source lines executed in file tmp.c
80.00% of 5 branches executed in file tmp.c
80.00% of 5 branches taken at least once in file tmp.c
50.00% of 2 calls executed in file tmp.c
Creating tmp.c.gcov.
```

Here is a sample of a resulting '`tmp.c.gcov`' file:

```
            -:    0:Source:tmp.c
            -:    0:Graph:tmp.gcno
            -:    0:Data:tmp.gcda
            -:    0:Runs:1
            -:    0:Programs:1
            -:    1:#include <stdio.h>
            -:    2:
            -:    3:int main (void)
function main called 1 returned 1 blocks executed 75%
            1:    4:{
            1:    5:  int i, total;
            -:    6:
            1:    7:  total = 0;
            -:    8:
           11:    9:  for (i = 0; i < 10; i++)
branch  0 taken 91% (fallthrough)
branch  1 taken 9%
           10:   10:    total += i;
            -:   11:
            1:   12:  if (total != 45)
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
    #####:   13:    printf ("Failure\n");
call    0 never executed
            -:   14:  else
            1:   15:    printf ("Success\n");
call    0 called 1 returned 100%
            1:   16:  return 0;
            -:   17:}
```

For each function, a line is printed showing how many times the function is called, how many times it returns and what percentage of the function's blocks were executed.

For each basic block, a line is printed after the last line of the basic block describing the branch or call that ends the basic block. There can be multiple branches and calls listed for a single source line if there are multiple basic blocks that end on that line. In this case, the branches and calls are each given a number. There is no simple way to map these branches and calls back to source constructs. In general, though, the lowest numbered branch or call will correspond to the leftmost construct on the source line.

For a branch, if it was executed at least once, then a percentage indicating the number of times the branch was taken divided by the number of times the branch was executed will be printed. Otherwise, the message "never executed" is printed.

For a call, if it was executed at least once, then a percentage indicating the number of times the call returned divided by the number of times the call was executed will be printed. This will usually be 100%, but may be less for functions that call `exit` or `longjmp`, and thus may not return every time they are called.

The execution counts are cumulative. If the example program were executed again without removing the '`.gcda`' file, the count for the number of times each line in the source was executed would be added to the results of the previous run(s). This is potentially useful in several ways. For example, it could be used to accumulate data over a number of program runs as part of a test verification suite, or to provide more accurate long-term information over a large number of program runs.

The data in the '.gcda' files is saved immediately before the program exits. For each source file compiled with '-fprofile-arcs', the profiling code first attempts to read in an existing '.gcda' file; if the file doesn't match the executable (differing number of basic block counts) it will ignore the contents of the file. It then adds in the new execution counts and finally writes the data to the file.

10.3 Using gcov with GCC Optimization

If you plan to use gcov to help optimize your code, you must first compile your program with two special GCC options: '-fprofile-arcs -ftest-coverage'. Aside from that, you can use any other GCC options; but if you want to prove that every single line in your program was executed, you should not compile with optimization at the same time. On some machines the optimizer can eliminate some simple code lines by combining them with other lines. For example, code like this:

```
if (a != b)
  c = 1;
else
  c = 0;
```

can be compiled into one instruction on some machines. In this case, there is no way for gcov to calculate separate execution counts for each line because there isn't separate code for each line. Hence the gcov output looks like this if you compiled the program with optimization:

```
100:   12:if (a != b)
100:   13:  c = 1;
100:   14:else
100:   15:  c = 0;
```

The output shows that this block of code, combined by optimization, executed 100 times. In one sense this result is correct, because there was only one instruction representing all four of these lines. However, the output does not indicate how many times the result was 0 and how many times the result was 1.

Inlineable functions can create unexpected line counts. Line counts are shown for the source code of the inlineable function, but what is shown depends on where the function is inlined, or if it is not inlined at all.

If the function is not inlined, the compiler must emit an out of line copy of the function, in any object file that needs it. If 'fileA.o' and 'fileB.o' both contain out of line bodies of a particular inlineable function, they will also both contain coverage counts for that function. When 'fileA.o' and 'fileB.o' are linked together, the linker will, on many systems, select one of those out of line bodies for all calls to that function, and remove or ignore the other. Unfortunately, it will not remove the coverage counters for the unused function body. Hence when instrumented, all but one use of that function will show zero counts.

If the function is inlined in several places, the block structure in each location might not be the same. For instance, a condition might now be calculable at compile time in some instances. Because the coverage of all the uses of the inline function will be shown for the same source lines, the line counts themselves might seem inconsistent.

10.4 Brief description of gcov data files

`gcov` uses two files for profiling. The names of these files are derived from the original *object* file by substituting the file suffix with either '`.gcno`', or '`.gcda`'. All of these files are placed in the same directory as the object file, and contain data stored in a platform-independent format.

The '`.gcno`' file is generated when the source file is compiled with the GCC '`-ftest-coverage`' option. It contains information to reconstruct the basic block graphs and assign source line numbers to blocks.

The '`.gcda`' file is generated when a program containing object files built with the GCC '`-fprofile-arcs`' option is executed. A separate '`.gcda`' file is created for each object file compiled with this option. It contains arc transition counts, and some summary information.

The full details of the file format is specified in '`gcov-io.h`', and functions provided in that header file should be used to access the coverage files.

10.5 Data file relocation to support cross-profiling

Running the program will cause profile output to be generated. For each source file compiled with '`-fprofile-arcs`', an accompanying '`.gcda`' file will be placed in the object file directory. That implicitly requires running the program on the same system as it was built or having the same absolute directory structure on the target system. The program will try to create the needed directory structure, if it is not already present.

To support cross-profiling, a program compiled with '`-fprofile-arcs`' can relocate the data files based on two environment variables:

- GCOV_PREFIX contains the prefix to add to the absolute paths in the object file. Prefix can be absolute, or relative. The default is no prefix.
- GCOV_PREFIX_STRIP indicates the how many initial directory names to strip off the hardwired absolute paths. Default value is 0.

  Note: If GCOV_PREFIX_STRIP is set without GCOV_PREFIX is undefined, then a relative path is made out of the hardwired absolute paths.

For example, if the object file '`/user/build/foo.o`' was built with '`-fprofile-arcs`', the final executable will try to create the data file '`/user/build/foo.gcda`' when running on the target system. This will fail if the corresponding directory does not exist and it is unable to create it. This can be overcome by, for example, setting the environment as '`GCOV_PREFIX=/target/run`' and '`GCOV_PREFIX_STRIP=1`'. Such a setting will name the data file '`/target/run/build/foo.gcda`'.

You must move the data files to the expected directory tree in order to use them for profile directed optimizations ('`--use-profile`'), or to use the `gcov` tool.

11 Known Causes of Trouble with GCC

This section describes known problems that affect users of GCC. Most of these are not GCC bugs per se—if they were, we would fix them. But the result for a user may be like the result of a bug.

Some of these problems are due to bugs in other software, some are missing features that are too much work to add, and some are places where people's opinions differ as to what is best.

11.1 Actual Bugs We Haven't Fixed Yet

- The `fixincludes` script interacts badly with automounters; if the directory of system header files is automounted, it tends to be unmounted while `fixincludes` is running. This would seem to be a bug in the automounter. We don't know any good way to work around it.

11.2 Cross-Compiler Problems

You may run into problems with cross compilation on certain machines, for several reasons.

- At present, the program '`mips-tfile`' which adds debug support to object files on Tru64 UNIX systems does not work in a cross compile environment.

11.3 Interoperation

This section lists various difficulties encountered in using GCC together with other compilers or with the assemblers, linkers, libraries and debuggers on certain systems.

- On many platforms, GCC supports a different ABI for C++ than do other compilers, so the object files compiled by GCC cannot be used with object files generated by another C++ compiler.

  An area where the difference is most apparent is name mangling. The use of different name mangling is intentional, to protect you from more subtle problems. Compilers differ as to many internal details of C++ implementation, including: how class instances are laid out, how multiple inheritance is implemented, and how virtual function calls are handled. If the name encoding were made the same, your programs would link against libraries provided from other compilers—but the programs would then crash when run. Incompatible libraries are then detected at link time, rather than at run time.
- On some BSD systems, including some versions of Ultrix, use of profiling causes static variable destructors (currently used only in C++) not to be run.
- On some SGI systems, when you use '`-lgl_s`' as an option, it gets translated magically to '`-lgl_s -lX11_s -lc_s`'. Naturally, this does not happen when you use GCC. You must specify all three options explicitly.
- On a SPARC, GCC aligns all values of type `double` on an 8-byte boundary, and it expects every `double` to be so aligned. The Sun compiler usually gives `double` values 8-byte alignment, with one exception: function arguments of type `double` may not be aligned.

As a result, if a function compiled with Sun CC takes the address of an argument of type `double` and passes this pointer of type `double *` to a function compiled with GCC, dereferencing the pointer may cause a fatal signal.

One way to solve this problem is to compile your entire program with GCC. Another solution is to modify the function that is compiled with Sun CC to copy the argument into a local variable; local variables are always properly aligned. A third solution is to modify the function that uses the pointer to dereference it via the following function `access_double` instead of directly with '`*`':

```
inline double
access_double (double *unaligned_ptr)
{
  union d2i { double d; int i[2]; };

  union d2i *p = (union d2i *) unaligned_ptr;
  union d2i u;

  u.i[0] = p->i[0];
  u.i[1] = p->i[1];

  return u.d;
}
```

Storing into the pointer can be done likewise with the same union.

- On Solaris, the `malloc` function in the '`libmalloc.a`' library may allocate memory that is only 4 byte aligned. Since GCC on the SPARC assumes that doubles are 8 byte aligned, this may result in a fatal signal if doubles are stored in memory allocated by the '`libmalloc.a`' library.

  The solution is to not use the '`libmalloc.a`' library. Use instead `malloc` and related functions from '`libc.a`'; they do not have this problem.

- On the HP PA machine, ADB sometimes fails to work on functions compiled with GCC. Specifically, it fails to work on functions that use `alloca` or variable-size arrays. This is because GCC doesn't generate HP-UX unwind descriptors for such functions. It may even be impossible to generate them.

- Debugging ('`-g`') is not supported on the HP PA machine, unless you use the preliminary GNU tools.

- Taking the address of a label may generate errors from the HP-UX PA assembler. GAS for the PA does not have this problem.

- Using floating point parameters for indirect calls to static functions will not work when using the HP assembler. There simply is no way for GCC to specify what registers hold arguments for static functions when using the HP assembler. GAS for the PA does not have this problem.

- In extremely rare cases involving some very large functions you may receive errors from the HP linker complaining about an out of bounds unconditional branch offset. This used to occur more often in previous versions of GCC, but is now exceptionally rare. If you should run into it, you can work around by making your function smaller.

- GCC compiled code sometimes emits warnings from the HP-UX assembler of the form:

  ```
  (warning) Use of GR3 when
    frame >= 8192 may cause conflict.
  ```

These warnings are harmless and can be safely ignored.

- In extremely rare cases involving some very large functions you may receive errors from the AIX Assembler complaining about a displacement that is too large. If you should run into it, you can work around by making your function smaller.
- The '`libstdc++.a`' library in GCC relies on the SVR4 dynamic linker semantics which merges global symbols between libraries and applications, especially necessary for C++ streams functionality. This is not the default behavior of AIX shared libraries and dynamic linking. '`libstdc++.a`' is built on AIX with "runtime-linking" enabled so that symbol merging can occur. To utilize this feature, the application linked with '`libstdc++.a`' must include the '`-Wl,-brtl`' flag on the link line. G++ cannot impose this because this option may interfere with the semantics of the user program and users may not always use '`g++`' to link his or her application. Applications are not required to use the '`-Wl,-brtl`' flag on the link line—the rest of the '`libstdc++.a`' library which is not dependent on the symbol merging semantics will continue to function correctly.
- An application can interpose its own definition of functions for functions invoked by '`libstdc++.a`' with "runtime-linking" enabled on AIX. To accomplish this the application must be linked with "runtime-linking" option and the functions explicitly must be exported by the application ('`-Wl,-brtl,-bE:exportfile`').
- AIX on the RS/6000 provides support (NLS) for environments outside of the United States. Compilers and assemblers use NLS to support locale-specific representations of various objects including floating-point numbers ('`.`' vs '`,`' for separating decimal fractions). There have been problems reported where the library linked with GCC does not produce the same floating-point formats that the assembler accepts. If you have this problem, set the `LANG` environment variable to '`C`' or '`En_US`'.
- Even if you specify '`-fdollars-in-identifiers`', you cannot successfully use '`$`' in identifiers on the RS/6000 due to a restriction in the IBM assembler. GAS supports these identifiers.

11.4 Incompatibilities of GCC

There are several noteworthy incompatibilities between GNU C and K&R (non-ISO) versions of C.

- GCC normally makes string constants read-only. If several identical-looking string constants are used, GCC stores only one copy of the string.

  One consequence is that you cannot call `mktemp` with a string constant argument. The function `mktemp` always alters the string its argument points to.

  Another consequence is that `sscanf` does not work on some very old systems when passed a string constant as its format control string or input. This is because `sscanf` incorrectly tries to write into the string constant. Likewise `fscanf` and `scanf`.

  The solution to these problems is to change the program to use `char`-array variables with initialization strings for these purposes instead of string constants.

- `-2147483648` is positive.

  This is because 2147483648 cannot fit in the type `int`, so (following the ISO C rules) its data type is `unsigned long int`. Negating this value yields 2147483648 again.

- GCC does not substitute macro arguments when they appear inside of string constants. For example, the following macro in GCC

  ```
  #define foo(a) "a"
  ```

  will produce output `"a"` regardless of what the argument *a* is.

- When you use `setjmp` and `longjmp`, the only automatic variables guaranteed to remain valid are those declared `volatile`. This is a consequence of automatic register allocation. Consider this function:

  ```
  jmp_buf j;

  foo ()
  {
    int a, b;

    a = fun1 ();
    if (setjmp (j))
      return a;

    a = fun2 ();
    /* longjmp (j) may occur in fun3. */
    return a + fun3 ();
  }
  ```

  Here `a` may or may not be restored to its first value when the `longjmp` occurs. If `a` is allocated in a register, then its first value is restored; otherwise, it keeps the last value stored in it.

  If you use the '`-W`' option with the '`-O`' option, you will get a warning when GCC thinks such a problem might be possible.

- Programs that use preprocessing directives in the middle of macro arguments do not work with GCC. For example, a program like this will not work:

  ```
  foobar (
  #define luser
          hack)
  ```

  ISO C does not permit such a construct.

- K&R compilers allow comments to cross over an inclusion boundary (i.e. started in an include file and ended in the including file).

- Declarations of external variables and functions within a block apply only to the block containing the declaration. In other words, they have the same scope as any other declaration in the same place.

  In some other C compilers, an `extern` declaration affects all the rest of the file even if it happens within a block.

- In traditional C, you can combine `long`, etc., with a typedef name, as shown here:

  ```
  typedef int foo;
  typedef long foo bar;
  ```

  In ISO C, this is not allowed: `long` and other type modifiers require an explicit `int`.

- PCC allows typedef names to be used as function parameters.

- Traditional C allows the following erroneous pair of declarations to appear together in a given scope:

  ```
  typedef int foo;
  typedef foo foo;
  ```

- GCC treats all characters of identifiers as significant. According to K&R-1 (2.2), "No more than the first eight characters are significant, although more may be used.". Also according to K&R-1 (2.2), "An identifier is a sequence of letters and digits; the first character must be a letter. The underscore _ counts as a letter.", but GCC also allows dollar signs in identifiers.
- PCC allows whitespace in the middle of compound assignment operators such as '`+=`'. GCC, following the ISO standard, does not allow this.
- GCC complains about unterminated character constants inside of preprocessing conditionals that fail. Some programs have English comments enclosed in conditionals that are guaranteed to fail; if these comments contain apostrophes, GCC will probably report an error. For example, this code would produce an error:

  ```
  #if 0
  You can't expect this to work.
  #endif
  ```

  The best solution to such a problem is to put the text into an actual C comment delimited by '`/*...*/`'.
- Many user programs contain the declaration '`long time ();`'. In the past, the system header files on many systems did not actually declare `time`, so it did not matter what type your program declared it to return. But in systems with ISO C headers, `time` is declared to return `time_t`, and if that is not the same as `long`, then '`long time ();`' is erroneous.

  The solution is to change your program to use appropriate system headers (`<time.h>` on systems with ISO C headers) and not to declare `time` if the system header files declare it, or failing that to use `time_t` as the return type of `time`.
- When compiling functions that return `float`, PCC converts it to a double. GCC actually returns a `float`. If you are concerned with PCC compatibility, you should declare your functions to return `double`; you might as well say what you mean.
- When compiling functions that return structures or unions, GCC output code normally uses a method different from that used on most versions of Unix. As a result, code compiled with GCC cannot call a structure-returning function compiled with PCC, and vice versa.

  The method used by GCC is as follows: a structure or union which is 1, 2, 4 or 8 bytes long is returned like a scalar. A structure or union with any other size is stored into an address supplied by the caller (usually in a special, fixed register, but on some machines it is passed on the stack). The target hook `TARGET_STRUCT_VALUE_RTX` tells GCC where to pass this address.

  By contrast, PCC on most target machines returns structures and unions of any size by copying the data into an area of static storage, and then returning the address of that storage as if it were a pointer value. The caller must copy the data from that memory area to the place where the value is wanted. GCC does not use this method because it is slower and nonreentrant.

  On some newer machines, PCC uses a reentrant convention for all structure and union returning. GCC on most of these machines uses a compatible convention when returning structures and unions in memory, but still returns small structures and unions in registers.

You can tell GCC to use a compatible convention for all structure and union returning with the option '`-fpcc-struct-return`'.

- GCC complains about program fragments such as '`0x74ae-0x4000`' which appear to be two hexadecimal constants separated by the minus operator. Actually, this string is a single *preprocessing token*. Each such token must correspond to one token in C. Since this does not, GCC prints an error message. Although it may appear obvious that what is meant is an operator and two values, the ISO C standard specifically requires that this be treated as erroneous.

  A *preprocessing token* is a *preprocessing number* if it begins with a digit and is followed by letters, underscores, digits, periods and '`e+`', '`e-`', '`E+`', '`E-`', '`p+`', '`p-`', '`P+`', or '`P-`' character sequences. (In strict C90 mode, the sequences '`p+`', '`p-`', '`P+`' and '`P-`' cannot appear in preprocessing numbers.)

  To make the above program fragment valid, place whitespace in front of the minus sign. This whitespace will end the preprocessing number.

11.5 Fixed Header Files

GCC needs to install corrected versions of some system header files. This is because most target systems have some header files that won't work with GCC unless they are changed. Some have bugs, some are incompatible with ISO C, and some depend on special features of other compilers.

Installing GCC automatically creates and installs the fixed header files, by running a program called `fixincludes`. Normally, you don't need to pay attention to this. But there are cases where it doesn't do the right thing automatically.

- If you update the system's header files, such as by installing a new system version, the fixed header files of GCC are not automatically updated. They can be updated using the `mkheaders` script installed in '*libexecdir*`/gcc/`*target*`/`*version*`/install-tools/`'.
- On some systems, header file directories contain machine-specific symbolic links in certain places. This makes it possible to share most of the header files among hosts running the same version of the system on different machine models.

  The programs that fix the header files do not understand this special way of using symbolic links; therefore, the directory of fixed header files is good only for the machine model used to build it.

  It is possible to make separate sets of fixed header files for the different machine models, and arrange a structure of symbolic links so as to use the proper set, but you'll have to do this by hand.

11.6 Standard Libraries

GCC by itself attempts to be a conforming freestanding implementation. See Chapter 2 [Language Standards Supported by GCC], page 5, for details of what this means. Beyond the library facilities required of such an implementation, the rest of the C library is supplied by the vendor of the operating system. If that C library doesn't conform to the C standards, then your programs might get warnings (especially when using '`-Wall`') that you don't expect.

For example, the `sprintf` function on SunOS 4.1.3 returns `char *` while the C standard says that `sprintf` returns an `int`. The `fixincludes` program could make the prototype for this function match the Standard, but that would be wrong, since the function will still return `char *`.

If you need a Standard compliant library, then you need to find one, as GCC does not provide one. The GNU C library (called `glibc`) provides ISO C, POSIX, BSD, SystemV and X/Open compatibility for GNU/Linux and HURD-based GNU systems; no recent version of it supports other systems, though some very old versions did. Version 2.2 of the GNU C library includes nearly complete C99 support. You could also ask your operating system vendor if newer libraries are available.

11.7 Disappointments and Misunderstandings

These problems are perhaps regrettable, but we don't know any practical way around them.

- Certain local variables aren't recognized by debuggers when you compile with optimization.

  This occurs because sometimes GCC optimizes the variable out of existence. There is no way to tell the debugger how to compute the value such a variable "would have had", and it is not clear that would be desirable anyway. So GCC simply does not mention the eliminated variable when it writes debugging information.

  You have to expect a certain amount of disagreement between the executable and your source code, when you use optimization.

- Users often think it is a bug when GCC reports an error for code like this:

  ```
  int foo (struct mumble *);

  struct mumble { ... };

  int foo (struct mumble *x)
  { ... }
  ```

  This code really is erroneous, because the scope of `struct mumble` in the prototype is limited to the argument list containing it. It does not refer to the `struct mumble` defined with file scope immediately below—they are two unrelated types with similar names in different scopes.

  But in the definition of `foo`, the file-scope type is used because that is available to be inherited. Thus, the definition and the prototype do not match, and you get an error.

  This behavior may seem silly, but it's what the ISO standard specifies. It is easy enough for you to make your code work by moving the definition of `struct mumble` above the prototype. It's not worth being incompatible with ISO C just to avoid an error for the example shown above.

- Accesses to bit-fields even in volatile objects works by accessing larger objects, such as a byte or a word. You cannot rely on what size of object is accessed in order to read or write the bit-field; it may even vary for a given bit-field according to the precise usage.

  If you care about controlling the amount of memory that is accessed, use volatile but do not use bit-fields.

- GCC comes with shell scripts to fix certain known problems in system header files. They install corrected copies of various header files in a special directory where only

GCC will normally look for them. The scripts adapt to various systems by searching all the system header files for the problem cases that we know about.

If new system header files are installed, nothing automatically arranges to update the corrected header files. They can be updated using the `mkheaders` script installed in '*libexecdir*`/gcc/`*target*`/`*version*`/install-tools/`'.

- On 68000 and x86 systems, for instance, you can get paradoxical results if you test the precise values of floating point numbers. For example, you can find that a floating point value which is not a NaN is not equal to itself. This results from the fact that the floating point registers hold a few more bits of precision than fit in a `double` in memory. Compiled code moves values between memory and floating point registers at its convenience, and moving them into memory truncates them.

  You can partially avoid this problem by using the '`-ffloat-store`' option (see Section 3.10 [Optimize Options], page 92).
- On AIX and other platforms without weak symbol support, templates need to be instantiated explicitly and symbols for static members of templates will not be generated.
- On AIX, GCC scans object files and library archives for static constructors and destructors when linking an application before the linker prunes unreferenced symbols. This is necessary to prevent the AIX linker from mistakenly assuming that static constructor or destructor are unused and removing them before the scanning can occur. All static constructors and destructors found will be referenced even though the modules in which they occur may not be used by the program. This may lead to both increased executable size and unexpected symbol references.

11.8 Common Misunderstandings with GNU C++

C++ is a complex language and an evolving one, and its standard definition (the ISO C++ standard) was only recently completed. As a result, your C++ compiler may occasionally surprise you, even when its behavior is correct. This section discusses some areas that frequently give rise to questions of this sort.

11.8.1 Declare *and* Define Static Members

When a class has static data members, it is not enough to *declare* the static member; you must also *define* it. For example:

```
class Foo
{
  ...
  void method();
  static int bar;
};
```

This declaration only establishes that the class `Foo` has an `int` named `Foo::bar`, and a member function named `Foo::method`. But you still need to define *both* `method` and `bar` elsewhere. According to the ISO standard, you must supply an initializer in one (and only one) source file, such as:

```
int Foo::bar = 0;
```

Other C++ compilers may not correctly implement the standard behavior. As a result, when you switch to `g++` from one of these compilers, you may discover that a program

that appeared to work correctly in fact does not conform to the standard: g++ reports as undefined symbols any static data members that lack definitions.

11.8.2 Name lookup, templates, and accessing members of base classes

The C++ standard prescribes that all names that are not dependent on template parameters are bound to their present definitions when parsing a template function or class.[1] Only names that are dependent are looked up at the point of instantiation. For example, consider

```
void foo(double);

struct A {
  template <typename T>
  void f () {
    foo (1);        // 1
    int i = N;      // 2
    T t;
    t.bar();        // 3
    foo (t);        // 4
  }

  static const int N;
};
```

Here, the names foo and N appear in a context that does not depend on the type of T. The compiler will thus require that they are defined in the context of use in the template, not only before the point of instantiation, and will here use ::foo(double) and A::N, respectively. In particular, it will convert the integer value to a double when passing it to ::foo(double).

Conversely, bar and the call to foo in the fourth marked line are used in contexts that do depend on the type of T, so they are only looked up at the point of instantiation, and you can provide declarations for them after declaring the template, but before instantiating it. In particular, if you instantiate A::f<int>, the last line will call an overloaded ::foo(int) if one was provided, even if after the declaration of struct A.

This distinction between lookup of dependent and non-dependent names is called two-stage (or dependent) name lookup. G++ implements it since version 3.4.

Two-stage name lookup sometimes leads to situations with behavior different from non-template codes. The most common is probably this:

```
template <typename T> struct Base {
  int i;
};

template <typename T> struct Derived : public Base<T> {
  int get_i() { return i; }
};
```

In get_i(), i is not used in a dependent context, so the compiler will look for a name declared at the enclosing namespace scope (which is the global scope here). It will not look into the base class, since that is dependent and you may declare specializations of Base

[1] The C++ standard just uses the term "dependent" for names that depend on the type or value of template parameters. This shorter term will also be used in the rest of this section.

even after declaring `Derived`, so the compiler can't really know what `i` would refer to. If there is no global variable `i`, then you will get an error message.

In order to make it clear that you want the member of the base class, you need to defer lookup until instantiation time, at which the base class is known. For this, you need to access `i` in a dependent context, by either using `this->i` (remember that `this` is of type `Derived<T>*`, so is obviously dependent), or using `Base<T>::i`. Alternatively, `Base<T>::i` might be brought into scope by a `using`-declaration.

Another, similar example involves calling member functions of a base class:

```
template <typename T> struct Base {
    int f();
};

template <typename T> struct Derived : Base<T> {
    int g() { return f(); };
};
```

Again, the call to `f()` is not dependent on template arguments (there are no arguments that depend on the type `T`, and it is also not otherwise specified that the call should be in a dependent context). Thus a global declaration of such a function must be available, since the one in the base class is not visible until instantiation time. The compiler will consequently produce the following error message:

```
x.cc: In member function 'int Derived<T>::g()':
x.cc:6: error: there are no arguments to 'f' that depend on a template
   parameter, so a declaration of 'f' must be available
x.cc:6: error: (if you use '-fpermissive', G++ will accept your code, but
   allowing the use of an undeclared name is deprecated)
```

To make the code valid either use `this->f()`, or `Base<T>::f()`. Using the '`-fpermissive`' flag will also let the compiler accept the code, by marking all function calls for which no declaration is visible at the time of definition of the template for later lookup at instantiation time, as if it were a dependent call. We do not recommend using '`-fpermissive`' to work around invalid code, and it will also only catch cases where functions in base classes are called, not where variables in base classes are used (as in the example above).

Note that some compilers (including G++ versions prior to 3.4) get these examples wrong and accept above code without an error. Those compilers do not implement two-stage name lookup correctly.

11.8.3 Temporaries May Vanish Before You Expect

It is dangerous to use pointers or references to *portions* of a temporary object. The compiler may very well delete the object before you expect it to, leaving a pointer to garbage. The most common place where this problem crops up is in classes like string classes, especially ones that define a conversion function to type `char *` or `const char *`—which is one reason why the standard `string` class requires you to call the `c_str` member function. However, any class that returns a pointer to some internal structure is potentially subject to this problem.

For example, a program may use a function `strfunc` that returns `string` objects, and another function `charfunc` that operates on pointers to `char`:

```
string strfunc ();
```

```
void charfunc (const char *);

void
f ()
{
  const char *p = strfunc().c_str();
  ...
  charfunc (p);
  ...
  charfunc (p);
}
```

In this situation, it may seem reasonable to save a pointer to the C string returned by the `c_str` member function and use that rather than call `c_str` repeatedly. However, the temporary string created by the call to `strfunc` is destroyed after `p` is initialized, at which point `p` is left pointing to freed memory.

Code like this may run successfully under some other compilers, particularly obsolete cfront-based compilers that delete temporaries along with normal local variables. However, the GNU C++ behavior is standard-conforming, so if your program depends on late destruction of temporaries it is not portable.

The safe way to write such code is to give the temporary a name, which forces it to remain until the end of the scope of the name. For example:

```
const string& tmp = strfunc ();
charfunc (tmp.c_str ());
```

11.8.4 Implicit Copy-Assignment for Virtual Bases

When a base class is virtual, only one subobject of the base class belongs to each full object. Also, the constructors and destructors are invoked only once, and called from the most-derived class. However, such objects behave unspecified when being assigned. For example:

```
struct Base{
  char *name;
  Base(char *n) : name(strdup(n)){}
  Base& operator= (const Base& other){
   free (name);
   name = strdup (other.name);
  }
};

struct A:virtual Base{
  int val;
  A():Base("A"){}
};

struct B:virtual Base{
  int bval;
  B():Base("B"){}
};

struct Derived:public A, public B{
  Derived():Base("Derived"){}
};

void func(Derived &d1, Derived &d2)
```

```
{
  d1 = d2;
}
```

The C++ standard specifies that '`Base::Base`' is only called once when constructing or copy-constructing a Derived object. It is unspecified whether '`Base::operator=`' is called more than once when the implicit copy-assignment for Derived objects is invoked (as it is inside '`func`' in the example).

G++ implements the "intuitive" algorithm for copy-assignment: assign all direct bases, then assign all members. In that algorithm, the virtual base subobject can be encountered more than once. In the example, copying proceeds in the following order: '`val`', '`name`' (via `strdup`), '`bval`', and '`name`' again.

If application code relies on copy-assignment, a user-defined copy-assignment operator removes any uncertainties. With such an operator, the application can define whether and how the virtual base subobject is assigned.

11.9 Certain Changes We Don't Want to Make

This section lists changes that people frequently request, but which we do not make because we think GCC is better without them.

- Checking the number and type of arguments to a function which has an old-fashioned definition and no prototype.

  Such a feature would work only occasionally—only for calls that appear in the same file as the called function, following the definition. The only way to check all calls reliably is to add a prototype for the function. But adding a prototype eliminates the motivation for this feature. So the feature is not worthwhile.

- Warning about using an expression whose type is signed as a shift count.

  Shift count operands are probably signed more often than unsigned. Warning about this would cause far more annoyance than good.

- Warning about assigning a signed value to an unsigned variable.

  Such assignments must be very common; warning about them would cause more annoyance than good.

- Warning when a non-void function value is ignored.

  C contains many standard functions that return a value that most programs choose to ignore. One obvious example is `printf`. Warning about this practice only leads the defensive programmer to clutter programs with dozens of casts to `void`. Such casts are required so frequently that they become visual noise. Writing those casts becomes so automatic that they no longer convey useful information about the intentions of the programmer. For functions where the return value should never be ignored, use the `warn_unused_result` function attribute (see Section 6.30 [Function Attributes], page 332).

- Making '`-fshort-enums`' the default.

  This would cause storage layout to be incompatible with most other C compilers. And it doesn't seem very important, given that you can get the same result in other ways. The case where it matters most is when the enumeration-valued object is inside a structure, and in that case you can specify a field width explicitly.

- Making bit-fields unsigned by default on particular machines where "the ABI standard" says to do so.

  The ISO C standard leaves it up to the implementation whether a bit-field declared plain `int` is signed or not. This in effect creates two alternative dialects of C.

  The GNU C compiler supports both dialects; you can specify the signed dialect with '`-fsigned-bitfields`' and the unsigned dialect with '`-funsigned-bitfields`'. However, this leaves open the question of which dialect to use by default.

  Currently, the preferred dialect makes plain bit-fields signed, because this is simplest. Since `int` is the same as `signed int` in every other context, it is cleanest for them to be the same in bit-fields as well.

  Some computer manufacturers have published Application Binary Interface standards which specify that plain bit-fields should be unsigned. It is a mistake, however, to say anything about this issue in an ABI. This is because the handling of plain bit-fields distinguishes two dialects of C. Both dialects are meaningful on every type of machine. Whether a particular object file was compiled using signed bit-fields or unsigned is of no concern to other object files, even if they access the same bit-fields in the same data structures.

  A given program is written in one or the other of these two dialects. The program stands a chance to work on most any machine if it is compiled with the proper dialect. It is unlikely to work at all if compiled with the wrong dialect.

  Many users appreciate the GNU C compiler because it provides an environment that is uniform across machines. These users would be inconvenienced if the compiler treated plain bit-fields differently on certain machines.

  Occasionally users write programs intended only for a particular machine type. On these occasions, the users would benefit if the GNU C compiler were to support by default the same dialect as the other compilers on that machine. But such applications are rare. And users writing a program to run on more than one type of machine cannot possibly benefit from this kind of compatibility.

  This is why GCC does and will treat plain bit-fields in the same fashion on all types of machines (by default).

  There are some arguments for making bit-fields unsigned by default on all machines. If, for example, this becomes a universal de facto standard, it would make sense for GCC to go along with it. This is something to be considered in the future.

  (Of course, users strongly concerned about portability should indicate explicitly in each bit-field whether it is signed or not. In this way, they write programs which have the same meaning in both C dialects.)

- Undefining `__STDC__` when '`-ansi`' is not used.

  Currently, GCC defines `__STDC__` unconditionally. This provides good results in practice.

  Programmers normally use conditionals on `__STDC__` to ask whether it is safe to use certain features of ISO C, such as function prototypes or ISO token concatenation. Since plain `gcc` supports all the features of ISO C, the correct answer to these questions is "yes".

Some users try to use `__STDC__` to check for the availability of certain library facilities. This is actually incorrect usage in an ISO C program, because the ISO C standard says that a conforming freestanding implementation should define `__STDC__` even though it does not have the library facilities. '`gcc -ansi -pedantic`' is a conforming freestanding implementation, and it is therefore required to define `__STDC__`, even though it does not come with an ISO C library.

Sometimes people say that defining `__STDC__` in a compiler that does not completely conform to the ISO C standard somehow violates the standard. This is illogical. The standard is a standard for compilers that claim to support ISO C, such as '`gcc -ansi`'—not for other compilers such as plain `gcc`. Whatever the ISO C standard says is relevant to the design of plain `gcc` without '`-ansi`' only for pragmatic reasons, not as a requirement.

GCC normally defines `__STDC__` to be 1, and in addition defines `__STRICT_ANSI__` if you specify the '`-ansi`' option, or a '`-std`' option for strict conformance to some version of ISO C. On some hosts, system include files use a different convention, where `__STDC__` is normally 0, but is 1 if the user specifies strict conformance to the C Standard. GCC follows the host convention when processing system include files, but when processing user files it follows the usual GNU C convention.

- Undefining `__STDC__` in C++.

  Programs written to compile with C++-to-C translators get the value of `__STDC__` that goes with the C compiler that is subsequently used. These programs must test `__STDC__` to determine what kind of C preprocessor that compiler uses: whether they should concatenate tokens in the ISO C fashion or in the traditional fashion.

  These programs work properly with GNU C++ if `__STDC__` is defined. They would not work otherwise.

  In addition, many header files are written to provide prototypes in ISO C but not in traditional C. Many of these header files can work without change in C++ provided `__STDC__` is defined. If `__STDC__` is not defined, they will all fail, and will all need to be changed to test explicitly for C++ as well.

- Deleting "empty" loops.

  Historically, GCC has not deleted "empty" loops under the assumption that the most likely reason you would put one in a program is to have a delay, so deleting them will not make real programs run any faster.

  However, the rationale here is that optimization of a nonempty loop cannot produce an empty one. This held for carefully written C compiled with less powerful optimizers but is not always the case for carefully written C++ or with more powerful optimizers. Thus GCC will remove operations from loops whenever it can determine those operations are not externally visible (apart from the time taken to execute them, of course). In case the loop can be proved to be finite, GCC will also remove the loop itself.

  Be aware of this when performing timing tests, for instance the following loop can be completely removed, provided `some_expression` can provably not change any global state.

```
{
  int sum = 0;
  int ix;
```

```
   for (ix = 0; ix != 10000; ix++)
      sum += some_expression;
}
```

Even though `sum` is accumulated in the loop, no use is made of that summation, so the accumulation can be removed.

- Making side effects happen in the same order as in some other compiler.

  It is never safe to depend on the order of evaluation of side effects. For example, a function call like this may very well behave differently from one compiler to another:

  ```
  void func (int, int);

  int i = 2;
  func (i++, i++);
  ```

  There is no guarantee (in either the C or the C++ standard language definitions) that the increments will be evaluated in any particular order. Either increment might happen first. `func` might get the arguments '`2, 3`', or it might get '`3, 2`', or even '`2, 2`'.

- Making certain warnings into errors by default.

  Some ISO C testsuites report failure when the compiler does not produce an error message for a certain program.

  ISO C requires a "diagnostic" message for certain kinds of invalid programs, but a warning is defined by GCC to count as a diagnostic. If GCC produces a warning but not an error, that is correct ISO C support. If testsuites call this "failure", they should be run with the GCC option '`-pedantic-errors`', which will turn these warnings into errors.

11.10 Warning Messages and Error Messages

The GNU compiler can produce two kinds of diagnostics: errors and warnings. Each kind has a different purpose:

Errors report problems that make it impossible to compile your program. GCC reports errors with the source file name and line number where the problem is apparent.

Warnings report other unusual conditions in your code that *may* indicate a problem, although compilation can (and does) proceed. Warning messages also report the source file name and line number, but include the text '`warning:`' to distinguish them from error messages.

Warnings may indicate danger points where you should check to make sure that your program really does what you intend; or the use of obsolete features; or the use of nonstandard features of GNU C or C++. Many warnings are issued only if you ask for them, with one of the '`-W`' options (for instance, '`-Wall`' requests a variety of useful warnings).

GCC always tries to compile your program if possible; it never gratuitously rejects a program whose meaning is clear merely because (for instance) it fails to conform to a standard. In some cases, however, the C and C++ standards specify that certain extensions are forbidden, and a diagnostic *must* be issued by a conforming compiler. The '`-pedantic`' option tells GCC to issue warnings in such cases; '`-pedantic-errors`' says to make them errors instead. This does not mean that *all* non-ISO constructs get warnings or errors.

See Section 3.8 [Options to Request or Suppress Warnings], page 49, for more detail on these and related command-line options.

12 Reporting Bugs

Your bug reports play an essential role in making GCC reliable.

When you encounter a problem, the first thing to do is to see if it is already known. See Chapter 11 [Trouble], page 663. If it isn't known, then you should report the problem.

12.1 Have You Found a Bug?

If you are not sure whether you have found a bug, here are some guidelines:

- If the compiler gets a fatal signal, for any input whatever, that is a compiler bug. Reliable compilers never crash.
- If the compiler produces invalid assembly code, for any input whatever (except an `asm` statement), that is a compiler bug, unless the compiler reports errors (not just warnings) which would ordinarily prevent the assembler from being run.
- If the compiler produces valid assembly code that does not correctly execute the input source code, that is a compiler bug.

  However, you must double-check to make sure, because you may have a program whose behavior is undefined, which happened by chance to give the desired results with another C or C++ compiler.

  For example, in many nonoptimizing compilers, you can write '`x;`' at the end of a function instead of '`return x;`', with the same results. But the value of the function is undefined if `return` is omitted; it is not a bug when GCC produces different results.

  Problems often result from expressions with two increment operators, as in `f (*p++, *p++)`. Your previous compiler might have interpreted that expression the way you intended; GCC might interpret it another way. Neither compiler is wrong. The bug is in your code.

  After you have localized the error to a single source line, it should be easy to check for these things. If your program is correct and well defined, you have found a compiler bug.
- If the compiler produces an error message for valid input, that is a compiler bug.
- If the compiler does not produce an error message for invalid input, that is a compiler bug. However, you should note that your idea of "invalid input" might be someone else's idea of "an extension" or "support for traditional practice".
- If you are an experienced user of one of the languages GCC supports, your suggestions for improvement of GCC are welcome in any case.

12.2 How and where to Report Bugs

Bugs should be reported to the bug database at `http://gcc.gnu.org/bugs/`.

13 How To Get Help with GCC

If you need help installing, using or changing GCC, there are two ways to find it:

- Send a message to a suitable network mailing list. First try `gcc-help@gcc.gnu.org` (for help installing or using GCC), and if that brings no response, try `gcc@gcc.gnu.org`. For help changing GCC, ask `gcc@gcc.gnu.org`. If you think you have found a bug in GCC, please report it following the instructions at see Section 12.2 [Bug Reporting], page 679.
- Look in the service directory for someone who might help you for a fee. The service directory is found at `http://www.fsf.org/resources/service`.

For further information, see `http://gcc.gnu.org/faq.html#support`.

14 Contributing to GCC Development

If you would like to help pretest GCC releases to assure they work well, current development sources are available by SVN (see `http://gcc.gnu.org/svn.html`). Source and binary snapshots are also available for FTP; see `http://gcc.gnu.org/snapshots.html`.

If you would like to work on improvements to GCC, please read the advice at these URLs:

```
http://gcc.gnu.org/contribute.html
http://gcc.gnu.org/contributewhy.html
```

for information on how to make useful contributions and avoid duplication of effort. Suggested projects are listed at `http://gcc.gnu.org/projects/`.

Funding Free Software

If you want to have more free software a few years from now, it makes sense for you to help encourage people to contribute funds for its development. The most effective approach known is to encourage commercial redistributors to donate.

Users of free software systems can boost the pace of development by encouraging for-a-fee distributors to donate part of their selling price to free software developers—the Free Software Foundation, and others.

The way to convince distributors to do this is to demand it and expect it from them. So when you compare distributors, judge them partly by how much they give to free software development. Show distributors they must compete to be the one who gives the most.

To make this approach work, you must insist on numbers that you can compare, such as, "We will donate ten dollars to the Frobnitz project for each disk sold." Don't be satisfied with a vague promise, such as "A portion of the profits are donated," since it doesn't give a basis for comparison.

Even a precise fraction "of the profits from this disk" is not very meaningful, since creative accounting and unrelated business decisions can greatly alter what fraction of the sales price counts as profit. If the price you pay is $50, ten percent of the profit is probably less than a dollar; it might be a few cents, or nothing at all.

Some redistributors do development work themselves. This is useful too; but to keep everyone honest, you need to inquire how much they do, and what kind. Some kinds of development make much more long-term difference than others. For example, maintaining a separate version of a program contributes very little; maintaining the standard version of a program for the whole community contributes much. Easy new ports contribute little, since someone else would surely do them; difficult ports such as adding a new CPU to the GNU Compiler Collection contribute more; major new features or packages contribute the most.

By establishing the idea that supporting further development is "the proper thing to do" when distributing free software for a fee, we can assure a steady flow of resources into making more free software.

Copyright © 1994 Free Software Foundation, Inc.
Verbatim copying and redistribution of this section is permitted
without royalty; alteration is not permitted.

The GNU Project and GNU/Linux

The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. (GNU is a recursive acronym for "GNU's Not Unix"; it is pronounced "guh-NEW".) Variants of the GNU operating system, which use the kernel Linux, are now widely used; though these systems are often referred to as "Linux", they are more accurately called GNU/Linux systems.

For more information, see:

`http://www.gnu.org/`
`http://www.gnu.org/gnu/linux-and-gnu.html`

GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so

available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
`http://fsf.org/`

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Contributors to GCC

The GCC project would like to thank its many contributors. Without them the project would not have been nearly as successful as it has been. Any omissions in this list are accidental. Feel free to contact `law@redhat.com` or `gerald@pfeifer.com` if you have been left out or some of your contributions are not listed. Please keep this list in alphabetical order.

- Analog Devices helped implement the support for complex data types and iterators.
- John David Anglin for threading-related fixes and improvements to libstdc++-v3, and the HP-UX port.
- James van Artsdalen wrote the code that makes efficient use of the Intel 80387 register stack.
- Abramo and Roberto Bagnara for the SysV68 Motorola 3300 Delta Series port.
- Alasdair Baird for various bug fixes.
- Giovanni Bajo for analyzing lots of complicated C++ problem reports.
- Peter Barada for his work to improve code generation for new ColdFire cores.
- Gerald Baumgartner added the signature extension to the C++ front end.
- Godmar Back for his Java improvements and encouragement.
- Scott Bambrough for help porting the Java compiler.
- Wolfgang Bangerth for processing tons of bug reports.
- Jon Beniston for his Microsoft Windows port of Java and port to Lattice Mico32.
- Daniel Berlin for better DWARF2 support, faster/better optimizations, improved alias analysis, plus migrating GCC to Bugzilla.
- Geoff Berry for his Java object serialization work and various patches.
- David Binderman tests weekly snapshots of GCC trunk against Fedora Rawhide for several architectures.
- Uros Bizjak for the implementation of x87 math built-in functions and for various middle end and i386 back end improvements and bug fixes.
- Eric Blake for helping to make GCJ and libgcj conform to the specifications.
- Janne Blomqvist for contributions to GNU Fortran.
- Segher Boessenkool for various fixes.
- Hans-J. Boehm for his garbage collector, IA-64 libffi port, and other Java work.
- Neil Booth for work on cpplib, lang hooks, debug hooks and other miscellaneous clean-ups.
- Steven Bosscher for integrating the GNU Fortran front end into GCC and for contributing to the tree-ssa branch.
- Eric Botcazou for fixing middle- and backend bugs left and right.
- Per Bothner for his direction via the steering committee and various improvements to the infrastructure for supporting new languages. Chill front end implementation. Initial implementations of cpplib, fix-header, config.guess, libio, and past C++ library (libg++) maintainer. Dreaming up, designing and implementing much of GCJ.

- Devon Bowen helped port GCC to the Tahoe.
- Don Bowman for mips-vxworks contributions.
- Dave Brolley for work on cpplib and Chill.
- Paul Brook for work on the ARM architecture and maintaining GNU Fortran.
- Robert Brown implemented the support for Encore 32000 systems.
- Christian Bruel for improvements to local store elimination.
- Herman A.J. ten Brugge for various fixes.
- Joerg Brunsmann for Java compiler hacking and help with the GCJ FAQ.
- Joe Buck for his direction via the steering committee.
- Craig Burley for leadership of the G77 Fortran effort.
- Stephan Buys for contributing Doxygen notes for libstdc++.
- Paolo Carlini for libstdc++ work: lots of efficiency improvements to the C++ strings, streambufs and formatted I/O, hard detective work on the frustrating localization issues, and keeping up with the problem reports.
- John Carr for his alias work, SPARC hacking, infrastructure improvements, previous contributions to the steering committee, loop optimizations, etc.
- Stephane Carrez for 68HC11 and 68HC12 ports.
- Steve Chamberlain for support for the Renesas SH and H8 processors and the PicoJava processor, and for GCJ config fixes.
- Glenn Chambers for help with the GCJ FAQ.
- John-Marc Chandonia for various libgcj patches.
- Denis Chertykov for contributing and maintaining the AVR port, the first GCC port for an 8-bit architecture.
- Scott Christley for his Objective-C contributions.
- Eric Christopher for his Java porting help and clean-ups.
- Branko Cibej for more warning contributions.
- The GNU Classpath project for all of their merged runtime code.
- Nick Clifton for arm, mcore, fr30, v850, m32r, rx work, '`--help`', and other random hacking.
- Michael Cook for libstdc++ cleanup patches to reduce warnings.
- R. Kelley Cook for making GCC buildable from a read-only directory as well as other miscellaneous build process and documentation clean-ups.
- Ralf Corsepius for SH testing and minor bug fixing.
- Stan Cox for care and feeding of the x86 port and lots of behind the scenes hacking.
- Alex Crain provided changes for the 3b1.
- Ian Dall for major improvements to the NS32k port.
- Paul Dale for his work to add uClinux platform support to the m68k backend.
- Dario Dariol contributed the four varieties of sample programs that print a copy of their source.
- Russell Davidson for fstream and stringstream fixes in libstdc++.

- Bud Davis for work on the G77 and GNU Fortran compilers.
- Mo DeJong for GCJ and libgcj bug fixes.
- DJ Delorie for the DJGPP port, build and libiberty maintenance, various bug fixes, and the M32C, MeP, and RL78 ports.
- Arnaud Desitter for helping to debug GNU Fortran.
- Gabriel Dos Reis for contributions to G++, contributions and maintenance of GCC diagnostics infrastructure, libstdc++-v3, including `valarray<>`, `complex<>`, maintaining the numerics library (including that pesky `<limits>` :-) and keeping up-to-date anything to do with numbers.
- Ulrich Drepper for his work on glibc, testing of GCC using glibc, ISO C99 support, CFG dumping support, etc., plus support of the C++ runtime libraries including for all kinds of C interface issues, contributing and maintaining `complex<>`, sanity checking and disbursement, configuration architecture, libio maintenance, and early math work.
- Zdenek Dvorak for a new loop unroller and various fixes.
- Michael Eager for his work on the Xilinx MicroBlaze port.
- Richard Earnshaw for his ongoing work with the ARM.
- David Edelsohn for his direction via the steering committee, ongoing work with the RS6000/PowerPC port, help cleaning up Haifa loop changes, doing the entire AIX port of libstdc++ with his bare hands, and for ensuring GCC properly keeps working on AIX.
- Kevin Ediger for the floating point formatting of num_put::do_put in libstdc++.
- Phil Edwards for libstdc++ work including configuration hackery, documentation maintainer, chief breaker of the web pages, the occasional iostream bug fix, and work on shared library symbol versioning.
- Paul Eggert for random hacking all over GCC.
- Mark Elbrecht for various DJGPP improvements, and for libstdc++ configuration support for locales and fstream-related fixes.
- Vadim Egorov for libstdc++ fixes in strings, streambufs, and iostreams.
- Christian Ehrhardt for dealing with bug reports.
- Ben Elliston for his work to move the Objective-C runtime into its own subdirectory and for his work on autoconf.
- Revital Eres for work on the PowerPC 750CL port.
- Marc Espie for OpenBSD support.
- Doug Evans for much of the global optimization framework, arc, m32r, and SPARC work.
- Christopher Faylor for his work on the Cygwin port and for caring and feeding the gcc.gnu.org box and saving its users tons of spam.
- Fred Fish for BeOS support and Ada fixes.
- Ivan Fontes Garcia for the Portuguese translation of the GCJ FAQ.
- Peter Gerwinski for various bug fixes and the Pascal front end.
- Kaveh R. Ghazi for his direction via the steering committee, amazing work to make '`-W -Wall -W* -Werror`' useful, and continuously testing GCC on a plethora of platforms.

Kaveh extends his gratitude to the CAIP Center at Rutgers University for providing him with computing resources to work on Free Software since the late 1980s.

- John Gilmore for a donation to the FSF earmarked improving GNU Java.
- Judy Goldberg for c++ contributions.
- Torbjorn Granlund for various fixes and the c-torture testsuite, multiply- and divide-by-constant optimization, improved long long support, improved leaf function register allocation, and his direction via the steering committee.
- Anthony Green for his '`-Os`' contributions, the moxie port, and Java front end work.
- Stu Grossman for gdb hacking, allowing GCJ developers to debug Java code.
- Michael K. Gschwind contributed the port to the PDP-11.
- Richard Guenther for his ongoing middle-end contributions and bug fixes and for release management.
- Ron Guilmette implemented the `protoize` and `unprotoize` tools, the support for Dwarf symbolic debugging information, and much of the support for System V Release 4. He has also worked heavily on the Intel 386 and 860 support.
- Sumanth Gundapaneni for contributing the CR16 port.
- Mostafa Hagog for Swing Modulo Scheduling (SMS) and post reload GCSE.
- Bruno Haible for improvements in the runtime overhead for EH, new warnings and assorted bug fixes.
- Andrew Haley for his amazing Java compiler and library efforts.
- Chris Hanson assisted in making GCC work on HP-UX for the 9000 series 300.
- Michael Hayes for various thankless work he's done trying to get the c30/c40 ports functional. Lots of loop and unroll improvements and fixes.
- Dara Hazeghi for wading through myriads of target-specific bug reports.
- Kate Hedstrom for staking the G77 folks with an initial testsuite.
- Richard Henderson for his ongoing SPARC, alpha, ia32, and ia64 work, loop opts, and generally fixing lots of old problems we've ignored for years, flow rewrite and lots of further stuff, including reviewing tons of patches.
- Aldy Hernandez for working on the PowerPC port, SIMD support, and various fixes.
- Nobuyuki Hikichi of Software Research Associates, Tokyo, contributed the support for the Sony NEWS machine.
- Kazu Hirata for caring and feeding the Renesas H8/300 port and various fixes.
- Katherine Holcomb for work on GNU Fortran.
- Manfred Hollstein for his ongoing work to keep the m88k alive, lots of testing and bug fixing, particularly of GCC configury code.
- Steve Holmgren for MachTen patches.
- Mat Hostetter for work on the TILE-Gx and TILEPro ports.
- Jan Hubicka for his x86 port improvements.
- Falk Hueffner for working on C and optimization bug reports.
- Bernardo Innocenti for his m68k work, including merging of ColdFire improvements and uClinux support.

- Christian Iseli for various bug fixes.
- Kamil Iskra for general m68k hacking.
- Lee Iverson for random fixes and MIPS testing.
- Andreas Jaeger for testing and benchmarking of GCC and various bug fixes.
- Jakub Jelinek for his SPARC work and sibling call optimizations as well as lots of bug fixes and test cases, and for improving the Java build system.
- Janis Johnson for ia64 testing and fixes, her quality improvement sidetracks, and web page maintenance.
- Kean Johnston for SCO OpenServer support and various fixes.
- Tim Josling for the sample language treelang based originally on Richard Kenner's "toy" language.
- Nicolai Josuttis for additional libstdc++ documentation.
- Klaus Kaempf for his ongoing work to make alpha-vms a viable target.
- Steven G. Kargl for work on GNU Fortran.
- David Kashtan of SRI adapted GCC to VMS.
- Ryszard Kabatek for many, many libstdc++ bug fixes and optimizations of strings, especially member functions, and for auto_ptr fixes.
- Geoffrey Keating for his ongoing work to make the PPC work for GNU/Linux and his automatic regression tester.
- Brendan Kehoe for his ongoing work with G++ and for a lot of early work in just about every part of libstdc++.
- Oliver M. Kellogg of Deutsche Aerospace contributed the port to the MIL-STD-1750A.
- Richard Kenner of the New York University Ultracomputer Research Laboratory wrote the machine descriptions for the AMD 29000, the DEC Alpha, the IBM RT PC, and the IBM RS/6000 as well as the support for instruction attributes. He also made changes to better support RISC processors including changes to common subexpression elimination, strength reduction, function calling sequence handling, and condition code support, in addition to generalizing the code for frame pointer elimination and delay slot scheduling. Richard Kenner was also the head maintainer of GCC for several years.
- Mumit Khan for various contributions to the Cygwin and Mingw32 ports and maintaining binary releases for Microsoft Windows hosts, and for massive libstdc++ porting work to Cygwin/Mingw32.
- Robin Kirkham for cpu32 support.
- Mark Klein for PA improvements.
- Thomas Koenig for various bug fixes.
- Bruce Korb for the new and improved fixincludes code.
- Benjamin Kosnik for his G++ work and for leading the libstdc++-v3 effort.
- Charles LaBrec contributed the support for the Integrated Solutions 68020 system.
- Asher Langton and Mike Kumbera for contributing Cray pointer support to GNU Fortran, and for other GNU Fortran improvements.
- Jeff Law for his direction via the steering committee, coordinating the entire egcs project and GCC 2.95, rolling out snapshots and releases, handling merges from GCC2,

reviewing tons of patches that might have fallen through the cracks else, and random but extensive hacking.

- Walter Lee for work on the TILE-Gx and TILEPro ports.
- Marc Lehmann for his direction via the steering committee and helping with analysis and improvements of x86 performance.
- Victor Leikehman for work on GNU Fortran.
- Ted Lemon wrote parts of the RTL reader and printer.
- Kriang Lerdsuwanakij for C++ improvements including template as template parameter support, and many C++ fixes.
- Warren Levy for tremendous work on libgcj (Java Runtime Library) and random work on the Java front end.
- Alain Lichnewsky ported GCC to the MIPS CPU.
- Oskar Liljeblad for hacking on AWT and his many Java bug reports and patches.
- Robert Lipe for OpenServer support, new testsuites, testing, etc.
- Chen Liqin for various S+core related fixes/improvement, and for maintaining the S+core port.
- Weiwen Liu for testing and various bug fixes.
- Manuel López-Ibáñez for improving '`-Wconversion`' and many other diagnostics fixes and improvements.
- Dave Love for his ongoing work with the Fortran front end and runtime libraries.
- Martin von Löwis for internal consistency checking infrastructure, various C++ improvements including namespace support, and tons of assistance with libstdc++/compiler merges.
- H.J. Lu for his previous contributions to the steering committee, many x86 bug reports, prototype patches, and keeping the GNU/Linux ports working.
- Greg McGary for random fixes and (someday) bounded pointers.
- Andrew MacLeod for his ongoing work in building a real EH system, various code generation improvements, work on the global optimizer, etc.
- Vladimir Makarov for hacking some ugly i960 problems, PowerPC hacking improvements to compile-time performance, overall knowledge and direction in the area of instruction scheduling, and design and implementation of the automaton based instruction scheduler.
- Bob Manson for his behind the scenes work on dejagnu.
- Philip Martin for lots of libstdc++ string and vector iterator fixes and improvements, and string clean up and testsuites.
- All of the Mauve project contributors, for Java test code.
- Bryce McKinlay for numerous GCJ and libgcj fixes and improvements.
- Adam Megacz for his work on the Microsoft Windows port of GCJ.
- Michael Meissner for LRS framework, ia32, m32r, v850, m88k, MIPS, powerpc, haifa, ECOFF debug support, and other assorted hacking.
- Jason Merrill for his direction via the steering committee and leading the G++ effort.

- Martin Michlmayr for testing GCC on several architectures using the entire Debian archive.
- David Miller for his direction via the steering committee, lots of SPARC work, improvements in jump.c and interfacing with the Linux kernel developers.
- Gary Miller ported GCC to Charles River Data Systems machines.
- Alfred Minarik for libstdc++ string and ios bug fixes, and turning the entire libstdc++ testsuite namespace-compatible.
- Mark Mitchell for his direction via the steering committee, mountains of C++ work, load/store hoisting out of loops, alias analysis improvements, ISO C `restrict` support, and serving as release manager for GCC 3.x.
- Alan Modra for various GNU/Linux bits and testing.
- Toon Moene for his direction via the steering committee, Fortran maintenance, and his ongoing work to make us make Fortran run fast.
- Jason Molenda for major help in the care and feeding of all the services on the gcc.gnu.org (formerly egcs.cygnus.com) machine—mail, web services, ftp services, etc etc. Doing all this work on scrap paper and the backs of envelopes would have been... difficult.
- Catherine Moore for fixing various ugly problems we have sent her way, including the haifa bug which was killing the Alpha & PowerPC Linux kernels.
- Mike Moreton for his various Java patches.
- David Mosberger-Tang for various Alpha improvements, and for the initial IA-64 port.
- Stephen Moshier contributed the floating point emulator that assists in cross-compilation and permits support for floating point numbers wider than 64 bits and for ISO C99 support.
- Bill Moyer for his behind the scenes work on various issues.
- Philippe De Muyter for his work on the m68k port.
- Joseph S. Myers for his work on the PDP-11 port, format checking and ISO C99 support, and continuous emphasis on (and contributions to) documentation.
- Nathan Myers for his work on libstdc++-v3: architecture and authorship through the first three snapshots, including implementation of locale infrastructure, string, shadow C headers, and the initial project documentation (DESIGN, CHECKLIST, and so forth). Later, more work on MT-safe string and shadow headers.
- Felix Natter for documentation on porting libstdc++.
- Nathanael Nerode for cleaning up the configuration/build process.
- NeXT, Inc. donated the front end that supports the Objective-C language.
- Hans-Peter Nilsson for the CRIS and MMIX ports, improvements to the search engine setup, various documentation fixes and other small fixes.
- Geoff Noer for his work on getting cygwin native builds working.
- Diego Novillo for his work on Tree SSA, OpenMP, SPEC performance tracking web pages, GIMPLE tuples, and assorted fixes.
- David O'Brien for the FreeBSD/alpha, FreeBSD/AMD x86-64, FreeBSD/ARM, FreeBSD/PowerPC, and FreeBSD/SPARC64 ports and related infrastructure improvements.

- Alexandre Oliva for various build infrastructure improvements, scripts and amazing testing work, including keeping libtool issues sane and happy.
- Stefan Olsson for work on mt_alloc.
- Melissa O'Neill for various NeXT fixes.
- Rainer Orth for random MIPS work, including improvements to GCC's o32 ABI support, improvements to dejagnu's MIPS support, Java configuration clean-ups and porting work, and maintaining the IRIX, Solaris 2, and Tru64 UNIX ports.
- Hartmut Penner for work on the s390 port.
- Paul Petersen wrote the machine description for the Alliant FX/8.
- Alexandre Petit-Bianco for implementing much of the Java compiler and continued Java maintainership.
- Matthias Pfaller for major improvements to the NS32k port.
- Gerald Pfeifer for his direction via the steering committee, pointing out lots of problems we need to solve, maintenance of the web pages, and taking care of documentation maintenance in general.
- Andrew Pinski for processing bug reports by the dozen.
- Ovidiu Predescu for his work on the Objective-C front end and runtime libraries.
- Jerry Quinn for major performance improvements in C++ formatted I/O.
- Ken Raeburn for various improvements to checker, MIPS ports and various cleanups in the compiler.
- Rolf W. Rasmussen for hacking on AWT.
- David Reese of Sun Microsystems contributed to the Solaris on PowerPC port.
- Volker Reichelt for keeping up with the problem reports.
- Joern Rennecke for maintaining the sh port, loop, regmove & reload hacking and developing and maintaining the Epiphany port.
- Loren J. Rittle for improvements to libstdc++-v3 including the FreeBSD port, threading fixes, thread-related configury changes, critical threading documentation, and solutions to really tricky I/O problems, as well as keeping GCC properly working on FreeBSD and continuous testing.
- Craig Rodrigues for processing tons of bug reports.
- Ola Rönnerup for work on mt_alloc.
- Gavin Romig-Koch for lots of behind the scenes MIPS work.
- David Ronis inspired and encouraged Craig to rewrite the G77 documentation in texinfo format by contributing a first pass at a translation of the old '`g77-0.5.16/f/DOC`' file.
- Ken Rose for fixes to GCC's delay slot filling code.
- Paul Rubin wrote most of the preprocessor.
- Pétur Runólfsson for major performance improvements in C++ formatted I/O and large file support in C++ filebuf.
- Chip Salzenberg for libstdc++ patches and improvements to locales, traits, Makefiles, libio, libtool hackery, and "long long" support.
- Juha Sarlin for improvements to the H8 code generator.

- Greg Satz assisted in making GCC work on HP-UX for the 9000 series 300.
- Roger Sayle for improvements to constant folding and GCC's RTL optimizers as well as for fixing numerous bugs.
- Bradley Schatz for his work on the GCJ FAQ.
- Peter Schauer wrote the code to allow debugging to work on the Alpha.
- William Schelter did most of the work on the Intel 80386 support.
- Tobias Schlüter for work on GNU Fortran.
- Bernd Schmidt for various code generation improvements and major work in the reload pass, serving as release manager for GCC 2.95.3, and work on the Blackfin and C6X ports.
- Peter Schmid for constant testing of libstdc++—especially application testing, going above and beyond what was requested for the release criteria—and libstdc++ header file tweaks.
- Jason Schroeder for jcf-dump patches.
- Andreas Schwab for his work on the m68k port.
- Lars Segerlund for work on GNU Fortran.
- Dodji Seketeli for numerous C++ bug fixes and debug info improvements.
- Joel Sherrill for his direction via the steering committee, RTEMS contributions and RTEMS testing.
- Nathan Sidwell for many C++ fixes/improvements.
- Jeffrey Siegal for helping RMS with the original design of GCC, some code which handles the parse tree and RTL data structures, constant folding and help with the original VAX & m68k ports.
- Kenny Simpson for prompting libstdc++ fixes due to defect reports from the LWG (thereby keeping GCC in line with updates from the ISO).
- Franz Sirl for his ongoing work with making the PPC port stable for GNU/Linux.
- Andrey Slepuhin for assorted AIX hacking.
- Trevor Smigiel for contributing the SPU port.
- Christopher Smith did the port for Convex machines.
- Danny Smith for his major efforts on the Mingw (and Cygwin) ports.
- Randy Smith finished the Sun FPA support.
- Scott Snyder for queue, iterator, istream, and string fixes and libstdc++ testsuite entries. Also for providing the patch to G77 to add rudimentary support for `INTEGER*1`, `INTEGER*2`, and `LOGICAL*1`.
- Zdenek Sojka for running automated regression testing of GCC and reporting numerous bugs.
- Jayant Sonar for contributing the CR16 port.
- Brad Spencer for contributions to the GLIBCPP_FORCE_NEW technique.
- Richard Stallman, for writing the original GCC and launching the GNU project.
- Jan Stein of the Chalmers Computer Society provided support for Genix, as well as part of the 32000 machine description.

- Nigel Stephens for various mips16 related fixes/improvements.
- Jonathan Stone wrote the machine description for the Pyramid computer.
- Graham Stott for various infrastructure improvements.
- John Stracke for his Java HTTP protocol fixes.
- Mike Stump for his Elxsi port, G++ contributions over the years and more recently his vxworks contributions
- Jeff Sturm for Java porting help, bug fixes, and encouragement.
- Shigeya Suzuki for this fixes for the bsdi platforms.
- Ian Lance Taylor for the Go frontend, the initial mips16 and mips64 support, general configury hacking, fixincludes, etc.
- Holger Teutsch provided the support for the Clipper CPU.
- Gary Thomas for his ongoing work to make the PPC work for GNU/Linux.
- Philipp Thomas for random bug fixes throughout the compiler
- Jason Thorpe for thread support in libstdc++ on NetBSD.
- Kresten Krab Thorup wrote the run time support for the Objective-C language and the fantastic Java bytecode interpreter.
- Michael Tiemann for random bug fixes, the first instruction scheduler, initial C++ support, function integration, NS32k, SPARC and M88k machine description work, delay slot scheduling.
- Andreas Tobler for his work porting libgcj to Darwin.
- Teemu Torma for thread safe exception handling support.
- Leonard Tower wrote parts of the parser, RTL generator, and RTL definitions, and of the VAX machine description.
- Daniel Towner and Hariharan Sandanagobalane contributed and maintain the picoChip port.
- Tom Tromey for internationalization support and for his many Java contributions and libgcj maintainership.
- Lassi Tuura for improvements to config.guess to determine HP processor types.
- Petter Urkedal for libstdc++ CXXFLAGS, math, and algorithms fixes.
- Andy Vaught for the design and initial implementation of the GNU Fortran front end.
- Brent Verner for work with the libstdc++ cshadow files and their associated configure steps.
- Todd Vierling for contributions for NetBSD ports.
- Jonathan Wakely for contributing libstdc++ Doxygen notes and XHTML guidance.
- Dean Wakerley for converting the install documentation from HTML to texinfo in time for GCC 3.0.
- Krister Walfridsson for random bug fixes.
- Feng Wang for contributions to GNU Fortran.
- Stephen M. Webb for time and effort on making libstdc++ shadow files work with the tricky Solaris 8+ headers, and for pushing the build-time header tree.

- John Wehle for various improvements for the x86 code generator, related infrastructure improvements to help x86 code generation, value range propagation and other work, WE32k port.
- Ulrich Weigand for work on the s390 port.
- Zack Weinberg for major work on cpplib and various other bug fixes.
- Matt Welsh for help with Linux Threads support in GCJ.
- Urban Widmark for help fixing java.io.
- Mark Wielaard for new Java library code and his work integrating with Classpath.
- Dale Wiles helped port GCC to the Tahoe.
- Bob Wilson from Tensilica, Inc. for the Xtensa port.
- Jim Wilson for his direction via the steering committee, tackling hard problems in various places that nobody else wanted to work on, strength reduction and other loop optimizations.
- Paul Woegerer and Tal Agmon for the CRX port.
- Carlo Wood for various fixes.
- Tom Wood for work on the m88k port.
- Canqun Yang for work on GNU Fortran.
- Masanobu Yuhara of Fujitsu Laboratories implemented the machine description for the Tron architecture (specifically, the Gmicro).
- Kevin Zachmann helped port GCC to the Tahoe.
- Ayal Zaks for Swing Modulo Scheduling (SMS).
- Xiaoqiang Zhang for work on GNU Fortran.
- Gilles Zunino for help porting Java to Irix.

The following people are recognized for their contributions to GNAT, the Ada front end of GCC:

- Bernard Banner
- Romain Berrendonner
- Geert Bosch
- Emmanuel Briot
- Joel Brobecker
- Ben Brosgol
- Vincent Celier
- Arnaud Charlet
- Chien Chieng
- Cyrille Comar
- Cyrille Crozes
- Robert Dewar
- Gary Dismukes
- Robert Duff

- Ed Falis
- Ramon Fernandez
- Sam Figueroa
- Vasiliy Fofanov
- Michael Friess
- Franco Gasperoni
- Ted Giering
- Matthew Gingell
- Laurent Guerby
- Jerome Guitton
- Olivier Hainque
- Jerome Hugues
- Hristian Kirtchev
- Jerome Lambourg
- Bruno Leclerc
- Albert Lee
- Sean McNeil
- Javier Miranda
- Laurent Nana
- Pascal Obry
- Dong-Ik Oh
- Laurent Pautet
- Brett Porter
- Thomas Quinot
- Nicolas Roche
- Pat Rogers
- Jose Ruiz
- Douglas Rupp
- Sergey Rybin
- Gail Schenker
- Ed Schonberg
- Nicolas Setton
- Samuel Tardieu

The following people are recognized for their contributions of new features, bug reports, testing and integration of classpath/libgcj for GCC version 4.1:

- Lillian Angel for `JTree` implementation and lots Free Swing additions and bug fixes.
- Wolfgang Baer for `GapContent` bug fixes.
- Anthony Balkissoon for `JList`, Free Swing 1.5 updates and mouse event fixes, lots of Free Swing work including `JTable` editing.

- Stuart Ballard for RMI constant fixes.
- Goffredo Baroncelli for `HTTPURLConnection` fixes.
- Gary Benson for `MessageFormat` fixes.
- Daniel Bonniot for `Serialization` fixes.
- Chris Burdess for lots of gnu.xml and http protocol fixes, `StAX` and `DOM` `xml:id` support.
- Ka-Hing Cheung for `TreePath` and `TreeSelection` fixes.
- Archie Cobbs for build fixes, VM interface updates, `URLClassLoader` updates.
- Kelley Cook for build fixes.
- Martin Cordova for Suggestions for better `SocketTimeoutException`.
- David Daney for `BitSet` bug fixes, `HttpURLConnection` rewrite and improvements.
- Thomas Fitzsimmons for lots of upgrades to the gtk+ AWT and Cairo 2D support. Lots of imageio framework additions, lots of AWT and Free Swing bug fixes.
- Jeroen Frijters for `ClassLoader` and nio cleanups, serialization fixes, better `Proxy` support, bug fixes and IKVM integration.
- Santiago Gala for `AccessControlContext` fixes.
- Nicolas Geoffray for `VMClassLoader` and `AccessController` improvements.
- David Gilbert for `basic` and `metal` icon and plaf support and lots of documenting, Lots of Free Swing and metal theme additions. `MetalIconFactory` implementation.
- Anthony Green for `MIDI` framework, `ALSA` and `DSSI` providers.
- Andrew Haley for `Serialization` and `URLClassLoader` fixes, gcj build speedups.
- Kim Ho for `JFileChooser` implementation.
- Andrew John Hughes for `Locale` and net fixes, URI RFC2986 updates, `Serialization` fixes, `Properties` XML support and generic branch work, VMIntegration guide update.
- Bastiaan Huisman for `TimeZone` bug fixing.
- Andreas Jaeger for mprec updates.
- Paul Jenner for better '`-Werror`' support.
- Ito Kazumitsu for `NetworkInterface` implementation and updates.
- Roman Kennke for `BoxLayout`, `GrayFilter` and `SplitPane`, plus bug fixes all over. Lots of Free Swing work including styled text.
- Simon Kitching for `String` cleanups and optimization suggestions.
- Michael Koch for configuration fixes, `Locale` updates, bug and build fixes.
- Guilhem Lavaux for configuration, thread and channel fixes and Kaffe integration. JCL native `Pointer` updates. Logger bug fixes.
- David Lichteblau for JCL support library global/local reference cleanups.
- Aaron Luchko for JDWP updates and documentation fixes.
- Ziga Mahkovec for `Graphics2D` upgraded to Cairo 0.5 and new regex features.
- Sven de Marothy for BMP imageio support, CSS and `TextLayout` fixes. `GtkImage` rewrite, 2D, awt, free swing and date/time fixes and implementing the Qt4 peers.
- Casey Marshall for crypto algorithm fixes, `FileChannel` lock, `SystemLogger` and `FileHandler` rotate implementations, NIO `FileChannel.map` support, security and policy updates.

- Bryce McKinlay for RMI work.
- Audrius Meskauskas for lots of Free Corba, RMI and HTML work plus testing and documenting.
- Kalle Olavi Niemitalo for build fixes.
- Rainer Orth for build fixes.
- Andrew Overholt for `File` locking fixes.
- Ingo Proetel for `Image`, `Logger` and `URLClassLoader` updates.
- Olga Rodimina for `MenuSelectionManager` implementation.
- Jan Roehrich for `BasicTreeUI` and `JTree` fixes.
- Julian Scheid for documentation updates and gjdoc support.
- Christian Schlichtherle for zip fixes and cleanups.
- Robert Schuster for documentation updates and beans fixes, `TreeNode` enumerations and `ActionCommand` and various fixes, XML and URL, AWT and Free Swing bug fixes.
- Keith Seitz for lots of JDWP work.
- Christian Thalinger for 64-bit cleanups, Configuration and VM interface fixes and `CACAO` integration, `fdlibm` updates.
- Gael Thomas for `VMClassLoader` boot packages support suggestions.
- Andreas Tobler for Darwin and Solaris testing and fixing, `Qt4` support for Darwin/OS X, `Graphics2D` support, `gtk+` updates.
- Dalibor Topic for better `DEBUG` support, build cleanups and Kaffe integration. `Qt4` build infrastructure, `SHA1PRNG` and `GdkPixbugDecoder` updates.
- Tom Tromey for Eclipse integration, generics work, lots of bug fixes and gcj integration including coordinating The Big Merge.
- Mark Wielaard for bug fixes, packaging and release management, `Clipboard` implementation, system call interrupts and network timeouts and `GdkPixpufDecoder` fixes.

In addition to the above, all of which also contributed time and energy in testing GCC, we would like to thank the following for their contributions to testing:

- Michael Abd-El-Malek
- Thomas Arend
- Bonzo Armstrong
- Steven Ashe
- Chris Baldwin
- David Billinghurst
- Jim Blandy
- Stephane Bortzmeyer
- Horst von Brand
- Frank Braun
- Rodney Brown
- Sidney Cadot
- Bradford Castalia

- Robert Clark
- Jonathan Corbet
- Ralph Doncaster
- Richard Emberson
- Levente Farkas
- Graham Fawcett
- Mark Fernyhough
- Robert A. French
- Jörgen Freyh
- Mark K. Gardner
- Charles-Antoine Gauthier
- Yung Shing Gene
- David Gilbert
- Simon Gornall
- Fred Gray
- John Griffin
- Patrik Hagglund
- Phil Hargett
- Amancio Hasty
- Takafumi Hayashi
- Bryan W. Headley
- Kevin B. Hendricks
- Joep Jansen
- Christian Joensson
- Michel Kern
- David Kidd
- Tobias Kuipers
- Anand Krishnaswamy
- A. O. V. Le Blanc
- llewelly
- Damon Love
- Brad Lucier
- Matthias Klose
- Martin Knoblauch
- Rick Lutowski
- Jesse Macnish
- Stefan Morrell
- Anon A. Mous
- Matthias Mueller

- Pekka Nikander
- Rick Niles
- Jon Olson
- Magnus Persson
- Chris Pollard
- Richard Polton
- Derk Reefman
- David Rees
- Paul Reilly
- Tom Reilly
- Torsten Rueger
- Danny Sadinoff
- Marc Schifer
- Erik Schnetter
- Wayne K. Schroll
- David Schuler
- Vin Shelton
- Tim Souder
- Adam Sulmicki
- Bill Thorson
- George Talbot
- Pedro A. M. Vazquez
- Gregory Warnes
- Ian Watson
- David E. Young
- And many others

And finally we'd like to thank everyone who uses the compiler, provides feedback and generally reminds us why we're doing this work in the first place.

Option Index

GCC's command line options are indexed here without any initial '-' or '--'. Where an option has both positive and negative forms (such as '-f*option*' and '-fno-*option*'), relevant entries in the manual are indexed under the most appropriate form; it may sometimes be useful to look up both forms.

#

..... 26

-

-fno-keep-inline-dllexport 97
-mcpu 263

8

8bit-idiv 217

A

A 152
all_load 189
allowable_client 191
ansi 5, 29, 148, 433, 675
arch_errors_fatal 189
aux-info 31
avx256-split-unaligned-load 217
avx256-split-unaligned-store 217

B

B 158
Bdynamic 283
bind_at_load 189
Bstatic 283
bundle 189
bundle_loader 189

C

c 25, 154
C 153
client_name 191
compatibility_version 191
coverage 79
current_version 191

D

d 81
D 143
da 84
dA 84
dD 84, 152
dead_strip 191
dependency-file 191
dH 84
dI 152
dM 152
dN 152
dp 85
dP 85
dU 152
dumpmachine 92
dumpspecs 92
dumpversion 92
dv 85
dx 85
dylib_file 191
dylinker_install_name 191
dynamic 191
dynamiclib 189

E

E 26, 154
EB 233
EL 233
exported_symbols_list 191

F

F 188
fabi-version 35
falign-functions 116
falign-jumps 116
falign-labels 116
falign-loops 116
fassociative-math 123
fasynchronous-unwind-tables 286
fauto-inc-dec 100
fbounds-check 285
fbranch-probabilities 125
fbranch-target-load-optimize 127
fbranch-target-load-optimize2 127
fbtr-bb-exclusive 127
fcall-saved 289
fcall-used 289
fcaller-saves 105
fcheck-data-deps 110
fcheck-new 35
fcombine-stack-adjustments 105
fcommon 367

fcompare-debug ... 75
fcompare-debug-second ... 76
fcompare-elim ... 121
fcond-mismatch ... 34
fconserve-space ... 36
fconserve-stack ... 105
fconstant-string-class ... 45
fconstexpr-depth ... 36
fcprop-registers ... 121
fcrossjumping ... 100
fcse-follow-jumps ... 99
fcse-skip-blocks ... 99
fcx-fortran-rules ... 125
fcx-limited-range ... 125
fdata-sections ... 127
fdbg-cnt ... 79
fdbg-cnt-list ... 79
fdce ... 100
fdebug-cpp ... 151
fdebug-prefix-map ... 77
fdebug-types-section ... 73
fdeduce-init-list ... 36
fdelayed-branch ... 102
fdelete-null-pointer-checks ... 101
fdevirtualize ... 101
fdiagnostics-show-location ... 48
fdiagnostics-show-option ... 49
fdirectives-only ... 150
fdisable- ... 80
fdollars-in-identifiers ... 150, 665
fdse ... 101
fdump-class-hierarchy ... 85
fdump-final-insns ... 75
fdump-ipa ... 85
fdump-noaddr ... 85
fdump-passes ... 86
fdump-rtl-alignments ... 81
fdump-rtl-all ... 84
fdump-rtl-asmcons ... 81
fdump-rtl-auto_inc_dec ... 81
fdump-rtl-barriers ... 81
fdump-rtl-bbpart ... 81
fdump-rtl-bbro ... 81
fdump-rtl-btl2 ... 81
fdump-rtl-bypass ... 81
fdump-rtl-ce1 ... 82
fdump-rtl-ce2 ... 82
fdump-rtl-ce3 ... 82
fdump-rtl-combine ... 81
fdump-rtl-compgotos ... 81
fdump-rtl-cprop_hardreg ... 82
fdump-rtl-csa ... 82
fdump-rtl-cse1 ... 82
fdump-rtl-cse2 ... 82
fdump-rtl-dbr ... 82
fdump-rtl-dce ... 82
fdump-rtl-dce1 ... 82
fdump-rtl-dce2 ... 82
fdump-rtl-dfinish ... 84
fdump-rtl-dfinit ... 84
fdump-rtl-eh ... 82
fdump-rtl-eh_ranges ... 82
fdump-rtl-expand ... 82
fdump-rtl-fwprop1 ... 82
fdump-rtl-fwprop2 ... 82
fdump-rtl-gcse1 ... 82
fdump-rtl-gcse2 ... 82
fdump-rtl-init-regs ... 82
fdump-rtl-initvals ... 82
fdump-rtl-into_cfglayout ... 82
fdump-rtl-ira ... 83
fdump-rtl-jump ... 83
fdump-rtl-loop2 ... 83
fdump-rtl-mach ... 83
fdump-rtl-mode_sw ... 83
fdump-rtl-outof_cfglayout ... 83
fdump-rtl-peephole2 ... 83
fdump-rtl-postreload ... 83
fdump-rtl-pro_and_epilogue ... 83
fdump-rtl-regclass ... 84
fdump-rtl-regmove ... 83
fdump-rtl-rnreg ... 83
fdump-rtl-sched1 ... 83
fdump-rtl-sched2 ... 83
fdump-rtl-see ... 83
fdump-rtl-seqabstr ... 83
fdump-rtl-shorten ... 83
fdump-rtl-sibling ... 83
fdump-rtl-sms ... 84
fdump-rtl-split1 ... 84
fdump-rtl-split2 ... 84
fdump-rtl-split3 ... 84
fdump-rtl-split4 ... 84
fdump-rtl-split5 ... 84
fdump-rtl-stack ... 84
fdump-rtl-subreg1 ... 84
fdump-rtl-subreg2 ... 84
fdump-rtl-subregs_of_mode_finish ... 84
fdump-rtl-subregs_of_mode_init ... 84
fdump-rtl-unshare ... 84
fdump-rtl-vartrack ... 84
fdump-rtl-vregs ... 84
fdump-rtl-web ... 84
fdump-statistics ... 86
fdump-translation-unit ... 85
fdump-tree ... 86
fdump-tree-alias ... 87
fdump-tree-all ... 88
fdump-tree-ccp ... 87
fdump-tree-cfg ... 87
fdump-tree-ch ... 87
fdump-tree-copyprop ... 87
fdump-tree-copyrename ... 88
fdump-tree-dce ... 88
fdump-tree-dom ... 88
fdump-tree-dse ... 88

fdump-tree-forwprop ... 88
fdump-tree-fre ... 87
fdump-tree-gimple ... 87
fdump-tree-mudflap ... 88
fdump-tree-nrv ... 88
fdump-tree-optimized ... 87
fdump-tree-original ... 87
fdump-tree-phiopt ... 88
fdump-tree-pre ... 87
fdump-tree-sink ... 88
fdump-tree-slp ... 88
fdump-tree-sra ... 88
fdump-tree-ssa ... 87
fdump-tree-store_copyprop ... 88
fdump-tree-storeccp ... 87
fdump-tree-vcg ... 87
fdump-tree-vect ... 88
fdump-tree-vrp ... 88
fdump-unnumbered ... 85
fdump-unnumbered-links ... 85
fdwarf2-cfi-asm ... 77
fearly-inlining ... 96
feliminate-dwarf2-dups ... 76
feliminate-unused-debug-symbols ... 73
feliminate-unused-debug-types ... 92
fenable- ... 80
fexceptions ... 286
fexcess-precision ... 122
fexec-charset ... 151
fexpensive-optimizations ... 101
fextended-identifiers ... 150
ffast-math ... 123
ffat-lto-objects ... 121
ffinite-math-only ... 124
ffix-and-continue ... 189
ffixed ... 289
ffloat-store ... 122, 670
ffor-scope ... 37
fforward-propagate ... 95
ffp-contract ... 95
ffreestanding ... 6, 33, 52, 339
ffriend-injection ... 36
ffunction-sections ... 127
fgcse ... 99
fgcse-after-reload ... 100
fgcse-las ... 100
fgcse-lm ... 100
fgcse-sm ... 100
fgnu-runtime ... 45
fgnu-tm ... 33
fgnu89-inline ... 31
fgraphite-identity ... 109
fhosted ... 33
fif-conversion ... 101
fif-conversion2 ... 101
filelist ... 191
findirect-data ... 189
findirect-inlining ... 96
finhibit-size-directive ... 288
finline-functions ... 96
finline-functions-called-once ... 96
finline-limit ... 97
finline-small-functions ... 96
finput-charset ... 151
finstrument-functions ... 290, 347
finstrument-functions-exclude-file-list ... 290
finstrument-functions-exclude-function-list ... 291
fipa-cp ... 106
fipa-cp-clone ... 107
fipa-matrix-reorg ... 107
fipa-profile ... 106
fipa-pta ... 106
fipa-pure-const ... 106
fipa-reference ... 106
fipa-sra ... 96
fira-loop-pressure ... 102
fira-verbose ... 102
fivopts ... 111
fkeep-inline-functions ... 97, 381
fkeep-static-consts ... 97
flat_namespace ... 191
flax-vector-conversions ... 34
fleading-underscore ... 292
floop-block ... 109
floop-flatten ... 110
floop-interchange ... 108
floop-parallelize-all ... 110
floop-strip-mine ... 109
flto ... 117
flto-partition ... 120
fmax-errors ... 49
fmem-report ... 78
fmerge-all-constants ... 98
fmerge-constants ... 97
fmerge-debug-strings ... 77
fmessage-length ... 48
fmodulo-sched ... 98
fmodulo-sched-allow-regmoves ... 98
fmove-loop-invariants ... 127
fms-extensions ... 33, 37, 617
fmudflap ... 98
fmudflapir ... 98
fmudflapth ... 98
fnext-runtime ... 45
fno-access-control ... 35
fno-asm ... 32
fno-branch-count-reg ... 98
fno-builtin ... 32, 52, 339, 433
fno-common ... 287, 367
fno-compare-debug ... 75
fno-debug-types-section ... 73
fno-default-inline ... 40, 95, 381
fno-defer-pop ... 95
fno-diagnostics-show-option ... 49

fno-dwarf2-cfi-asm 77
fno-elide-constructors 36
fno-enforce-eh-specs 37
fno-for-scope 37
fno-function-cse 98
fno-gnu-keywords 37
fno-guess-branch-probability 113
fno-ident 288
fno-implement-inlines 37, 626
fno-implicit-inline-templates 37
fno-implicit-templates 37, 628
fno-inline 96
fno-ira-share-save-slots 102
fno-ira-share-spill-slots 102
fno-jump-tables 289
fno-math-errno 123
fno-merge-debug-strings 77
fno-nil-receivers 45
fno-nonansi-builtins 37
fno-operator-names 38
fno-optional-diags 38
fno-peephole 113
fno-peephole2 113
fno-pretty-templates 38
fno-rtti 38
fno-sched-interblock 103
fno-sched-spec 103
fno-set-stack-executable 218
fno-show-column 152
fno-signed-bitfields 35
fno-signed-zeros 124
fno-stack-limit 291
fno-threadsafe-statics 39
fno-toplevel-reorder 117
fno-trapping-math 124
fno-unsigned-bitfields 35
fno-use-cxa-get-exception-ptr 39
fno-var-tracking-assignments 91
fno-var-tracking-assignments-toggle 91
fno-weak 40
fno-working-directory 151
fno-zero-initialized-in-bss 98
fnon-call-exceptions 286
fnothrow-opt 37
fobjc-abi-version 45
fobjc-call-cxx-cdtors 45
fobjc-direct-dispatch 46
fobjc-exceptions 46
fobjc-gc 46
fobjc-nilcheck 46
fobjc-std 46
fomit-frame-pointer 95
fopenmp 33
foptimize-register-move 101
foptimize-sibling-calls 96
force_cpusubtype_ALL 190
force_flat_namespace 191
fpack-struct 289
fpartial-inlining 113
fpcc-struct-return 286, 667
fpch-deps 147
fpch-preprocess 147
fpeel-loops 126
fpermissive 38
fpic 288
fPIC 288
fpie 289
fPIE 289
fplan9-extensions 617
fpost-ipa-mem-report 78
fpre-ipa-mem-report 78
fpredictive-commoning 113
fprefetch-loop-arrays 113
fpreprocessed 150
fprofile-arcs 78, 437
fprofile-correction 121
fprofile-dir 121
fprofile-generate 121
fprofile-use 122
fprofile-values 126
fpu 263
frandom-seed 89
freciprocal-math 124
frecord-gcc-switches 288
free 101
freg-struct-return 287
fregmove 101
frename-registers 126
freorder-blocks 114
freorder-blocks-and-partition 114
freorder-functions 114
freplace-objc-classes 47
frepo 38, 627
frerun-cse-after-loop 99
freschedule-modulo-scheduled-loops 105
frounding-math 124
fsched-critical-path-heuristic 104
fsched-dep-count-heuristic 105
fsched-group-heuristic 104
fsched-last-insn-heuristic 104
fsched-pressure 103
fsched-rank-heuristic 104
fsched-spec-insn-heuristic 104
fsched-spec-load 103
fsched-spec-load-dangerous 103
fsched-stalled-insns 103
fsched-stalled-insns-dep 104
fsched-verbose 89
fsched2-use-superblocks 104
fschedule-insns 103
fschedule-insns2 103
fsection-anchors 128
fsel-sched-pipelining 105
fsel-sched-pipelining-outer-loops 105
fselective-scheduling 105
fselective-scheduling2 105

`fshort-double` ... 287
`fshort-enums` ... 287, 303, 376, 674
`fshort-wchar` ... 287
`fshrink-wrap` ... 105
`fsignaling-nans` ... 125
`fsigned-bitfields` ... 35, 675
`fsigned-char` ... 34, 300
`fsingle-precision-constant` ... 125
`fsplit-ivs-in-unroller` ... 113
`fsplit-stack` ... 292, 347
`fsplit-wide-types` ... 99
`fstack-check` ... 291
`fstack-limit-register` ... 291
`fstack-limit-symbol` ... 291
`fstack-protector` ... 127
`fstack-protector-all` ... 127
`fstack-usage` ... 78
`fstats` ... 38
`fstrict-aliasing` ... 114
`fstrict-enums` ... 38
`fstrict-overflow` ... 115
`fstrict-volatile-bitfields` ... 293
`fsyntax-only` ... 49
`ftabstop` ... 151
`ftemplate-depth` ... 39
`ftest-coverage` ... 79
`fthread-jumps` ... 99
`ftime-report` ... 78
`ftls-model` ... 292
`ftracer` ... 112, 126
`ftrack-macro-expansion` ... 151
`ftrapv` ... 285
`ftree-bit-ccp` ... 107
`ftree-builtin-call-dce` ... 107
`ftree-ccp` ... 107
`ftree-ch` ... 108
`ftree-copy-prop` ... 106
`ftree-copyrename` ... 112
`ftree-dce` ... 107
`ftree-dominator-opts` ... 108
`ftree-dse` ... 108
`ftree-forwprop` ... 106
`ftree-fre` ... 106
`ftree-loop-im` ... 111
`ftree-loop-ivcanon` ... 111
`ftree-loop-linear` ... 108
`ftree-loop-optimize` ... 108
`ftree-parallelize-loops` ... 111
`ftree-phiprop` ... 106
`ftree-pre` ... 106
`ftree-pta` ... 111
`ftree-reassoc` ... 106
`ftree-sink` ... 107
`ftree-slp-vectorize` ... 112
`ftree-sra` ... 112
`ftree-ter` ... 112
`ftree-vect-loop-version` ... 112
`ftree-vectorize` ... 112
`ftree-vectorizer-verbose` ... 88
`ftree-vrp` ... 112
`funit-at-a-time` ... 117
`funroll-all-loops` ... 113, 126
`funroll-loops` ... 112, 126
`funsafe-loop-optimizations` ... 100
`funsafe-math-optimizations` ... 123
`funsigned-bitfields` ... 35, 303, 675
`funsigned-char` ... 34, 300
`funswitch-loops` ... 127
`funwind-tables` ... 286
`fuse-cxa-atexit` ... 39
`fvar-tracking` ... 90
`fvar-tracking-assignments` ... 91
`fvar-tracking-assignments-toggle` ... 91
`fvariable-expansion-in-unroller` ... 113
`fvect-cost-model` ... 112
`fverbose-asm` ... 288
`fvisibility` ... 292
`fvisibility-inlines-hidden` ... 39
`fvisibility-ms-compat` ... 40
`fvpt` ... 126
`fweb` ... 117
`fwhole-program` ... 117
`fwide-exec-charset` ... 151
`fworking-directory` ... 151
`fwrapv` ... 286
`fzero-link` ... 47

G

`g` ... 72
`G` ... 224, 238, 260, 281
`gcoff` ... 74
`gdwarf-`*`version`* ... 74
`gen-decls` ... 47
`gfull` ... 189
`ggdb` ... 73
`gno-record-gcc-switches` ... 74
`gno-strict-dwarf` ... 74
`grecord-gcc-switches` ... 74
`gstabs` ... 73
`gstabs+` ... 73
`gstrict-dwarf` ... 74
`gtoggle` ... 75
`gused` ... 188
`gvms` ... 74
`gxcoff` ... 74
`gxcoff+` ... 74

H

`H` ... 153
`headerpad_max_install_names` ... 191
`help` ... 26, 153

I

I 144, 158
I- 148, 159
idirafter 149
iframework 188
imacros 149
image_base 191
imultilib 150
include 149
init 191
install_name 191
iprefix 149
iquote 150, 158
isysroot 149
isystem 150
iwithprefix 149
iwithprefixbefore 149

K

keep_private_externs 191

L

l 154
L 158
lobjc 155

M

m 258
M 145
m1 269
m10 247
m128bit-long-double 209
m16-bit 187
m1reg- 169
m2 269
m210 230
m2a 269
m2a-nofpu 269
m2a-single 269
m2a-single-only 269
m3 269
m31 267
m32 ..... 217, 253, 278, 281
m32-bit 187
m32bit-doubles 263
m32r 223
m32r2 223
m32rx 223
m340 230
m3dnow 213
m3e 269
m4 269
m4-nofpu 269
m4-single 269
m4-single-only 269
m40 247
m45 247
m4a 270
m4a-nofpu 269
m4a-single 270
m4a-single-only 269
m4al 270
m4byte-functions 230
m5200 227
m5206e 227
m528x 227
m5307 227
m5407 227
m64 ..... 217, 253, 267, 278, 281
m64bit-doubles 263
m68000 226
m68010 226
m68020 226
m68020-40 227
m68020-60 227
m68030 226
m68040 226
m68060 226
m68881 227
m8-bit 187
m96bit-long-double 209
mabi 169, 215, 258
mabi=32 235
mabi=64 235
mabi=eabi 235
mabi=gnu 245
mabi=ibmlongdouble 259
mabi=ieeelongdouble 259
mabi=mmixware 245
mabi=n32 235
mabi=no-spe 259
mabi=o64 235
mabi=spe 258
mabicalls 236
mabort-on-noreturn 172
mabsdiff 231
mabshi 247
mac0 247
macc-4 198
macc-8 198
maccumulate-args 177
maccumulate-outgoing-args 215, 272
maddress-space-conversion 280
madjust-unroll 272
mads 259
maix-struct-return 258
maix32 254
maix64 254
malign-300 201
malign-double 209
malign-int 228
malign-labels 198

`malign-loops` ... 224
`malign-natural` ... 254
`malign-power` ... 254
`mall-opts` ... 231
`malloc-cc` ... 197
`malpha-as` ... 194
`maltivec` ... 252
`mam33` ... 246
`mam33-2` ... 246
`mam34` ... 246
`mandroid` ... 200
`mapcs` ... 170
`mapcs-frame` ... 169
`mapp-regs` ... 274, 282
`march` .. 171, 185, 186, 201, 203, 207, 225, 234, 267
`marm` ... 174
`mas100-syntax` ... 264
`masm=`*`dialect`* ... 208
`matomic-updates` ... 280
`mauto-pic` ... 219
`maverage` ... 231
`mavoid-indexed-addresses` ... 256
`max-vect-align` ... 169
`mb` ... 270
`mbackchain` ... 266
`mbarrel-shift-enabled` ... 222
`mbase-addresses` ... 245
`mbased=` ... 231
`mbcopy` ... 247
`mbcopy-builtin` ... 247
`mbig` ... 257
`mbig-endian` ... 170, 185, 219, 230, 257
`mbig-endian-data` ... 264
`mbig-switch` ... 201, 282
`mbigtable` ... 270
`mbionic` ... 200
`mbit-align` ... 256
`mbit-ops` ... 187
`mbitfield` ... 228
`mbitops` ... 231, 270
`mblock-move-inline-limit` ... 260
`mbranch-cheap` ... 248
`mbranch-cost` ... 167, 177, 243
`mbranch-cost=`*`num`* ... 273
`mbranch-cost=`*`number`* ... 224
`mbranch-expensive` ... 247
`mbranch-hints` ... 279
`mbranch-likely` ... 243
`mbranch-predict` ... 245
`mbss-plt` ... 252
`mbuild-constants` ... 193
`mbwx` ... 194
`mc=` ... 231
`mc68000` ... 226
`mc68020` ... 226
`mcache-size` ... 280
`mcall-eabi` ... 258
`mcall-freebsd` ... 258
`mcall-linux` ... 258
`mcall-netbsd` ... 258
`mcall-prologues` ... 177
`mcall-sysv` ... 258
`mcall-sysv-eabi` ... 258
`mcall-sysv-noeabi` ... 258
`mcallee-super-interworking` ... 174
`mcaller-super-interworking` ... 174
`mcallgraph-data` ... 230
`mcbranchdi` ... 273
`mcc-init` ... 186
`mcfv4e` ... 227
`mcheck-zero-division` ... 240
`mcirrus-fix-invalid-insns` ... 173
`mcix` ... 194
`mcld` ... 213
`mclip` ... 231
`mcmodel` ... 278
`mcmodel=kernel` ... 217
`mcmodel=large` ... 217, 252
`mcmodel=medium` ... 217, 252
`mcmodel=small` ... 217, 252
`mcmove` ... 167
`mcmpb` ... 249
`mcmpeqdi` ... 273
`mcode-readable` ... 239
`mcond-exec` ... 199
`mcond-move` ... 199
`mconfig=` ... 231
`mconsole` ... 218
`mconst-align` ... 186
`mconst16` ... 284
`mconstant-gp` ... 219
`mcop` ... 231
`mcop32` ... 231
`mcop64` ... 231
`mcorea` ... 184
`mcoreb` ... 185
`mcpu` ... 170, 186, 195, 200, 207, 225, 248, 251, 275, 281
`mcpu=` ... 182, 223, 232
`mcpu32` ... 227
`mcr16c` ... 187
`mcr16cplus` ... 187
`mcrc32` ... 214
`mcsync-anomaly` ... 183
`mcx16` ... 214
`MD` ... 147
`mdalign` ... 270
`mdata-align` ... 186
`mdata-model` ... 187
`mdc` ... 231
`mdebug` ... 224, 267
`mdebug-main=`*`prefix`* ... 196, 222
`mdec-asm` ... 248
`mdisable-callt` ... 283
`mdisable-fpregs` ... 201
`mdisable-indexing` ... 202

`mdiv` ... 228, 230, 231
`mdiv=`*strategy* ... 271
`mdivide-breaks` ... 240
`mdivide-enabled` ... 222
`mdivide-traps` ... 240
`mdivsi3_libfunc=`*name* ... 272
`mdll` ... 218
`mdlmzb` ... 256
`mdmx` ... 238
`mdouble` ... 197
`mdouble-float` ... 237, 255
`mdsp` ... 237
`mdspr2` ... 237
`mdual-nops` ... 280
`mdwarf2-asm` ... 220
`mdword` ... 197
`mdynamic-no-pic` ... 257
`mea32` ... 280
`mea64` ... 280
`meabi` ... 259
`mearly-stop-bits` ... 220
`meb` ... 231, 268
`mel` ... 231, 268
`melf` ... 187, 245
`memb` ... 259
`membedded-data` ... 239
`memregs=` ... 223
`mep` ... 282
`mepsilon` ... 245
`merror-reloc` ... 279
`mesa` ... 267
`metrax100` ... 186
`metrax4` ... 186
`mexplicit-relocs` ... 194, 240
`mextern-sdata` ... 239
`MF` ... 146
`mfast-fp` ... 184
`mfast-indirect-calls` ... 202
`mfaster-structs` ... 275
`mfdpic` ... 197
`mfentry` ... 216
`mfix` ... 194
`mfix-24k` ... 241
`mfix-and-continue` ... 189
`mfix-at697f` ... 278
`mfix-cortex-m3-ldrd` ... 175
`mfix-r10000` ... 241
`mfix-r4000` ... 241
`mfix-r4400` ... 241
`mfix-sb1` ... 242
`mfix-vr4120` ... 242
`mfix-vr4130` ... 242
`mfixed-cc` ... 197
`mfixed-range` ... 202, 220, 272, 280
`mflat` ... 274
`mflip-mips16` ... 235
`mfloat-abi` ... 170
`mfloat-gprs` ... 253
`mfloat-ieee` ... 194
`mfloat-vax` ... 194
`mfloat32` ... 247
`mfloat64` ... 247
`mflush-func` ... 243
`mflush-func=`*name* ... 224
`mflush-trap=`*number* ... 224
`mfmaf` ... 278
`mfmovd` ... 270
`mforce-no-pic` ... 284
`mfp` ... 172
`mfp-exceptions` ... 244
`mfp-mode` ... 168
`mfp-reg` ... 192
`mfp-rounding-mode` ... 192
`mfp-trap-mode` ... 192
`mfp16-format` ... 172
`mfp32` ... 237
`mfp64` ... 237
`mfpe` ... 172
`mfpmath` ... 122, 207
`mfpr-32` ... 196
`mfpr-64` ... 197
`mfprnd` ... 249
`mfpu` ... 172, 247, 255, 274
`mfriz` ... 262
`mfull-toc` ... 253
`mfused-madd` ... 220, 241, 256, 268, 273, 284
`mg` ... 283
`MG` ... 146
`mgas` ... 194, 202
`mgen-cell-microcode` ... 252
`mgettrcost=`*number* ... 272
`mglibc` ... 200
`mgnu` ... 283
`mgnu-as` ... 219
`mgnu-ld` ... 203, 219
`mgotplt` ... 187
`mgp32` ... 237
`mgp64` ... 237
`mgpopt` ... 239
`mgpr-32` ... 196
`mgpr-64` ... 196
`mgprel-ro` ... 198
`mh` ... 201
`mhalf-reg-file` ... 167
`mhard-dfp` ... 249, 266
`mhard-float` ... 197, 227, 232, 237, 255, 265, 274
`mhard-quad-float` ... 275
`mhardlit` ... 230
`mhint-max-distance` ... 281
`mhint-max-nops` ... 280
`mhitachi` ... 270
`mhp-ld` ... 203
`micplb` ... 185
`mid-shared-library` ... 183
`mieee` ... 192, 270
`mieee-conformant` ... 193

`mieee-fp` ... 208
`mieee-with-inexact` ... 192
`milp32` ... 220
`mimpure-text` ... 274
`mincoming-stack-boundary` ... 211
`mindexed-addressing` ... 272
`minline-all-stringops` ... 216
`minline-float-divide-max-throughput` ... 219
`minline-float-divide-min-latency` ... 219
`minline-ic_invalidate` ... 270
`minline-int-divide-max-throughput` ... 220
`minline-int-divide-min-latency` ... 219
`minline-plt` ... 184, 197
`minline-sqrt-max-throughput` ... 220
`minline-sqrt-min-latency` ... 220
`minline-stringops-dynamically` ... 216
`minsert-sched-nops` ... 257
`mint-register` ... 265
`mint16` ... 247
`mint32` ... 187, 201, 247
`mint8` ... 177
`minterlink-mips16` ... 235
`minvalid-symbols` ... 273
`mio-volatile` ... 231
`mips1` ... 235
`mips16` ... 235
`mips2` ... 235
`mips3` ... 235
`mips32` ... 235
`mips32r2` ... 235
`mips3d` ... 238
`mips4` ... 235
`mips64` ... 235
`mips64r2` ... 235
`misel` ... 252
`misize` ... 271
`missue-rate=`*`number`* ... 224
`mivc2` ... 231
`mjump-in-delay` ... 201
`mkernel` ... 189
`mknuthdiv` ... 245
`ml` ... 231, 270
`mlarge-data` ... 194
`mlarge-data-threshold=`*`number`* ... 209
`mlarge-mem` ... 279
`mlarge-text` ... 195
`mleadz` ... 232
`mleaf-id-shared-library` ... 183
`mlibfuncs` ... 245
`mlibrary-pic` ... 198
`mlinked-fp` ... 198
`mlinker-opt` ... 202
`mlinux` ... 187
`mlittle` ... 257
`mlittle-endian` ... 170, 185, 219, 230, 257, 278
`mlittle-endian-data` ... 264
`mliw` ... 246
`mllsc` ... 237
`mlocal-sdata` ... 238
`mlong-calls` ... 168, 173, 184, 198, 241, 282
`mlong-double-128` ... 266
`mlong-double-64` ... 266
`mlong-load-store` ... 202
`mlong32` ... 238
`mlong64` ... 238
`mlongcall` ... 261
`mlongcalls` ... 285
`mlow-64k` ... 183
`mlp64` ... 220
`mm` ... 232
`MM` ... 146
`mmac` ... 187, 268
`mmad` ... 241
`mmalloc64` ... 196, 222
`mmax` ... 194
`mmax-constant-size` ... 264
`mmax-stack-frame` ... 186
`mmcount-ra-address` ... 244
`mmcu` ... 175
`MMD` ... 147
`mmedia` ... 197
`mmemcpy` ... 232, 240
`mmemory-latency` ... 196
`mmemory-model` ... 278
`mmfcrf` ... 249
`mmfpgpr` ... 249
`mminimal-toc` ... 253
`mminmax` ... 232
`mmmx` ... 213
`mmodel=large` ... 223
`mmodel=medium` ... 223
`mmodel=small` ... 223
`mmovbe` ... 214
`mmt` ... 238
`mmul` ... 249
`mmul-bug-workaround` ... 186
`mmuladd` ... 197
`mmulhw` ... 256
`mmult` ... 232
`mmult-bug` ... 246
`mmulti-cond-exec` ... 199
`mmulticore` ... 184
`mmultiple` ... 255
`mmvcle` ... 267
`mmvme` ... 259
`mn` ... 201
`mnested-cond-exec` ... 200
`mnew-mnemonics` ... 251
`mnhwloop` ... 268
`mno-3dnow` ... 213
`mno-4byte-functions` ... 230
`mno-abicalls` ... 236
`mno-abshi` ... 247
`mno-ac0` ... 247
`mno-address-space-conversion` ... 280
`mno-align-double` ... 209

-mno-align-int ... 228
-mno-align-loops ... 224
-mno-align-stringops ... 216
-mno-altivec ... 252
-mno-am33 ... 246
-mno-app-regs ... 274, 282
-mno-as100-syntax ... 264
-mno-atomic-updates ... 280
-mno-avoid-indexed-addresses ... 256
-mno-backchain ... 266
-mno-base-addresses ... 245
-mno-bit-align ... 256
-mno-bitfield ... 228
-mno-branch-likely ... 243
-mno-branch-predict ... 245
-mno-bwx ... 194
-mno-callgraph-data ... 230
-mno-check-zero-division ... 240
-mno-cirrus-fix-invalid-insns ... 173
-mno-cix ... 194
-mno-clearbss ... 232
-mno-cmpb ... 249
-mno-cond-exec ... 199
-mno-cond-move ... 199
-mno-const-align ... 186
-mno-const16 ... 284
-mno-crt0 ... 246
-mno-csync-anomaly ... 183
-mno-data-align ... 186
-mno-debug ... 267
-mno-div ... 228, 230
-mno-dlmzb ... 256
-mno-double ... 197
-mno-dsp ... 237
-mno-dspr2 ... 237
-mno-dwarf2-asm ... 220
-mno-dword ... 197
-mno-eabi ... 259
-mno-early-stop-bits ... 220
-mno-eflags ... 198
-mno-embedded-data ... 239
-mno-ep ... 282
-mno-epsilon ... 245
-mno-explicit-relocs ... 194, 240
-mno-extern-sdata ... 239
-mno-fancy-math-387 ... 208
-mno-faster-structs ... 275
-mno-fix ... 194
-mno-fix-24k ... 241
-mno-fix-r10000 ... 241
-mno-fix-r4000 ... 241
-mno-fix-r4400 ... 241
-mno-flat ... 274
-mno-float32 ... 247
-mno-float64 ... 247
-mno-flush-func ... 224
-mno-flush-trap ... 224
-mno-fmaf ... 278
-mno-fp-in-toc ... 253
-mno-fp-regs ... 192
-mno-fp-ret-in-387 ... 208
-mno-fprnd ... 249
-mno-fpu ... 274
-mno-fused-madd ... 220, 241, 256, 268, 284
-mno-gnu-as ... 219
-mno-gnu-ld ... 219
-mno-gotplt ... 187
-mno-gpopt ... 239
-mno-hard-dfp ... 249, 266
-mno-hardlit ... 230
-mno-id-shared-library ... 183
-mno-ieee-fp ... 208
-mno-inline-float-divide ... 219
-mno-inline-int-divide ... 220
-mno-inline-sqrt ... 220
-mno-int16 ... 247
-mno-int32 ... 247
-mno-interlink-mips16 ... 235
-mno-interrupts ... 177
-mno-isel ... 252
-mno-knuthdiv ... 245
-mno-leaf-id-shared-library ... 184
-mno-libfuncs ... 245
-mno-llsc ... 237
-mno-local-sdata ... 238
-mno-long-calls ... 173, 184, 203, 241, 282
-mno-longcall ... 261
-mno-longcalls ... 285
-mno-low-64k ... 183
-mno-lsim ... 196, 230
-mno-mad ... 241
-mno-max ... 194
-mno-mcount-ra-address ... 244
-mno-mdmx ... 238
-mno-media ... 197
-mno-memcpy ... 240
-mno-mfcrf ... 249
-mno-mfpgpr ... 249
-mno-mips16 ... 235
-mno-mips3d ... 238
-mno-mmx ... 213
-mno-mt ... 238
-mno-mul-bug-workaround ... 186
-mno-muladd ... 197
-mno-mulhw ... 256
-mno-mult-bug ... 246
-mno-multi-cond-exec ... 199
-mno-multiple ... 255
-mno-mvcle ... 267
-mno-nested-cond-exec ... 200
-mno-optimize-membar ... 200
-mno-opts ... 232
-mno-pack ... 198
-mno-packed-stack ... 266
-mno-paired ... 253
-mno-paired-single ... 238

mno-pic ... 219
mno-pid ... 265
mno-plt ... 236
mno-popc ... 277
mno-popcntb ... 249
mno-popcntd ... 249
mno-postinc ... 169
mno-postmodify ... 169
mno-power ... 249
mno-power2 ... 249
mno-powerpc ... 249
mno-powerpc-gfxopt ... 249
mno-powerpc-gpopt ... 249
mno-powerpc64 ... 249
mno-prolog-function ... 282
mno-prologue-epilogue ... 187
mno-prototype ... 259
mno-push-args ... 215
mno-red-zone ... 217
mno-register-names ... 219
mno-regnames ... 260
mno-relax-immediate ... 230
mno-relocatable ... 256
mno-relocatable-lib ... 257
mno-round-nearest ... 168
mno-rtd ... 228
mno-scc ... 199
mno-sched-ar-data-spec ... 221
mno-sched-ar-in-data-spec ... 221
mno-sched-br-data-spec ... 220
mno-sched-br-in-data-spec ... 221
mno-sched-control-spec ... 221
mno-sched-count-spec-in-critical-path ... 221
mno-sched-in-control-spec ... 221
mno-sched-prefer-non-control-spec-insns ... 221
mno-sched-prefer-non-data-spec-insns ... 221
mno-sched-prolog ... 170
mno-sdata ... 219, 260
mno-sep-data ... 184
mno-serialize-volatile ... 284
mno-short ... 228
mno-side-effects ... 186
mno-sim ... 264
mno-single-exit ... 246
mno-slow-bytes ... 230
mno-small-exec ... 267
mno-smartmips ... 237
mno-soft-cmpsf ... 167
mno-soft-float ... 191
mno-space-regs ... 202
mno-spe ... 253
mno-specld-anomaly ... 183
mno-split-addresses ... 240
mno-sse ... 213
mno-stack-align ... 186
mno-stack-bias ... 279
mno-strict-align ... 229, 256
mno-string ... 255
mno-sum-in-toc ... 253
mno-sym32 ... 238
mno-target-align ... 285
mno-text-section-literals ... 284
mno-tls-markers ... 261
mno-toc ... 257
mno-toplevel-symbols ... 245
mno-tpf-trace ... 268
mno-unaligned-access ... 175
mno-unaligned-doubles ... 275
mno-uninit-const-in-rodata ... 239
mno-update ... 255
mno-v8plus ... 277
mno-vect-double ... 169
mno-vis ... 277
mno-vis2 ... 277
mno-vis3 ... 277
mno-vliw-branch ... 199
mno-volatile-asm-stop ... 219
mno-vrsave ... 252
mno-vsx ... 253
mno-wide-bitfields ... 230
mno-xgot ... 229, 236
mno-xl-compat ... 254
mno-zero-extend ... 245
mnobitfield ... 228
mnoieee ... 270
mnoliw ... 246
mnomacsave ... 270
mnop-fun-dllimport ... 218
mnops ... 167
mnosetlb ... 247
mnosplit-lohi ... 169
mold-mnemonics ... 251
momit-leaf-frame-pointer ... 183, 216
mone-byte-bool ... 189
moptimize-membar ... 200
MP ... 146
mpa-risc-1-0 ... 201
mpa-risc-1-1 ... 201
mpa-risc-2-0 ... 201
mpack ... 198
mpacked-stack ... 266
mpadstruct ... 271
mpaired ... 253
mpaired-single ... 238
mpc32 ... 210
mpc64 ... 210
mpc80 ... 210
mpcrel ... 229
mpdebug ... 186
mpe ... 254
mpe-aligned-commons ... 218
mpic-register ... 173
mpid ... 265
mplt ... 236
mpointers-to-nested-functions ... 263

mpoke-function-name ... 173
mpopc ... 277
mpopcntb ... 249
mpopcntd ... 249
mportable-runtime ... 202
mpower ... 249
mpower2 ... 249
mpowerpc ... 249
mpowerpc-gfxopt ... 249
mpowerpc-gpopt ... 249
mpowerpc64 ... 249
mprefer-short-insn-regs ... 167
mprefergot ... 271
mpreferred-stack-boundary ... 211
mpretend-cmove ... 273
mprioritize-restricted-insns ... 257
mprolog-function ... 282
mprologue-epilogue ... 187
mprototype ... 259
mpt-fixed ... 273
mpush-args ... 215
MQ ... 147
mr10k-cache-barrier ... 242
mrecip ... 214, 261
mrecip-precision ... 262
mrecip=opt ... 214, 262
mregister-names ... 219
mregnames ... 260
mregparm ... 210
mrelax ... 177, 201, 246, 264, 270
mrelax-immediate ... 230
mrelax-pic-calls ... 244
mrelocatable ... 256
mrelocatable-lib ... 257
mrepeat ... 232
mreturn-pointer-on-d0 ... 246
mrtd ... 209, 228, 335
mrtp ... 283
ms ... 201, 232
ms2600 ... 201
msafe-dma ... 279
msafe-hints ... 281
msahf ... 214
msatur ... 232
msave-acc-in-interrupts ... 265
msave-toc-indirect ... 263
mscc ... 199
msched-ar-data-spec ... 221
msched-ar-in-data-spec ... 221
msched-br-data-spec ... 220
msched-br-in-data-spec ... 221
msched-control-spec ... 221
msched-costly-dep ... 257
msched-count-spec-in-critical-path ... 221
msched-fp-mem-deps-zero-cost ... 222
msched-in-control-spec ... 221
msched-max-memory-insns ... 222
msched-max-memory-insns-hard-limit ... 222
msched-prefer-non-control-spec-insns ... 221
msched-prefer-non-data-spec-insns ... 221
msched-spec-ldc ... 221, 222
msched-stop-bits-after-every-cycle ... 222
mschedule ... 202
mscore5 ... 268
mscore5u ... 269
mscore7 ... 269
mscore7d ... 269
msda ... 282
msdata ... 219, 260
msdata=all ... 185
msdata=data ... 260
msdata=default ... 185, 260
msdata=eabi ... 260
msdata=none ... 185, 223, 260
msdata=sdata ... 224
msdata=sysv ... 260
msdata=use ... 224
msdram ... 185, 232
msecure-plt ... 252
msel-sched-dont-check-control-spec ... 222
msep-data ... 184
mserialize-volatile ... 284
msetlb ... 247
mshared-library-id ... 184
mshort ... 228
mshort-calls ... 178
msign-extend-enabled ... 222
msim ... 183, 185, 187, 223, 232, 249, 259, 264, 284
msimnovec ... 232
msimple-fpu ... 255
msingle-exit ... 246
msingle-float ... 237, 255
msingle-pic-base ... 173, 257
msio ... 203
mslow-bytes ... 230
msmall-data ... 194
msmall-data-limit ... 264
msmall-divides ... 233
msmall-exec ... 267
msmall-mem ... 279
msmall-model ... 196
msmall-text ... 195
msmall16 ... 168
msmartmips ... 237
msoft-atomic ... 271
msoft-float ... 191, 197, 202, 208, 227, 232, 237, 247, 255, 265, 274
msoft-quad-float ... 275
msp8 ... 178
mspace ... 271, 282
mspe ... 253
mspecld-anomaly ... 183
msplit-addresses ... 240
msplit-vecmove-early ... 169
msse ... 213
msse2avx ... 216

`msseregparm` ... 210
`mstack-align` ... 186
`mstack-bias` ... 279
`mstack-check-l1` ... 183
`mstack-guard` ... 268
`mstack-increment` ... 230
`mstack-offset` ... 168
`mstack-size` ... 268
`mstackrealign` ... 211
`mstdmain` ... 279
`mstrict-align` ... 229, 256
`mstrict-X` ... 178
`mstring` ... 255
`mstringop-strategy=`*`alg`* ... 216
`mstructure-size-boundary` ... 172
`msvr4-struct-return` ... 258
`msym32` ... 238
`msynci` ... 244
`MT` ... 146
`mtarget-align` ... 285
`mtda` ... 282
`mtext-section-literals` ... 284
`mtf` ... 232
`mthread` ... 218
`mthreads` ... 216
`mthumb` ... 174
`mthumb-interwork` ... 170
`mtiny-stack` ... 178
`mtiny=` ... 232
`mtls` ... 198
`mTLS` ... 198
`mtls-dialect` ... 174, 215
`mtls-direct-seg-refs` ... 216
`mtls-markers` ... 261
`mtls-size` ... 220
`mtoc` ... 257
`mtomcat-stats` ... 200
`mtoplevel-symbols` ... 245
`mtp` ... 174
`mtpcs-frame` ... 174
`mtpcs-leaf-frame` ... 174
`mtpf-trace` ... 268
`mtrap-precision` ... 193
`mtune` .. 171, 186, 195, 204, 220, 226, 234, 246, 252, 267, 277
`muclibc` ... 200
`muls` ... 268
`multcost=`*`number`* ... 271
`multi_module` ... 191
`multilib-library-pic` ... 198
`multiply-enabled` ... 222
`multiply_defined` ... 191
`multiply_defined_unused` ... 191
`munaligned-access` ... 175
`munaligned-doubles` ... 275
`municode` ... 218
`muninit-const-in-rodata` ... 239
`munix` ... 283
`munix-asm` ... 248
`munsafe-dma` ... 279
`mupdate` ... 255
`muser-enabled` ... 222
`musermode` ... 271
`mv850` ... 282
`mv850e` ... 283
`mv850e1` ... 282
`mv850e2` ... 282
`mv850e2v3` ... 282
`mv850es` ... 283
`mv8plus` ... 277
`mveclibabi` ... 215, 262
`mvect8-ret-in-mem` ... 210
`mvis` ... 277
`mvis2` ... 277
`mvis3` ... 277
`mvliw-branch` ... 199
`mvms-return-codes` ... 196, 222
`mvolatile-asm-stop` ... 219
`mvr4130-align` ... 244
`mvrsave` ... 252
`mvsx` ... 253
`mvxworks` ... 259
`mvzeroupper` ... 214
`mwarn-cell-microcode` ... 252
`mwarn-dynamicstack` ... 268
`mwarn-framesize` ... 268
`mwarn-reloc` ... 279
`mwide-bitfields` ... 230
`mwin32` ... 218
`mwindows` ... 218
`mword-relocations` ... 175
`mwords-little-endian` ... 170
`mx32` ... 217
`mxgot` ... 229, 236
`mxilinx-fpu` ... 255
`mxl-barrel-shift` ... 233
`mxl-compat` ... 254
`mxl-float-convert` ... 233
`mxl-float-sqrt` ... 233
`mxl-gp-opt` ... 233
`mxl-multiply-high` ... 233
`mxl-pattern-compare` ... 233
`mxl-soft-div` ... 233
`mxl-soft-mul` ... 232
`mxl-stack-check` ... 233
`myellowknife` ... 259
`mzarch` ... 267
`mzda` ... 282
`mzero-extend` ... 245

N

`no-canonical-prefixes` ... 28
`no-integrated-cpp` ... 34
`no_dead_strip_inits_and_terms` ... 191
`noall_load` ... 191

nocpp 241
nodefaultlibs 155
nofixprebinding 191
nofpu 263
nolibdld 204
nomultidefs 191
non-static 283
noprebind 191
noseglinkedit 191
nostartfiles 155
nostdinc 149
nostdinc++ 40, 149
nostdlib 155

O

o 26, 144
O 93
O0 94
O1 93
O2 93
O3 94
Ofast 94
Os 94

P

p 77
P 153
pagezero_size 191
param 128
pass-exit-codes 25
pedantic 5, 50, 145, 309, 421, 677
pedantic-errors 5, 50, 145, 677
pg 77
pie 155
pipe 26
prebind 191
prebind_all_twolevel_modules 191
print-file-name 91
print-libgcc-file-name 91
print-multi-directory 91
print-multi-lib 91
print-multi-os-directory 91
print-objc-runtime-info 48
print-prog-name 91
print-search-dirs 91
print-sysroot 92
print-sysroot-headers-suffix 92
private_bundle 191
pthread 261, 274
pthreads 274

Q

Q 77
Qn 281
Qy 281

R

rdynamic 156
read_only_relocs 191
remap 153

S

s 156
S 26, 154
save-temps 89
save-temps=obj 90
sectalign 191
sectcreate 191
sectobjectsymbols 191
sectorder 191
seg_addr_table 191
seg_addr_table_filename 191
seg1addr 191
segaddr 191
seglinkedit 191
segprot 191
segs_read_only_addr 191
segs_read_write_addr 191
shared 156
shared-libgcc 156
short-calls 168
sim 187
sim2 187
single_module 191
specs 159
static 156, 191, 204
static-libgcc 156
std 5, 30, 433, 675
std= 148
sub_library 191
sub_umbrella 191
symbolic 157
sysroot 159

T

T 157
target-help 26, 153
threads 204
time 90
tno-android-cc 200
tno-android-ld 200
traditional 34, 665
traditional-cpp 34, 153
trigraphs 34, 153
twolevel_namespace 191

U

u 157
U 144
umbrella 191

undef 144
undefined 191
unexported_symbols_list 191

V

v 26, 153
version 28, 154

W

w 49, 145
W 51, 68, 70, 666
Wa 154
Wabi 40
Waddress 67
Waggregate-return 67
Wall 50, 144, 668
Warray-bounds 61
Wassign-intercept 47
Wattributes 67
Wbad-function-cast 65
Wbuiltin-macro-redefined 67
Wcast-align 65
Wcast-qual 65
Wchar-subscripts 52
Wclobbered 66
Wcomment 52, 144
Wcomments 144
Wconversion 66
Wconversion-null 66
Wctor-dtor-privacy 42
Wdeclaration-after-statement 64
Wdelete-non-virtual-dtor 42
Wdeprecated 69
Wdeprecated-declarations 69
Wdisabled-optimization 72
Wdiv-by-zero 62
Wdouble-promotion 52
weak_reference_mismatches 191
Weffc++ 43
Wempty-body 66
Wendif-labels 64, 145
Wenum-compare 66
Werror 49, 145
Werror= 49
Wextra 51, 68, 70
Wfatal-errors 49
Wfloat-equal 62
Wformat 52, 68, 338
Wformat-contains-nul 53
Wformat-extra-args 53
Wformat-nonliteral 54, 339
Wformat-security 54
Wformat-y2k 53
Wformat-zero-length 53
Wformat=2 54
Wframe-larger-than 64
Wfree-nonheap-object 64
whatsloaded 191
whyload 191
Wignored-qualifiers 54
Wimplicit 54
Wimplicit-function-declaration 54
Wimplicit-int 54
Winit-self 54
Winline 70, 381
Wint-to-pointer-cast 71
Winvalid-offsetof 71
Winvalid-pch 71
Wjump-misses-init 66
Wl 157
Wlarger-than-*len* 64
Wlarger-than=*len* 64
Wlogical-op 67
Wlong-long 71
Wmain 55
Wmaybe-uninitialized 59
Wmissing-braces 55
Wmissing-declarations 68
Wmissing-field-initializers 68
Wmissing-format-attribute 68
Wmissing-include-dirs 55
Wmissing-parameter-type 68
Wmissing-prototypes 68
Wmultichar 69
Wnarrowing 42
Wnested-externs 70
Wno-abi 40
Wno-address 67
Wno-aggregate-return 67
Wno-all 50
Wno-array-bounds 61
Wno-assign-intercept 47
Wno-attributes 67
Wno-bad-function-cast 65
Wno-builtin-macro-redefined 67
Wno-cast-align 65
Wno-cast-qual 65
Wno-char-subscripts 52
Wno-clobbered 66
Wno-comment 52
Wno-conversion 66
Wno-conversion-null 66
Wno-coverage-mismatch 52
Wno-ctor-dtor-privacy 42
Wno-declaration-after-statement 64
Wno-delete-non-virtual-dtor 42
Wno-deprecated 69
Wno-deprecated-declarations 69
Wno-disabled-optimization 72
Wno-div-by-zero 62
Wno-double-promotion 52
Wno-effc++ 43
Wno-empty-body 66
Wno-endif-labels 64

-Wno-enum-compare ... 66
-Wno-error ... 49
-Wno-error= ... 49
-Wno-extra ... 51, 68, 70
-Wno-fatal-errors ... 49
-Wno-float-equal ... 62
-Wno-format ... 52, 68
-Wno-format-contains-nul ... 53
-Wno-format-extra-args ... 53
-Wno-format-nonliteral ... 54
-Wno-format-security ... 54
-Wno-format-y2k ... 53
-Wno-format-zero-length ... 53
-Wno-format=2 ... 54
-Wno-free-nonheap-object ... 64
-Wno-ignored-qualifiers ... 54
-Wno-implicit ... 54
-Wno-implicit-function-declaration ... 54
-Wno-implicit-int ... 54
-Wno-init-self ... 54
-Wno-inline ... 70
-Wno-int-to-pointer-cast ... 71
-Wno-invalid-offsetof ... 71
-Wno-invalid-pch ... 71
-Wno-jump-misses-init ... 66
-Wno-logical-op ... 67
-Wno-long-long ... 71
-Wno-main ... 55
-Wno-maybe-uninitialized ... 59
-Wno-missing-braces ... 55
-Wno-missing-declarations ... 68
-Wno-missing-field-initializers ... 68
-Wno-missing-format-attribute ... 68
-Wno-missing-include-dirs ... 55
-Wno-missing-parameter-type ... 68
-Wno-missing-prototypes ... 68
-Wno-mudflap ... 72
-Wno-multichar ... 69
-Wno-narrowing ... 42
-Wno-nested-externs ... 70
-Wno-noexcept ... 42
-Wno-non-template-friend ... 43
-Wno-non-virtual-dtor ... 43
-Wno-nonnull ... 54
-Wno-old-style-cast ... 44
-Wno-old-style-declaration ... 67
-Wno-old-style-definition ... 68
-Wno-overflow ... 70
-Wno-overlength-strings ... 72
-Wno-overloaded-virtual ... 44
-Wno-override-init ... 70
-Wno-packed ... 70
-Wno-packed-bitfield-compat ... 70
-Wno-padded ... 70
-Wno-parentheses ... 55
-Wno-pedantic-ms-format ... 65
-Wno-pmf-conversions ... 44, 629
-Wno-pointer-arith ... 65
-Wno-pointer-sign ... 72
-Wno-pointer-to-int-cast ... 71
-Wno-pragmas ... 60
-Wno-protocol ... 47
-Wno-redundant-decls ... 70
-Wno-reorder ... 43
-Wno-return-type ... 57
-Wno-selector ... 47
-Wno-sequence-point ... 56
-Wno-shadow ... 64
-Wno-sign-compare ... 67
-Wno-sign-conversion ... 67
-Wno-sign-promo ... 44
-Wno-stack-protector ... 72
-Wno-strict-aliasing ... 60
-Wno-strict-aliasing=n ... 60
-Wno-strict-null-sentinel ... 43
-Wno-strict-overflow ... 60
-Wno-strict-prototypes ... 67
-Wno-strict-selector-match ... 48
-Wno-suggest-attribute= ... 61
-Wno-suggest-attribute=const ... 61
-Wno-suggest-attribute=noreturn ... 61
-Wno-suggest-attribute=pure ... 61
-Wno-switch ... 57
-Wno-switch-default ... 57
-Wno-switch-enum ... 57
-Wno-sync-nand ... 57
-Wno-system-headers ... 62
-Wno-traditional ... 62
-Wno-traditional-conversion ... 63
-Wno-trampolines ... 62
-Wno-trigraphs ... 57
-Wno-type-limits ... 65
-Wno-undeclared-selector ... 48
-Wno-undef ... 64
-Wno-uninitialized ... 58
-Wno-unknown-pragmas ... 59
-Wno-unsafe-loop-optimizations ... 64
-Wno-unused ... 58
-Wno-unused-but-set-parameter ... 57
-Wno-unused-but-set-variable ... 57
-Wno-unused-function ... 58
-Wno-unused-label ... 58
-Wno-unused-parameter ... 58
-Wno-unused-result ... 58
-Wno-unused-value ... 58
-Wno-unused-variable ... 58
-Wno-variadic-macros ... 71
-Wno-vector-operation-performance ... 71
-Wno-vla ... 71
-Wno-volatile-register-var ... 72
-Wno-write-strings ... 65
-Wno-zero-as-null-pointer-constant ... 66
-Wnoexcept ... 42
-Wnon-template-friend ... 43
-Wnon-virtual-dtor ... 43
-Wnonnull ... 54

`Wnormalized=` ... 69
`Wold-style-cast` ... 44
`Wold-style-declaration` ... 67
`Wold-style-definition` ... 68
`Woverflow` ... 70
`Woverlength-strings` ... 72
`Woverloaded-virtual` ... 44
`Woverride-init` ... 70
`Wp` ... 143
`Wpacked` ... 70
`Wpacked-bitfield-compat` ... 70
`Wpadded` ... 70
`Wparentheses` ... 55
`Wpedantic-ms-format` ... 65
`Wpmf-conversions` ... 44
`Wpointer-arith` ... 65, 328
`Wpointer-sign` ... 72
`Wpointer-to-int-cast` ... 71
`Wpragmas` ... 60
`Wprotocol` ... 47
`wrapper` ... 28
`Wredundant-decls` ... 70
`Wreorder` ... 43
`Wreturn-type` ... 57
`Wselector` ... 47
`Wsequence-point` ... 56
`Wshadow` ... 64
`Wsign-compare` ... 67
`Wsign-conversion` ... 67
`Wsign-promo` ... 44
`Wstack-protector` ... 72
`Wstack-usage` ... 64
`Wstrict-aliasing` ... 60
`Wstrict-aliasing=n` ... 60
`Wstrict-null-sentinel` ... 43
`Wstrict-overflow` ... 60
`Wstrict-prototypes` ... 67
`Wstrict-selector-match` ... 48
`Wsuggest-attribute=` ... 61
`Wsuggest-attribute=const` ... 61
`Wsuggest-attribute=noreturn` ... 61
`Wsuggest-attribute=pure` ... 61
`Wswitch` ... 57
`Wswitch-default` ... 57
`Wswitch-enum` ... 57
`Wsync-nand` ... 57
`Wsystem-headers` ... 62, 145
`Wtraditional` ... 62, 144
`Wtraditional-conversion` ... 63
`Wtrampolines` ... 62
`Wtrigraphs` ... 57, 144
`Wtype-limits` ... 65
`Wundeclared-selector` ... 48
`Wundef` ... 64, 145
`Wuninitialized` ... 58
`Wunknown-pragmas` ... 59
`Wunsafe-loop-optimizations` ... 64
`Wunsuffixed-float-constants` ... 72
`Wunused` ... 58
`Wunused-but-set-parameter` ... 57
`Wunused-but-set-variable` ... 57
`Wunused-function` ... 58
`Wunused-label` ... 58
`Wunused-local-typedefs` ... 58
`Wunused-macros` ... 145
`Wunused-parameter` ... 58
`Wunused-result` ... 58
`Wunused-value` ... 58
`Wunused-variable` ... 58
`Wvariadic-macros` ... 71
`Wvector-operation-performance` ... 71
`Wvla` ... 71
`Wvolatile-register-var` ... 72
`Wwrite-strings` ... 65
`Wzero-as-null-pointer-constant` ... 66

X

`x` ... 25, 147
`Xassembler` ... 154
`Xbind-lazy` ... 283
`Xbind-now` ... 283
`Xlinker` ... 157
`Xpreprocessor` ... 143

Y

`Ym` ... 281
`YP` ... 281

Keyword Index

!

'!' in constraint 393

#

'#' in constraint 394
#pragma 610
#pragma implementation 626
#pragma implementation, implied 626
#pragma interface 625
#pragma, reason for not using 361

$

$ 365

%

'%' in constraint 394
%include 160
%include_noerr 160
%rename 160

&

'&' in constraint 393

'

' 667

*

'*' in constraint 394
*__builtin_assume_aligned 438

+

'+' in constraint 393

-

'-lgcc', use with '-nodefaultlibs' 155
'-lgcc', use with '-nostdlib' 155
'-nodefaultlibs' and unresolved references ... 155
'-nostdlib' and unresolved references 155

.

.sdata/.sdata2 references (PowerPC) 260

/

// 365

<

'<' in constraint 391

=

'=' in constraint 393

>

'>' in constraint 391

?

'?' in constraint 393
?: extensions 317
?: side effect 317

_

'_' in variables in macros 316
__atomic_add_fetch 430
__atomic_always_lock_free 431
__atomic_and_fetch 430
__atomic_clear 431
__atomic_compare_exchange 430
__atomic_compare_exchange_n 429
__atomic_exchange 429
__atomic_exchange_n 429
__atomic_fetch_add 430
__atomic_fetch_and 430
__atomic_fetch_nand 430
__atomic_fetch_or 430
__atomic_fetch_sub 430
__atomic_fetch_xor 430
__atomic_is_lock_free 431
__atomic_load 429
__atomic_load_n 429
__atomic_nand_fetch 430
__atomic_or_fetch 430
__atomic_signal_fence 431
__atomic_store 429
__atomic_store_n 429
__atomic_sub_fetch 430
__atomic_test_and_set 430
__atomic_thread_fence 431
__atomic_xor_fetch 430
__builtin___clear_cache 438
__builtin___fprintf_chk 431
__builtin___memcpy_chk 431
__builtin___memmove_chk 431

__builtin___mempcpy_chk ... 431
__builtin___memset_chk ... 431
__builtin___printf_chk ... 431
__builtin___snprintf_chk ... 431
__builtin___sprintf_chk ... 431
__builtin___stpcpy_chk ... 431
__builtin___strcat_chk ... 431
__builtin___strcpy_chk ... 431
__builtin___strncat_chk ... 431
__builtin___strncpy_chk ... 431
__builtin___vfprintf_chk ... 431
__builtin___vprintf_chk ... 431
__builtin___vsnprintf_chk ... 431
__builtin___vsprintf_chk ... 431
__builtin_apply ... 315
__builtin_apply_args ... 314
__builtin_bswap32 ... 442
__builtin_bswap64 ... 442
__builtin_choose_expr ... 435
__builtin_clrsb ... 441
__builtin_clrsbl ... 441
__builtin_clrsbll ... 441
__builtin_clz ... 440
__builtin_clzl ... 441
__builtin_clzll ... 441
__builtin_complex ... 436
__builtin_constant_p ... 436
__builtin_ctz ... 441
__builtin_ctzl ... 441
__builtin_ctzll ... 441
__builtin_expect ... 437
__builtin_extract_return_address ... 422
__builtin_ffs ... 440
__builtin_ffsl ... 441
__builtin_ffsll ... 441
__builtin_fpclassify ... 433, 439
__builtin_frame_address ... 423
__builtin_frob_return_address ... 423
__builtin_huge_val ... 439
__builtin_huge_valf ... 439
__builtin_huge_vall ... 439
__builtin_huge_valq ... 538
__builtin_inf ... 439
__builtin_infd128 ... 439
__builtin_infd32 ... 439
__builtin_infd64 ... 439
__builtin_inff ... 439
__builtin_infl ... 439
__builtin_infq ... 538
__builtin_isfinite ... 433
__builtin_isgreater ... 433
__builtin_isgreaterequal ... 433
__builtin_isinf_sign ... 433, 440
__builtin_isless ... 433
__builtin_islessequal ... 433
__builtin_islessgreater ... 433
__builtin_isnormal ... 433
__builtin_isunordered ... 433
__builtin_nan ... 440
__builtin_nand128 ... 440
__builtin_nand32 ... 440
__builtin_nand64 ... 440
__builtin_nanf ... 440
__builtin_nanl ... 440
__builtin_nans ... 440
__builtin_nansf ... 440
__builtin_nansl ... 440
__builtin_object_size ... 431
__builtin_offsetof ... 425
__builtin_parity ... 441
__builtin_parityl ... 441
__builtin_parityll ... 441
__builtin_popcount ... 441
__builtin_popcountl ... 441
__builtin_popcountll ... 441
__builtin_powi ... 433, 442
__builtin_powif ... 433, 442
__builtin_powil ... 433, 442
__builtin_prefetch ... 438
__builtin_return ... 315
__builtin_return_address ... 422
__builtin_rx_brk ... 603
__builtin_rx_clrpsw ... 603
__builtin_rx_int ... 604
__builtin_rx_machi ... 604
__builtin_rx_maclo ... 604
__builtin_rx_mulhi ... 604
__builtin_rx_mullo ... 604
__builtin_rx_mvfachi ... 604
__builtin_rx_mvfacmi ... 604
__builtin_rx_mvfc ... 604
__builtin_rx_mvtachi ... 604
__builtin_rx_mvtaclo ... 604
__builtin_rx_mvtc ... 604
__builtin_rx_mvtipl ... 604
__builtin_rx_racw ... 604
__builtin_rx_revw ... 604
__builtin_rx_rmpa ... 605
__builtin_rx_round ... 605
__builtin_rx_sat ... 605
__builtin_rx_setpsw ... 605
__builtin_rx_wait ... 605
__builtin_trap ... 437
__builtin_types_compatible_p ... 434
__builtin_unreachable ... 437
__builtin_va_arg_pack ... 315
__builtin_va_arg_pack_len ... 315
__complex__ keyword ... 318
__declspec(dllexport) ... 336
__declspec(dllimport) ... 336
__ea SPU Named Address Spaces ... 324
__extension__ ... 421
__far M32C Named Address Spaces ... 324
__far RL78 Named Address Spaces ... 324
__flash AVR Named Address Spaces ... 322
__flash1 AVR Named Address Spaces ... 322

__flash2 AVR Named Address Spaces 322
__flash3 AVR Named Address Spaces 322
__flash4 AVR Named Address Spaces 322
__flash5 AVR Named Address Spaces 322
__float128 data type 319
__float80 data type 319
__fp16 data type 319
__func__ identifier 421
__FUNCTION__ identifier 421
__imag__ keyword 318
__int128 data types 318
__memx AVR Named Address Spaces 322
__PRETTY_FUNCTION__ identifier 421
__real__ keyword 318
__STDC_HOSTED__ 5
__sync_add_and_fetch 426
__sync_and_and_fetch 426
__sync_bool_compare_and_swap 427
__sync_fetch_and_add 426
__sync_fetch_and_and 426
__sync_fetch_and_nand 426
__sync_fetch_and_or 426
__sync_fetch_and_sub 426
__sync_fetch_and_xor 426
__sync_lock_release 427
__sync_lock_test_and_set 427
__sync_nand_and_fetch 426
__sync_or_and_fetch 426
__sync_sub_and_fetch 426
__sync_synchronize 427
__sync_val_compare_and_swap 427
__sync_xor_and_fetch 426
__thread 618
_Accum data type 321
_Complex keyword 318
_Decimal128 data type 320
_Decimal32 data type 320
_Decimal64 data type 320
_exit 433
_Exit 433
_Fract data type 321
_Sat data type 321

0

'0' in constraint 392

A

ABI 651
abort 433
abs 433
accessing volatiles 382, 623
acos 433
acosf 433
acosh 433
acoshf 433
acoshl 433
acosl 433
Ada 3
additional floating types 319
address constraints 392
address of a label 311
address_operand 392
alias attribute 332
aligned attribute 333, 365, 374
alignment 380
alloc_size attribute 333
alloca 433
alloca vs variable-length arrays 326
Allow nesting in an interrupt handler on the Blackfin processor 346
alternate keywords 420
always_inline function attribute 333
AMD x86-64 Options 204
AMD1 5
ANSI C 5
ANSI C standard 5
ANSI C89 5
ANSI support 29
ANSI X3.159-1989 5
apostrophes 667
application binary interface 651
ARM [Annotated C++ Reference Manual] 634
ARM options 169
arrays of length zero 324
arrays of variable length 326
arrays, non-lvalue 328
artificial function attribute 334
asin 433
asinf 433
asinh 433
asinhf 433
asinhl 433
asinl 433
asm constraints 390
asm expressions 383
assembler instructions 383
assembler names for identifiers 417
assembly code, invalid 679
atan 433
atan2 433
atan2f 433
atan2l 433
atanf 433
atanh 433
atanhf 433
atanhl 433
atanl 433
attribute of types 374
attribute of variables 365
attribute syntax 361
autoincrement/decrement addressing 391
automatic inline for C++ member fns 381
AVR Options 175

B

Backwards Compatibility 634
base class members 671
`bcmp` 433
`below100` attribute 374
binary compatibility 651
Binary constants using the '`0b`' prefix 620
Blackfin Options 182
bound pointer to member function 629
bounds checking 98
bug criteria 679
bugs 679
bugs, known 663
built-in functions 32, 433
`bzero` 433

C

C compilation options 9
C intermediate output, nonexistent 3
C language extensions 309
C language, traditional 34
C standard 5
C standards 5
`c++` 29
C++ 3
C++ comments 365
C++ compilation options 9
C++ interface and implementation headers 625
C++ language extensions 623
C++ member fns, automatically `inline` 381
C++ misunderstandings 670
C++ options, command-line 35
C++ pragmas, effect on inlining 626
C++ source file suffixes 29
C++ static data, declaring and defining 670
`C_INCLUDE_PATH` 296
C11 5
C1X 5
C6X Options 185
C89 5
C90 5
C94 5
C95 5
C99 5
C9X 5
`cabs` 433
`cabsf` 433
`cabsl` 433
`cacos` 433
`cacosf` 433
`cacosh` 433
`cacoshf` 433
`cacoshl` 433
`cacosl` 433
`callee_pop_aggregate_return` attribute 346
calling functions through the function vector on H8/300, M16C, M32C and SH2A processors 340
`calloc` 433
`carg` 433
`cargf` 433
`cargl` 433
case labels in initializers 330
case ranges 331
`casin` 433
`casinf` 433
`casinh` 433
`casinhf` 433
`casinhl` 433
`casinl` 433
cast to a union 331
`catan` 433
`catanf` 433
`catanh` 433
`catanhf` 433
`catanhl` 433
`catanl` 433
`cbrt` 433
`cbrtf` 433
`cbrtl` 433
`ccos` 433
`ccosf` 433
`ccosh` 433
`ccoshf` 433
`ccoshl` 433
`ccosl` 433
`ceil` 433
`ceilf` 433
`ceill` 433
`cexp` 433
`cexpf` 433
`cexpl` 433
character set, execution 151
character set, input 151
character set, input normalization 69
character set, wide execution 151
`cimag` 433
`cimagf` 433
`cimagl` 433
`cleanup` attribute 366
`clog` 433
`clogf` 433
`clogl` 433
COBOL 3
code generation conventions 285
code, mixed with declarations 332
`cold` function attribute 350
command options 9
comments, C++ style 365
`common` attribute 367
comparison of signed and unsigned values, warning 67
compiler bugs, reporting 679

compiler compared to C++ preprocessor 3
compiler options, C++ 35
compiler options, Objective-C and Objective-C++ 45
compiler version, specifying 167
COMPILER_PATH 295
complex conjugation 318
complex numbers 318
compound literals 328
computed gotos 311
conditional expressions, extensions 317
conflicting types 669
conj 433
conjf 433
conjl 433
const applied to function 332
const function attribute 335
constants in constraints 391
constraint modifier characters 393
constraint, matching 392
constraints, asm 390
constraints, machine specific 394
constructing calls 314
constructor expressions 328
constructor function attribute 335
contributors 709
copysign 433
copysignf 433
copysignl 433
core dump 679
cos 433
cosf 433
cosh 433
coshf 433
coshl 433
cosl 433
CPATH 296
CPLUS_INCLUDE_PATH 296
cpow 433
cpowf 433
cpowl 433
cproj 433
cprojf 433
cprojl 433
CR16 Options 187
creal 433
crealf 433
creall 433
CRIS Options 185
cross compiling 167
csin 433
csinf 433
csinh 433
csinhf 433
csinhl 433
csinl 433
csqrt 433
csqrtf 433
csqrtl 433
ctan 433
ctanf 433
ctanh 433
ctanhf 433
ctanhl 433
ctanl 433

D

Darwin options 188
dcgettext 433
dd integer suffix 320
DD integer suffix 320
deallocating variable length arrays 326
debugging information options 72
decimal floating types 320
declaration scope 666
declarations inside expressions 309
declarations, mixed with code 332
declaring attributes of functions 332
declaring static data in C++ 670
defining static data in C++ 670
dependencies for make as output 296
dependencies, make 145
DEPENDENCIES_OUTPUT 296
dependent name lookup 671
deprecated attribute 367
deprecated attribute. 335
designated initializers 330
designator lists 331
designators 330
destructor function attribute 335
df integer suffix 320
DF integer suffix 320
dgettext 433
diagnostic messages 48
dialect options 29
digits in constraint 392
directory options 158
disinterrupt attribute 336
dl integer suffix 320
DL integer suffix 320
dollar signs in identifier names 365
double-word arithmetic 318
downward funargs 312
drem 433
dremf 433
dreml 433

E

'E' in constraint 391
earlyclobber operand 393
eight bit data on the H8/300, H8/300H, and H8S 337
EIND 178
empty structures 326

environment variables 294
erf 433
erfc 433
erfcf 433
erfcl 433
erff 433
erfl 433
error function attribute 334
error messages 677
escaped newlines 328
exception handler functions on the Blackfin processor 337
exclamation point 393
exit 433
exp 433
exp10 433
exp10f 433
exp10l 433
exp2 433
exp2f 433
exp2l 433
expf 433
expl 433
explicit register variables 418
expm1 433
expm1f 433
expm1l 433
expressions containing statements 309
expressions, constructor 328
extended asm 383
extensible constraints 392
extensions, ?: 317
extensions, C language 309
extensions, C++ language 623
external declaration scope 666
externally_visible attribute 337

F

'F' in constraint 391
fabs 433
fabsf 433
fabsl 433
fatal signal 679
fdim 433
fdimf 433
fdiml 433
FDL, GNU Free Documentation License 701
ffs 433
file name suffix 23
file names 154
fixed-point types 321
flatten function attribute 334
flexible array members 324
float as function value type 667
floating point precision 670
floating-point precision 122
floor 433
floorf 433
floorl 433
fma 433
fmaf 433
fmal 433
fmax 433
fmaxf 433
fmaxl 433
fmin 433
fminf 433
fminl 433
fmod 433
fmodf 433
fmodl 433
force_align_arg_pointer attribute 350
format function attribute 338
format_arg function attribute 339
Fortran 3
forwarder_section attribute 342
forwarding calls 314
fprintf 433
fprintf_unlocked 433
fputs 433
fputs_unlocked 433
FR30 Options 196
freestanding environment 5
freestanding implementation 5
frexp 433
frexpf 433
frexpl 433
FRV Options 196
fscanf 433
fscanf, and constant strings 665
function addressability on the M32R/D 345
function attributes 332
function pointers, arithmetic 328
function prototype declarations 364
function without a prologue/epilogue code 346
function, size of pointer to 328
functions called via pointer on the RS/6000 and PowerPC 344
functions in arbitrary sections 332
functions that are dynamically resolved 332
functions that are passed arguments in registers on the 386 ..... 332, 350
functions that behave like malloc 332
functions that do not pop the argument stack on the 386 332
functions that do pop the argument stack on the 386 335
functions that have different compilation options on the 386 332
functions that have different optimization options 332
functions that have no side effects 332
functions that never return 332
functions that pop the argument stack on the 386 ..... 332, 338, 352

functions that return more than once 332
functions which do not handle memory bank switching on 68HC11/68HC12 346
functions which handle memory bank switching 338
functions with non-null pointer arguments 332
functions with `printf`, `scanf`, `strftime` or `strfmon` style arguments 332

G

'`g`' in constraint 392
'`G`' in constraint 392
`g++` 29
G++ 3
`gamma` 433
`gamma_r` 433
`gammaf` 433
`gammaf_r` 433
`gammal` 433
`gammal_r` 433
GCC 3
GCC command options 9
`GCC_COMPARE_DEBUG` 295
`GCC_EXEC_PREFIX` 295
`gcc_struct` 379
`gcc_struct` attribute 372
`gcov` 78
`gettext` 433
global offset table 288
global register after `longjmp` 419
global register variables 418
GNAT 3
GNU C Compiler 3
GNU Compiler Collection 3
`gnu_inline` function attribute 333
Go 3
goto with computed label 311
`gprof` 77
grouping options 9

H

'`H`' in constraint 392
half-precision floating point 319
hardware models and configurations, specifying 167
hex floats 320
`hk` fixed-suffix 321
`HK` fixed-suffix 321
hosted environment 5, 33
hosted implementation 5
`hot` function attribute 349
HPPA Options 201
`hr` fixed-suffix 321
`HR` fixed-suffix 321
`hypot` 433
`hypotf` 433
`hypotl` 433

I

'`i`' in constraint 391
'`I`' in constraint 391
i386 and x86-64 Windows Options 218
i386 Options 204
IA-64 Options 218
IBM RS/6000 and PowerPC Options 249
identifier names, dollar signs in 365
identifiers, names in assembler code 417
`ifunc` attribute 341
`ilogb` 433
`ilogbf` 433
`ilogbl` 433
`imaxabs` 433
implementation-defined behavior, C language 299
implementation-defined behavior, C++ language 307
implied `#pragma implementation` 626
incompatibilities of GCC 665
increment operators 679
`index` 433
indirect calls on ARM 344
indirect calls on MIPS 345
`init_priority` attribute 629
initializations in expressions 328
initializers with labeled elements 330
initializers, non-constant 328
`inline` automatic for C++ member fns 381
inline functions 380
inline functions, omission of 381
inlining and C++ pragmas 626
installation trouble 663
integrating function code 380
Intel 386 Options 204
interface and implementation headers, C++ 625
intermediate C version, nonexistent 3
interrupt handler functions 334, 338, 342
interrupt handler functions on the AVR processors 352
interrupt handler functions on the Blackfin, m68k, H8/300 and SH processors 343
interrupt service routines on ARM 343
interrupt thread functions on fido 343
introduction 1
invalid assembly code 679
invalid input 679
invoking `g++` 29
`isalnum` 433
`isalpha` 433
`isascii` 433
`isblank` 433
`iscntrl` 433
`isdigit` 433
`isgraph` 433

islower ... 433
ISO 9899 ... 5
ISO C ... 5
ISO C standard ... 5
ISO C11 ... 5
ISO C1X ... 5
ISO C90 ... 5
ISO C94 ... 5
ISO C95 ... 5
ISO C99 ... 5
ISO C9X ... 5
ISO support ... 29
ISO/IEC 9899 ... 5
isprint ... 433
ispunct ... 433
isspace ... 433
isupper ... 433
iswalnum ... 433
iswalpha ... 433
iswblank ... 433
iswcntrl ... 433
iswdigit ... 433
iswgraph ... 433
iswlower ... 433
iswprint ... 433
iswpunct ... 433
iswspace ... 433
iswupper ... 433
iswxdigit ... 433
isxdigit ... 433

J

j0 ... 433
j0f ... 433
j0l ... 433
j1 ... 433
j1f ... 433
j1l ... 433
Java ... 3
java_interface attribute ... 629
jn ... 433
jnf ... 433
jnl ... 433

K

k fixed-suffix ... 321
K fixed-suffix ... 321
keep_interrupts_masked attribute ... 343
keywords, alternate ... 420
known causes of trouble ... 663

L

l1_data variable attribute ... 371
l1_data_A variable attribute ... 371
l1_data_B variable attribute ... 371
l1_text function attribute ... 344
l2 function attribute ... 344
l2 variable attribute ... 371
labeled elements in initializers ... 330
labels as values ... 311
labs ... 433
LANG ... 294, 295
language dialect options ... 29
LC_ALL ... 294
LC_CTYPE ... 294
LC_MESSAGES ... 294
ldexp ... 433
ldexpf ... 433
ldexpl ... 433
leaf function attribute ... 344
length-zero arrays ... 324
lgamma ... 433
lgamma_r ... 433
lgammaf ... 433
lgammaf_r ... 433
lgammal ... 433
lgammal_r ... 433
Libraries ... 154
LIBRARY_PATH ... 295
link options ... 154
linker script ... 157
lk fixed-suffix ... 321
LK fixed-suffix ... 321
LL integer suffix ... 318
llabs ... 433
llk fixed-suffix ... 321
LLK fixed-suffix ... 321
llr fixed-suffix ... 321
LLR fixed-suffix ... 321
llrint ... 433
llrintf ... 433
llrintl ... 433
llround ... 433
llroundf ... 433
llroundl ... 433
LM32 options ... 222
load address instruction ... 392
local labels ... 310
local variables in macros ... 316
local variables, specifying registers ... 420
locale ... 294
locale definition ... 295
log ... 433
log10 ... 433
log10f ... 433
log10l ... 433
log1p ... 433
log1pf ... 433
log1pl ... 433
log2 ... 433
log2f ... 433
log2l ... 433
logb ... 433

`logbf` ... 433
`logbl` ... 433
`logf` ... 433
`logl` ... 433
`long long` data types ... 318
`longjmp` ... 419
`longjmp` incompatibilities ... 666
`longjmp` warnings ... 59
`lr` fixed-suffix ... 321
`LR` fixed-suffix ... 321
`lrint` ... 433
`lrintf` ... 433
`lrintl` ... 433
`lround` ... 433
`lroundf` ... 433
`lroundl` ... 433

M

'`m`' in constraint ... 390
M32C options ... 223
M32R/D options ... 223
M680x0 options ... 225
machine dependent options ... 167
machine specific constraints ... 394
macro with variable arguments ... 327
macros containing `asm` ... 386
macros, inline alternative ... 380
macros, local labels ... 310
macros, local variables in ... 316
macros, statements in expressions ... 309
macros, types of arguments ... 316
`make` ... 145
`malloc` ... 433
`malloc` attribute ... 345
matching constraint ... 392
MCore options ... 230
member fns, automatically `inline` ... 381
`memchr` ... 433
`memcmp` ... 433
`memcpy` ... 433
memory references in constraints ... 390
`mempcpy` ... 433
`memset` ... 433
MeP options ... 231
Mercury ... 3
message formatting ... 48
messages, warning ... 49
messages, warning and error ... 677
MicroBlaze Options ... 232
middle-operands, omitted ... 317
MIPS options ... 233
`mips16` attribute ... 345
misunderstandings in C++ ... 670
mixed declarations and code ... 332
`mktemp`, and constant strings ... 665
MMIX Options ... 245
MN10300 options ... 246
`mode` attribute ... 367
`modf` ... 433
`modff` ... 433
`modfl` ... 433
modifiers in constraints ... 393
`ms_abi` attribute ... 346
`ms_hook_prologue` attribute ... 346
`ms_struct` ... 379
`ms_struct` attribute ... 372
mudflap ... 98
multiple alternative constraints ... 393
multiprecision arithmetic ... 318

N

'`n`' in constraint ... 391
Named Address Spaces ... 322
names used in assembler code ... 417
naming convention, implementation headers ... 626
`nearbyint` ... 433
`nearbyintf` ... 433
`nearbyintl` ... 433
nested functions ... 312
newlines (escaped) ... 328
`nextafter` ... 433
`nextafterf` ... 433
`nextafterl` ... 433
`nexttoward` ... 433
`nexttowardf` ... 433
`nexttowardl` ... 433
NFC ... 69
NFKC ... 69
NMI handler functions on the Blackfin processor ... 347
`no_instrument_function` function attribute ... 347
`no_split_stack` function attribute ... 347
`noclone` function attribute ... 347
`nocommon` attribute ... 367
`noinline` function attribute ... 347
`nomips16` attribute ... 345
non-constant initializers ... 328
non-static inline function ... 381
`nonnull` function attribute ... 347
`noreturn` function attribute ... 347
`nothrow` function attribute ... 348

O

'`o`' in constraint ... 390
`OBJC_INCLUDE_PATH` ... 296
Objective-C ... 3, 7
Objective-C and Objective-C++ options, command-line ... 45
Objective-C++ ... 3, 7
offsettable address ... 390
old-style function definitions ... 364
omitted middle-operands ... 317
open coding ... 380

OpenMP parallel 33
operand constraints, `asm` 390
`optimize` function attribute 348
optimize options 92
options to control diagnostics formatting 48
options to control warnings 49
options, C++ 35
options, code generation 285
options, debugging 72
options, dialect 29
options, directory search 158
options, GCC command 9
options, grouping 9
options, linking 154
options, Objective-C and Objective-C++ 45
options, optimization 92
options, order 9
options, preprocessor 143
order of evaluation, side effects 677
order of options 9
`OS_main` AVR function attribute 348
`OS_task` AVR function attribute 348
other register constraints 392
output file option 26
overloaded virtual function, warning 44

P

'`p`' in constraint 392
`packed` attribute 367
parameter forward declaration 327
Pascal 3
`pcs` function attribute 349
PDP-11 Options 247
PIC 288
picoChip options 248
pmf 629
pointer arguments 335
pointer to member function 629
portions of temporary objects, pointers to 672
`pow` 433
`pow10` 433
`pow10f` 433
`pow10l` 433
PowerPC options 248
`powf` 433
`powl` 433
pragma GCC optimize 616
pragma GCC pop_options 616
pragma GCC push_options 616
pragma GCC reset_options 616
pragma GCC target 616
pragma, address 610
pragma, align 612
pragma, call 611
pragma, coprocessor available 611
pragma, coprocessor call_saved 611
pragma, coprocessor subclass 611
pragma, custom io_volatile 610
pragma, diagnostic 614, 615
pragma, disinterrupt 611
pragma, extern_prefix 613
pragma, fini 612
pragma, init 612
pragma, long_calls 610
pragma, long_calls_off 610
pragma, longcall 611
pragma, mark 612
pragma, memregs 610
pragma, no_long_calls 610
pragma, options align 612
pragma, pop_macro 615
pragma, push_macro 615
pragma, reason for not using 361
pragma, redefine_extname 612
pragma, segment 612
pragma, unused 612
pragma, visibility 615
pragma, weak 614
pragmas 610
pragmas in C++, effect on inlining 626
pragmas, interface and implementation 625
pragmas, warning of unknown 59
precompiled headers 297
preprocessing numbers 668
preprocessing tokens 668
preprocessor options 143
`printf` 433
`printf_unlocked` 433
`prof` 77
`progmem` AVR variable attribute 370
promotion of formal parameters 364
`pure` function attribute 349
push address instruction 392
`putchar` 433
`puts` 433

Q

`q` floating point suffix 319
`Q` floating point suffix 319
`qsort`, and global register variables 419
question mark 393

R

`r` fixed-suffix 321
`R` fixed-suffix 321
'`r`' in constraint 391
`RAMPD` 180
`RAMPX` 180
`RAMPY` 180
`RAMPZ` 180
ranges in case statements 331
read-only strings 665
register variable after `longjmp` 419

registers ... 383
registers for local variables ... 420
registers in constraints ... 391
registers, global allocation ... 418
registers, global variables in ... 418
`regparm` attribute ... 350
relocation truncated to fit (ColdFire) ... 229
relocation truncated to fit (MIPS) ... 236
`remainder` ... 433
`remainderf` ... 433
`remainderl` ... 433
`remquo` ... 433
`remquof` ... 433
`remquol` ... 433
reordering, warning ... 43
reporting bugs ... 679
`resbank` attribute ... 350
rest argument (in macro) ... 327
restricted pointers ... 623
restricted references ... 623
restricted this pointer ... 623
`returns_twice` attribute ... 351
`rindex` ... 433
`rint` ... 433
`rintf` ... 433
`rintl` ... 433
RL78 Options ... 249
`round` ... 433
`roundf` ... 433
`roundl` ... 433
RS/6000 and PowerPC Options ... 249
RTTI ... 625
run-time options ... 285
RX Options ... 263

S

'`s`' in constraint ... 392
S/390 and zSeries Options ... 265
save all registers on the Blackfin, H8/300, H8/300H, and H8S ... 351
save volatile registers on the MicroBlaze ... 351
`scalb` ... 433
`scalbf` ... 433
`scalbl` ... 433
`scalbln` ... 433
`scalblnf` ... 433
`scalbn` ... 433
`scalbnf` ... 433
`scanf`, and constant strings ... 665
`scanfnl` ... 433
scope of a variable length array ... 326
scope of declaration ... 669
scope of external declarations ... 666
Score Options ... 268
search path ... 158
`section` function attribute ... 351
`section` variable attribute ... 368
`sentinel` function attribute ... 351
`setjmp` ... 419
`setjmp` incompatibilities ... 666
shared strings ... 665
`shared` variable attribute ... 368
side effect in `?:` ... 317
side effects, macro argument ... 309
side effects, order of evaluation ... 677
`signbit` ... 433
`signbitd128` ... 433
`signbitd32` ... 433
`signbitd64` ... 433
`signbitf` ... 433
`signbitl` ... 433
signed and unsigned values, comparison warning ... 67
`significand` ... 433
`significandf` ... 433
`significandl` ... 433
simple constraints ... 390
`sin` ... 433
`sincos` ... 433
`sincosf` ... 433
`sincosl` ... 433
`sinf` ... 433
`sinh` ... 433
`sinhf` ... 433
`sinhl` ... 433
`sinl` ... 433
`sizeof` ... 316
smaller data references ... 224
smaller data references (PowerPC) ... 260
`snprintf` ... 433
Solaris 2 options ... 273
SPARC options ... 274
Spec Files ... 160
specified registers ... 418
specifying compiler version and target machine ... 167
specifying hardware config ... 167
specifying machine version ... 167
specifying registers for local variables ... 420
speed of compilation ... 297
`sprintf` ... 433
SPU options ... 279
`sqrt` ... 433
`sqrtf` ... 433
`sqrtl` ... 433
`sscanf` ... 433
`sscanf`, and constant strings ... 665
`sseregparm` attribute ... 350
statements inside expressions ... 309
static data in C++, declaring and defining ... 670
`stpcpy` ... 433
`stpncpy` ... 433
`strcasecmp` ... 433
`strcat` ... 433
`strchr` ... 433

`strcmp` ... 433
`strcpy` ... 433
`strcspn` ... 433
`strdup` ... 433
`strfmon` ... 433
`strftime` ... 433
string constants ... 665
`strlen` ... 433
`strncasecmp` ... 433
`strncat` ... 433
`strncmp` ... 433
`strncpy` ... 433
`strndup` ... 433
`strpbrk` ... 433
`strrchr` ... 433
`strspn` ... 433
`strstr` ... 433
`struct` ... 617
structures ... 667
structures, constructor expression ... 328
submodel options ... 167
subscripting ... 328
subscripting and function values ... 328
suffixes for C++ source ... 29
`SUNPRO_DEPENDENCIES` ... 296
suppressing warnings ... 49
surprises in C++ ... 670
syntax checking ... 49
`syscall_linkage` attribute ... 352
system headers, warnings from ... 62
`sysv_abi` attribute ... 346

T

`tan` ... 433
`tanf` ... 433
`tanh` ... 433
`tanhf` ... 433
`tanhl` ... 433
`tanl` ... 433
`target` function attribute ... 352
target machine, specifying ... 167
target options ... 167
`target("abm")` attribute ... 353
`target("aes")` attribute ... 353
`target("align-stringops")` attribute ... 354
`target("altivec")` attribute ... 355
`target("arch=ARCH")` attribute ... 354
`target("avoid-indexed-addresses")` attribute ... 357
`target("cld")` attribute ... 354
`target("cmpb")` attribute ... 355
`target("cpu=CPU")` attribute ... 357
`target("dlmzb")` attribute ... 355
`target("fancy-math-387")` attribute ... 354
`target("fma4")` attribute ... 353
`target("fpmath=FPMATH")` attribute ... 354
`target("fprnd")` attribute ... 355
`target("friz")` attribute ... 357
`target("fused-madd")` attribute ... 354
`target("hard-dfp")` attribute ... 355
`target("ieee-fp")` attribute ... 354
`target("inline-all-stringops")` attribute ... 354
`target("inline-stringops-dynamically")` attribute ... 354
`target("isel")` attribute ... 355
`target("longcall")` attribute ... 357
`target("lwp")` attribute ... 353
`target("mfcrf")` attribute ... 355
`target("mfpgpr")` attribute ... 355
`target("mmx")` attribute ... 353
`target("mulhw")` attribute ... 355
`target("multiple")` attribute ... 356
`target("paired")` attribute ... 357
`target("pclmul")` attribute ... 353
`target("popcnt")` attribute ... 353
`target("popcntb")` attribute ... 356
`target("popcntd")` attribute ... 356
`target("powerpc-gfxopt")` attribute ... 356
`target("powerpc-gpopt")` attribute ... 356
`target("recip")` attribute ... 354
`target("recip-precision")` attribute ... 356
`target("sse")` attribute ... 353
`target("sse2")` attribute ... 353
`target("sse3")` attribute ... 353
`target("sse4")` attribute ... 353
`target("sse4.1")` attribute ... 353
`target("sse4.2")` attribute ... 353
`target("sse4a")` attribute ... 353
`target("ssse3")` attribute ... 354
`target("string")` attribute ... 356
`target("tune=TUNE")` attribute ... 354, 357
`target("update")` attribute ... 356
`target("vsx")` attribute ... 356
`target("xop")` attribute ... 353
TC1 ... 5
TC2 ... 5
TC3 ... 5
Technical Corrigenda ... 5
Technical Corrigendum 1 ... 5
Technical Corrigendum 2 ... 5
Technical Corrigendum 3 ... 5
template instantiation ... 626
temporaries, lifetime of ... 672
`tgamma` ... 433
`tgammaf` ... 433
`tgammal` ... 433
Thread-Local Storage ... 618
thunks ... 312
TILE-Gx options ... 281
TILEPro options ... 281
tiny data section on the H8/300H and H8S ... 358
TLS ... 618
`tls_model` attribute ... 369
`TMPDIR` ... 294
`toascii` ... 433

`tolower` ... 433
`toupper` ... 433
`towlower` ... 433
`towupper` ... 433
traditional C language ... 34
`trunc` ... 433
`truncf` ... 433
`truncl` ... 433
two-stage name lookup ... 671
type alignment ... 380
type attributes ... 374
`type_info` ... 625
typedef names as function parameters ... 666
`typeof` ... 316

U

`uhk` fixed-suffix ... 321
`UHK` fixed-suffix ... 321
`uhr` fixed-suffix ... 321
`UHR` fixed-suffix ... 321
`uk` fixed-suffix ... 321
`UK` fixed-suffix ... 321
`ulk` fixed-suffix ... 321
`ULK` fixed-suffix ... 321
`ULL` integer suffix ... 318
`ullk` fixed-suffix ... 321
`ULLK` fixed-suffix ... 321
`ullr` fixed-suffix ... 321
`ULLR` fixed-suffix ... 321
`ulr` fixed-suffix ... 321
`ULR` fixed-suffix ... 321
undefined behavior ... 679
undefined function value ... 679
underscores in variables in macros ... 316
`union` ... 617
union, casting to a ... 331
unions ... 667
unknown pragmas, warning ... 59
unresolved references and '`-nodefaultlibs`' ... 155
unresolved references and '`-nostdlib`' ... 155
`unused` attribute ... 358
`ur` fixed-suffix ... 321
`UR` fixed-suffix ... 321
`use_debug_exception_return` attribute ... 343
`use_shadow_register_set` attribute ... 343
`used` attribute ... 358
User stack pointer in interrupts on the Blackfin ... 344

V

'`V`' in constraint ... 391
V850 Options ... 281
vague linkage ... 624
value after `longjmp` ... 419
variable addressability on the IA-64 ... 345
variable addressability on the M32R/D ... 371
variable alignment ... 380
variable attributes ... 365
variable number of arguments ... 327
variable-length array scope ... 326
variable-length arrays ... 326
variables in specified registers ... 418
variables, local, in macros ... 316
variadic macros ... 327
VAX options ... 283
`version_id` attribute ... 358
`vfprintf` ... 433
`vfscanf` ... 433
`visibility` attribute ... 358
VLAs ... 326
`vliw` attribute ... 360
void pointers, arithmetic ... 328
void, size of pointer to ... 328
volatile access ... 382, 623
`volatile` applied to function ... 332
volatile read ... 382, 623
volatile write ... 382, 623
`vprintf` ... 433
`vscanf` ... 433
`vsnprintf` ... 433
`vsprintf` ... 433
`vsscanf` ... 433
vtable ... 624
VxWorks Options ... 283

W

`w` floating point suffix ... 319
`W` floating point suffix ... 319
`warn_unused_result` attribute ... 360
warning for comparison of signed and unsigned values ... 67
warning for overloaded virtual function ... 44
warning for reordering of member initializers ... 43
warning for unknown pragmas ... 59
`warning` function attribute ... 334
warning messages ... 49
warnings from system headers ... 62
warnings vs errors ... 677
`weak` attribute ... 360
`weakref` attribute ... 360
whitespace ... 667

X

'`X`' in constraint ... 392
X3.159-1989 ... 5
x86-64 options ... 284
x86-64 Options ... 204
Xstormy16 Options ... 284
Xtensa Options ... 284

Y

y0 433
y0f 433
y0l 433
y1 433
y1f 433
y1l 433
yn 433
ynf 433
ynl 433

Z

zero-length arrays 324
zero-size structures 326
zSeries options 285