

Using the GNU Compiler Collection

For GCC version 17.0.0 (pre-release)

(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: <http://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.

This file documents the use of the GNU compilers.

Copyright © 1988-2026 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

1	Programming Languages Supported by GCC	1
2	Language Standards Supported by GCC	3
3	GCC Command Options	9
4	C Implementation-Defined Behavior	563
5	C++ Implementation-Defined Behavior	573
6	Extensions to the C Language Family	575
7	Built-in Functions Provided by GCC	797
8	Extensions to the C++ Language	1031
9	GNU Objective-C Features	1047
10	Binary Compatibility	1063
11	<code>gcov</code> —a Test Coverage Program	1067
12	<code>gcov-tool</code> —an Offline Gcda Profile Processing Tool	1093
13	<code>gcov-dump</code> —an Offline Gcda and Gcno Profile Dump Tool	1097
14	<code>lto-dump</code> —Tool for dumping LTO object files.	1099
15	Known Causes of Trouble with GCC	1101
16	Reporting Bugs	1117
17	How To Get Help with GCC	1119
18	Contributing to GCC Development	1121
	Funding Free Software	1123
	The GNU Project and GNU/Linux	1125
	GNU General Public License	1127
	GNU Free Documentation License	1139
	Contributors to GCC	1147
A	Indices	1165

Table of Contents

1	Programming Languages Supported by GCC ..	1
2	Language Standards Supported by GCC	3
2.1	C Language	3
2.2	C++ Language	5
2.3	Objective-C and Objective-C++ Languages	6
2.4	COBOL Language	7
2.5	Go Language	7
2.6	D language	7
2.7	Modula-2 language	7
2.8	References for Other Languages	7
3	GCC Command Options	9
3.1	Option Summary	9
3.2	Options Controlling the Kind of Output	34
3.3	Compiling C++ Programs	44
3.4	Options Controlling C Dialect	45
3.5	Options Controlling C++ Dialect	52
3.6	Options Controlling Objective-C and Objective-C++ Dialects ..	82
3.7	Options Controlling OpenMP and OpenACC	86
3.8	Options to Control Diagnostic Messages Formatting	88
3.9	Options to Request or Suppress Warnings	101
3.10	Options That Control Static Analysis	170
3.11	Options for Debugging Your Program	189
3.12	Options That Control Optimization	196
3.13	Program Instrumentation Options	245
3.14	Options Controlling the Preprocessor	267
3.15	Passing Options to the Assembler	275
3.16	Options for Linking	276
3.17	Options for Directory Search	282
3.18	Options for Code Generation Conventions	286
3.19	GCC Developer Options	297
3.20	Target-Specific Options	316
3.20.1	AArch64 Options	317
3.20.1.1	-march and -mcpu Feature Modifiers	324
3.20.2	Adapteva Epiphany Options	329
3.20.3	AMD GCN Options	331
3.20.4	ARC Options	333
3.20.5	ARM Options	341
3.20.6	AVR Options	359
3.20.6.1	AVR Optimization Options	364
3.20.6.2	EIND and Devices with More Than 128 Ki Bytes of Flash	366

3.20.6.3	Handling of the RAMPD , RAMPX , RAMPY and RAMPZ Special Function Registers	367
3.20.6.4	AVR Built-in Macros	368
3.20.6.5	AVR Internal Options	371
3.20.7	Blackfin Options	372
3.20.8	C6X Options	374
3.20.9	CRIS Options	375
3.20.10	C-SKY Options	377
3.20.11	Cygwin and MinGW Options	380
3.20.12	Darwin Options	381
3.20.13	DEC Alpha Options	386
3.20.14	eBPF Options	391
3.20.15	FR30 Options	393
3.20.16	FRV Options	393
3.20.17	FT32 Options	396
3.20.18	GNU/Linux Options	396
3.20.19	H8/300 Options	397
3.20.20	HPPA Options	397
3.20.21	IA-64 Options	401
3.20.22	LM32 Options	404
3.20.23	LoongArch Options	405
3.20.24	LynxOS Options	411
3.20.25	M32R/D Options	411
3.20.26	M680x0 Options	412
3.20.27	MCORE Options	418
3.20.28	MicroBlaze Options	419
3.20.29	MIPS Options	420
3.20.30	MMIX Options	435
3.20.31	MN10300 Options	437
3.20.32	Moxie Options	438
3.20.33	MSP430 Options	438
3.20.34	NDS32 Options	441
3.20.35	Nvidia PTX Options	444
3.20.36	OpenRISC Options	445
3.20.37	PDP-11 Options	447
3.20.38	Picolibc Options	447
3.20.39	PowerPC Options	448
3.20.40	PRU Options	448
3.20.41	RISC-V Options	450
3.20.42	RL78 Options	467
3.20.43	IBM RS/6000 and PowerPC Options	469
3.20.44	RX Options	485
3.20.45	S/390 and zSeries Options	488
3.20.46	SH Options	494
3.20.47	Solaris 2 Options	500
3.20.48	SPARC Options	501
3.20.49	Options for System V	507

3.20.50	V850 Options	507
3.20.51	VAX Options	510
3.20.52	Visium Options	511
3.20.53	VMS Options	511
3.20.54	VxWorks Options	512
3.20.55	x86 Options	512
3.20.56	x86 Windows Options	549
3.20.57	Xstormy16 Options	549
3.20.58	Xtensa Options	549
3.20.59	zSeries Options	552
3.21	Environment Variables Affecting GCC	552
3.22	Using Precompiled Headers	555
3.23	C++ Modules	557
3.23.1	Module Mapper	559
3.23.2	Module Preprocessing	560
3.23.3	Compiled Module Interface	561
4	C Implementation-Defined Behavior	563
4.1	Translation	563
4.2	Environment	563
4.3	Identifiers	563
4.4	Characters	564
4.5	Integers	565
4.6	Floating Point	566
4.7	Constant expressions	567
4.8	Arrays and Pointers	567
4.9	Hints	568
4.10	Structures, Unions, Enumerations, and Bit-Fields	568
4.11	Qualifiers	569
4.12	Types	570
4.13	Declarators	570
4.14	Statements	570
4.15	Preprocessing Directives	570
4.16	Library Functions	571
4.17	Architecture	571
4.18	Locale-Specific Behavior	572
5	C++ Implementation-Defined Behavior	573
5.1	Conditionally-Supported Behavior	573
5.2	Exception Handling	573

6	Extensions to the C Language Family.....	575
6.1	Additional Numeric Types.....	575
6.1.1	128-bit Integers.....	575
6.1.2	Double-Word Integers	575
6.1.3	Complex Numbers.....	575
6.1.4	Additional Floating Types.....	577
6.1.5	Half-Precision Floating Point	578
6.1.6	Decimal Floating Types	579
6.1.7	Fixed-Point Types.....	580
6.2	Array, Union, and Struct Extensions.....	581
6.2.1	Arrays of Variable Length	581
6.2.2	Arrays of Length Zero	582
6.2.3	Structures with No Members	584
6.2.4	Unions with Flexible Array Members	584
6.2.5	Structures with only Flexible Array Members	584
6.2.6	Unnamed Structure and Union Fields.....	584
6.2.7	Cast to a Union Type	585
6.2.8	Non-Lvalue Arrays May Have Subscripts.....	586
6.2.9	Non-Constant Initializers	586
6.2.10	Compound Literals.....	587
6.2.11	Designated Initializers.....	588
6.3	Named Address Spaces.....	589
6.3.1	AVR Named Address Spaces.....	589
6.3.2	PRU Named Address Spaces.....	592
6.3.3	RL78 Named Address Spaces	592
6.3.4	x86 Named Address Spaces.....	592
6.3.5	Xtensa Named Address Spaces.....	592
6.4	Attributes Specific to GCC.....	593
6.4.1	Common Attributes	595
6.4.2	Target-Specific Attributes	648
6.4.2.1	AArch64 Attributes.....	649
6.4.2.2	AMD GCN Attributes.....	652
6.4.2.3	ARC Attributes	653
6.4.2.4	ARM Attributes.....	655
6.4.2.5	AVR Attributes	657
6.4.2.6	Blackfin Attributes	661
6.4.2.7	BPF Attributes.....	663
6.4.2.8	C-SKY Attributes	663
6.4.2.9	Epiphany Attributes.....	663
6.4.2.10	H8/300 Attributes.....	664
6.4.2.11	IA-64 Attributes.....	665
6.4.2.12	LoongArch Attributes	666
6.4.2.13	M32R/D Attributes	670
6.4.2.14	m68k Attributes.....	671
6.4.2.15	MicroBlaze Attributes	671
6.4.2.16	Microsoft Windows Attributes	672

6.4.2.17	MIPS Attributes	674
6.4.2.18	MSP430 Attributes	676
6.4.2.19	NDS32 Attributes	678
6.4.2.20	Nvidia PTX Attributes	679
6.4.2.21	PowerPC Attributes	679
6.4.2.22	RISC-V Attributes	682
6.4.2.23	RL78 Attributes	684
6.4.2.24	RX Attributes	684
6.4.2.25	S/390 Attributes	685
6.4.2.26	SH Attributes	686
6.4.2.27	Symbian OS Attributes	687
6.4.2.28	V850 Attributes	688
6.4.2.29	Visium Attributes	688
6.4.2.30	x86 Attributes	688
6.4.2.31	Xstormy16 Attributes	702
6.4.2.32	Xtensa Attributes	702
6.4.3	GNU Attribute Syntax	703
6.5	Pragmas Accepted by GCC	706
6.5.1	AArch64 Pragmas	706
6.5.2	ARM Pragmas	707
6.5.3	LoongArch Pragmas	707
6.5.4	PRU Pragmas	707
6.5.5	RS/6000 and PowerPC Pragmas	707
6.5.6	S/390 Pragmas	707
6.5.7	Darwin Pragmas	708
6.5.8	Solaris Pragmas	708
6.5.9	Symbol-Renaming Pragmas	709
6.5.10	Structure-Layout Pragmas	709
6.5.11	Weak Pragmas	710
6.5.12	Diagnostic Pragmas	710
6.5.13	Visibility Pragmas	712
6.5.14	Push/Pop Macro Pragmas	712
6.5.15	Function Specific Option Pragmas	713
6.5.16	Loop-Specific Pragmas	714
6.6	Thread-Local Storage	715
6.6.1	ISO/IEC 9899:1999 Edits for Thread-Local Storage	715
6.6.2	ISO/IEC 14882:1998 Edits for Thread-Local Storage	716
6.7	OpenMP	717
6.8	OpenACC	718
6.9	An Inline Function is As Fast As a Macro	718
6.10	When is a Volatile Object Accessed?	720
6.11	How to Use Inline Assembly Language in C Code	721
6.11.1	Basic Asm — Assembler Instructions Without Operands ..	721
6.11.2	Extended Asm - Assembler Instructions with C Expression Operands	723
6.11.2.1	Volatile	725
6.11.2.2	Assembler Template	727

6.11.2.3	Output Operands	728
6.11.2.4	Flag Output Operands	731
6.11.2.5	Input Operands	733
6.11.2.6	Clobbers and Scratch Registers	734
6.11.2.7	Goto Labels	737
6.11.2.8	Generic Operand Modifiers	739
6.11.2.9	AArch64 Operand Modifiers	739
6.11.2.10	x86 Operand Modifiers	740
6.11.2.11	x86 Floating-Point asm Operands	741
6.11.2.12	MSP430 Operand Modifiers	742
6.11.2.13	LoongArch Operand Modifiers	743
6.11.2.14	RISC-V Operand Modifiers	744
6.11.2.15	SH Operand Modifiers	744
6.11.3	Constraints for asm Operands	744
6.11.3.1	Simple Constraints	745
6.11.3.2	Multiple Alternative Constraints	747
6.11.3.3	Constraint Modifier Characters	748
6.11.3.4	Constraints for Particular Machines	749
6.11.4	C++11 Constant Expressions instead of String Literals ..	773
6.11.5	Controlling Names Used in Assembler Code	773
6.11.6	Variables in Specified Registers	774
6.11.6.1	Defining Global Register Variables	774
6.11.6.2	Specifying Registers for Local Variables	775
6.11.6.3	Hard Register Constraints	777
6.11.7	Size of an asm	779
6.12	Other Extensions to C Syntax	780
6.12.1	Statements and Declarations in Expressions	780
6.12.2	Locally Declared Labels	782
6.12.3	Labels as Values	783
6.12.4	Nested Functions	784
6.12.5	Referring to a Type with typeof	786
6.12.6	Determining the Number of Elements of Arrays	787
6.12.7	The maximum and minimum representable values of a type ..	787
6.12.8	Support for offsetof	788
6.12.9	Determining the Alignment of Functions, Types or Variables	788
6.12.10	Extensions to enum Type Declarations	788
6.12.11	Support for the _Bool Type	789
6.12.12	Macros with a Variable Number of Arguments	789
6.12.13	Conditionals with Omitted Operands	790
6.12.14	Case Ranges	790
6.12.15	Mixed Declarations, Labels and Code	791
6.12.16	C++ Style Comments	791
6.12.17	Slightly Looser Rules for Escaped Newlines	791
6.12.18	Hex Floats	791
6.12.19	Binary Constants using the '0b' Prefix	791
6.12.20	Dollar Signs in Identifier Names	792

6.12.21	The Character ESC in Constants	792
6.12.22	Raw String Literals	792
6.12.23	Alternate Keywords	792
6.12.24	Function Names as Strings	793
6.13	Extensions to C Semantics	793
6.13.1	Prototypes and Old-Style Function Definitions	794
6.13.2	Arithmetic on void- and Function-Pointers	794
6.13.3	Pointer Arguments in Variadic Functions	794
6.13.4	Pointers to Arrays with Qualifiers Work as Expected	795
6.13.5	Const and Volatile Functions	795

7 Built-in Functions Provided by GCC 797

7.1	Builtins for C Library Functions	797
7.2	Additional Builtins for Numeric Operations	799
7.2.1	Floating-Point Format Builtins	799
7.2.2	Bit Operation Builtins	802
7.2.3	Byte-Swapping Builtins	806
7.2.4	CRC Builtins	807
7.2.5	Built-in Functions to Perform Arithmetic with Overflow Checking	809
7.3	Builtins for Stack Allocation	812
7.4	Nonlocal Gotos	813
7.5	Constructing Function Calls	814
7.6	Getting the Return or Frame Address of a Function	816
7.7	Stack scrubbing internal interfaces	817
7.8	Using Vector Instructions through Built-in Functions	818
7.9	Builtins for Atomic Memory Access	822
7.9.1	Built-in Functions for Memory Model Aware Atomic Operations	822
7.9.2	Legacy __sync Built-in Functions for Atomic Memory Access	827
7.10	Object Size Checking	830
7.10.1	Object Size Checking Built-in Functions	830
7.10.2	Object Size Checking and Source Fortification	831
7.10.2.1	Formatted Output Function Checking	831
7.11	Built-in functions for C++ allocations and deallocations	832
7.12	Other Built-in Functions Provided by GCC	832
7.13	Built-in Functions Specific to Particular Target Machines	843
7.13.1	AArch64 Built-in Functions	843
7.13.2	Alpha Built-in Functions	843
7.13.3	ARC Built-in Functions	844
7.13.4	ARC SIMD Built-in Functions	847
7.13.5	Arm C Language Extensions (ACLE)	851
7.13.6	ARM Floating Point Status and Control Intrinsics	851
7.13.7	ARM ARMv8-M Security Extensions	852
7.13.8	AVR Built-in Functions	852

7.13.9	Blackfin Built-in Functions	854
7.13.10	BPF Built-in Functions.....	854
7.13.11	FR-V Built-in Functions.....	857
7.13.11.1	Argument Types	857
7.13.11.2	Directly-Mapped Integer Functions	858
7.13.11.3	Directly-Mapped Media Functions	858
7.13.11.4	Raw Read/Write Functions.....	860
7.13.11.5	Other Built-in Functions.....	860
7.13.12	LoongArch Base Built-in Functions	861
7.13.12.1	Data Types	861
7.13.12.2	Directly-mapped Builtin Functions.....	861
7.13.12.3	Directly-mapped Division Builtin Functions.....	863
7.13.12.4	Other Builtin Functions	863
7.13.13	LoongArch SX Vector Intrinsics	863
7.13.13.1	SX Data Types.....	863
7.13.13.2	Directly-mapped SX Builtin Functions.....	864
7.13.13.3	Directly-mapped SX Division Builtin Functions...	877
7.13.14	LoongArch ASX Vector Intrinsics.....	877
7.13.14.1	ASX Data Types.....	877
7.13.14.2	Directly-mapped ASX Builtin Functions	878
7.13.14.3	Directly-mapped ASX Division Builtin Functions..	892
7.13.14.4	Directly-mapped SX and ASX Conversion Builtin Functions	892
7.13.15	MIPS DSP Built-in Functions	895
7.13.16	MIPS Paired-Single Support.....	900
7.13.17	MIPS Loongson Built-in Functions	900
7.13.17.1	Paired-Single Arithmetic	902
7.13.17.2	Paired-Single Built-in Functions	903
7.13.17.3	MIPS-3D Built-in Functions	904
7.13.18	MIPS SIMD Architecture (MSA) Support	906
7.13.18.1	MIPS SIMD Architecture Built-in Functions.....	907
7.13.19	Other MIPS Built-in Functions	920
7.13.20	MSP430 Built-in Functions	920
7.13.21	NDS32 Built-in Functions	920
7.13.22	Nvidia PTX Built-in Functions	921
7.13.23	Basic PowerPC Built-in Functions	921
7.13.23.1	Basic PowerPC Built-in Functions Available on all Configurations.....	921
7.13.23.2	Basic PowerPC Built-in Functions Available on ISA 2.05.....	925
7.13.23.3	Basic PowerPC Built-in Functions Available on ISA 2.06.....	927
7.13.23.4	Basic PowerPC Built-in Functions Available on ISA 2.07.....	928
7.13.23.5	Basic PowerPC Built-in Functions Available on ISA 3.0.....	928

7.13.23.6	Basic PowerPC Built-in Functions Available on ISA 3.1	930
7.13.24	PowerPC AltiVec/VSX Built-in Functions	932
7.13.24.1	PowerPC AltiVec Built-in Functions on ISA 2.05 ..	934
7.13.24.2	PowerPC AltiVec Built-in Functions Available on ISA 2.06	943
7.13.24.3	PowerPC AltiVec Built-in Functions Available on ISA 2.07	945
7.13.24.4	PowerPC AltiVec Built-in Functions Available on ISA 3.0	948
7.13.24.5	PowerPC AltiVec Built-in Functions Available on ISA 3.1	953
7.13.24.6	PowerPC AltiVec/VSX Built-in Functions Available on Future ISA	964
7.13.25	PowerPC Hardware Transactional Memory Built-in Functions	965
7.13.25.1	PowerPC HTM Low Level Built-in Functions	965
7.13.25.2	PowerPC HTM High Level Inline Functions	967
7.13.26	PowerPC Atomic Memory Operation Functions	969
7.13.27	PowerPC Matrix-Multiply Assist Built-in Functions	970
7.13.28	PRU Built-in Functions	972
7.13.29	RISC-V Built-in Functions	972
7.13.30	RISC-V Vector Intrinsics	973
7.13.31	CORE-V Built-in Functions	973
7.13.32	RX Built-in Functions	993
7.13.33	S/390 System z Built-in Functions	995
7.13.34	SH Built-in Functions	997
7.13.35	SPARC VIS Built-in Functions	997
7.13.36	TI C6X Built-in Functions	1001
7.13.37	x86 Built-in Functions	1002
7.13.38	x86 Transactional Memory Intrinsics	1028
7.13.39	x86 Control-Flow Protection Intrinsics	1029
8	Extensions to the C++ Language	1031
8.1	When is a Volatile C++ Object Accessed?	1031
8.2	Restricting Pointer Aliasing	1031
8.3	Vague Linkage	1032
8.4	C++ Interface and Implementation Pragmas	1033
8.5	Where's the Template?	1034
8.6	Extracting the Function Pointer from a Bound Pointer to Member Function	1036
8.7	C++-Specific Variable, Function, and Type Attributes	1037
8.8	Function Multiversioning	1040
8.9	Type Traits	1042
8.10	Deprecated Features	1045
8.11	Backwards Compatibility	1045

9	GNU Objective-C Features	1047
9.1	GNU Objective-C Runtime API	1047
9.1.1	Modern GNU Objective-C Runtime API	1047
9.1.2	Traditional GNU Objective-C Runtime API	1048
9.2	+load: Executing Code before main	1048
9.2.1	What You Can and Cannot Do in +load	1049
9.3	Type Encoding	1050
9.3.1	Legacy Type Encoding	1052
9.3.2	@encode	1052
9.3.3	Method Signatures	1053
9.4	Garbage Collection	1053
9.5	Constant String Objects	1054
9.6	compatibility_alias	1055
9.7	Exceptions	1055
9.8	Synchronization	1057
9.9	Fast Enumeration	1057
9.9.1	Using Fast Enumeration	1057
9.9.2	C99-Like Fast Enumeration Syntax	1057
9.9.3	Fast Enumeration Details	1058
9.9.4	Fast Enumeration Protocol	1059
9.10	Messaging with the GNU Objective-C Runtime	1060
9.10.1	Dynamically Registering Methods	1060
9.10.2	Forwarding Hook	1060
10	Binary Compatibility	1063
11	gcov—a Test Coverage Program	1067
11.1	Introduction to gcov	1067
11.2	Invoking gcov	1067
11.3	Using gcov with GCC Optimization	1084
11.4	Brief Description of gcov Data Files	1085
11.5	Data File Relocation to Support Cross-Profiling	1085
11.6	Profiling and Test Coverage in Freestanding Environments ..	1086
11.6.1	Overview	1086
11.6.2	Tutorial	1087
11.6.3	System Initialization Caveats	1091
12	gcov-tool—an Offline Gcda Profile Processing Tool	1093
12.1	Introduction to gcov-tool	1093
12.2	Invoking gcov-tool	1093

13	gcov-dump—an Offline Gcda and Gcno Profile Dump Tool.....	1097
13.1	Introduction to gcov-dump.....	1097
13.2	Invoking gcov-dump.....	1097
14	lto-dump—Tool for dumping LTO object files.....	1099
14.1	Introduction to lto-dump.....	1099
14.2	Invoking lto-dump.....	1099
15	Known Causes of Trouble with GCC.....	1101
15.1	Actual Bugs We Haven't Fixed Yet.....	1101
15.2	Interoperation.....	1101
15.3	Incompatibilities of GCC.....	1103
15.4	Fixed Header Files.....	1106
15.5	Standard Libraries.....	1106
15.6	Disappointments and Misunderstandings.....	1107
15.7	Common Misunderstandings with GNU C++.....	1108
15.7.1	Declare <i>and</i> Define Static Members.....	1108
15.7.2	Name Lookup, Templates, and Accessing Members of Base Classes.....	1109
15.7.3	Temporaries May Vanish Before You Expect.....	1110
15.7.4	Implicit Copy-Assignment for Virtual Bases.....	1111
15.8	Certain Changes We Don't Want to Make.....	1112
15.9	Warning Messages and Error Messages.....	1115
16	Reporting Bugs.....	1117
16.1	Have You Found a Bug?.....	1117
16.2	How and Where to Report Bugs.....	1117
17	How To Get Help with GCC.....	1119
18	Contributing to GCC Development.....	1121
	Funding Free Software.....	1123
	The GNU Project and GNU/Linux.....	1125
	GNU General Public License.....	1127
	GNU Free Documentation License.....	1139
	ADDENDUM: How to use this License for your documents.....	1146

Contributors to GCC	1147
----------------------------------	-------------

Appendix A Indices.....	1165
----------------------------------	-------------

A.1 Option Index	1165
A.2 Attribute Index.....	1203
A.3 Concept and Symbol Index	1207

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++, Fortran, Ada, D, 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 Mercury. 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 COBOL compiler is *gcobol*, 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

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. The ANSI standard, but not the ISO standard, also came with a Rationale document. This standard, in both its forms, is commonly known as *C89*, or occasionally as *C90*, from the dates of ratification. 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 45.

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*. (While in development, drafts of this standard version were referred to as *C9X*.) GCC has substantially complete support for this standard version; see <https://gcc.gnu.org/projects/c-status.html> for details. To select this standard, use `-std=c99` or `-std=iso9899:1999`.

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. (While in development, drafts of this standard version were referred to as *C1X*.) GCC has substantially complete support for this standard, enabled with `-std=c11` or `-std=iso9899:2011`. A version with corrections integrated was prepared in 2017 and published in 2018 as ISO/IEC 9899:2018; it is known as *C17* and is supported with `-std=c17` or `-std=iso9899:2017`; the corrections are also applied with `-std=c11`, and the only difference between the options is the value of `__STDC_VERSION__`.

A fifth version of the C standard, known as *C23*, was published in 2024 as ISO/IEC 9899:2024. (While in development, drafts of this standard version were referred to as *C2X*.) Support for this is enabled with `-std=c23` or `-std=iso9899:2024`.

A further version of the C standard, known as *C2Y*, is under development; experimental and incomplete support for this is enabled with `-std=c2y`.

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 575.

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. Use of the `-std` options listed above disables 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), `-std=gnu11` (for C11 with GNU extensions), `-std=gnu17` (for C17 with GNU extensions) or `-std=gnu23` (for C23 with GNU extensions).

The default, if no C language dialect options are given, is `-std=gnu23`.

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. Since C23, freestanding implementations are required to support a larger range of library facilities, including some functions from other headers.

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 is an example of a program running in a freestanding environment; a program using the facilities of an operating system is an example of a program running in a hosted environment.

GCC aims towards being usable as the compiler for a conforming freestanding or hosted implementation. By default, it acts 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 the compiler for a freestanding environment, use the option `-ffreestanding`; it then defines `__STDC_HOSTED__` to 0 and does 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 45.

GCC generally provides library facilities in headers that do not declare functions with external linkage (which includes the headers required by C11 and before to be provided by freestanding implementations), but not those included in other headers. Additionally, GCC provides `<stdatomic.h>`, even though it declares some functions with external linkage (which are provided in `libatomic`). On a few platforms, some of the headers not declaring functions with external linkage are instead obtained from the OS's C library, which may mean that they lack support for features from more recent versions of the C standard that are supported in GCC's own versions of those headers. On some platforms, GCC provides `<tgmath.h>` (but this implementation does not support interfaces added in C23).

To use the facilities of a hosted environment, and some of the facilities required in a freestanding environment by C23, you need to find them elsewhere (for example, in the GNU C library). See Section 15.5 [Standard Libraries], page 1106.

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`. Contrary to the standards covering `memcpy` GCC expects the case of an exact overlap of source and destination to work and not invoke undefined behavior. Finally, if `__builtin_trap` is used, and the target does not implement the `trap` pattern, then GCC emits a call to `abort`.

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

2.2 C++ Language

GCC supports the original ISO C++ standard published in 1998, and the 2011, 2014, 2017 and mostly 2020 and 2024 revisions.

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, all of which have been implemented in GCC. For details see <https://gcc.gnu.org/projects/cxx-status.html#cxx11>. To select this standard in GCC, use the option `-std=c++11`.

Another revised ISO C++ standard was published in 2014 as ISO/IEC 14882:2014, and is referred to as C++14; before its publication it was sometimes referred to as C++1y. C++14 contains several further changes to the C++ language, all of which have been implemented in GCC. For details see <https://gcc.gnu.org/projects/cxx-status.html#cxx14>. To select this standard in GCC, use the option `-std=c++14`.

The C++ language was further revised in 2017 and ISO/IEC 14882:2017 was published. This is referred to as C++17, and before publication was often referred to as C++1z. GCC supports all the changes in that specification. For further details see <https://gcc.gnu.org/projects/cxx-status.html#cxx17>. Use the option `-std=c++17` to select this variant of C++.

Another revised ISO C++ standard was published in 2020 as ISO/IEC 14882:2020, and is referred to as C++20; before its publication it was sometimes referred to as C++2a. GCC supports most of the changes in the new specification. For further details see <https://gcc.gnu.org/projects/cxx-status.html#cxx20>. To select this standard in GCC, use the option `-std=c++20`.

Yet another revised ISO C++ standard was published in 2024 as ISO/IEC 14882:2024, and is referred to as C++23; before its publication it was sometimes referred to as C++2b. GCC supports most of the changes in the new specification. For further details see <https://gcc.gnu.org/projects/cxx-status.html#cxx23>. To select this standard in GCC, use the option `-std=c++23`.

More information about the C++ standards is available on the ISO C++ committee’s web site at <https://www.open-std.org/jtc1/sc22/wg21/>.

To obtain all the diagnostics required by any of the standard versions described above you should specify `-pedantic` or `-pedantic-errors`, otherwise GCC will allow some non-ISO C++ features as extensions. See Section 3.9 [Warning Options], page 101.

By default, GCC also provides some additional extensions to the C++ language that on rare occasions conflict with the C++ standard. See Section 3.5 [C++ Dialect Options], page 52. Use of the `-std` options listed above disables 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=gnu++98` (for C++98 with GNU extensions), or `-std=gnu++11` (for C++11 with GNU extensions), or `-std=gnu++14` (for C++14 with GNU extensions), or `-std=gnu++17` (for C++17 with GNU extensions), or `-std=gnu++20` (for C++20 with GNU extensions), or `-std=gnu++23` (for C++23 with GNU extensions).

The default, if no C++ language dialect options are given, is `-std=gnu++20`.

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” (<https://www.gnustep.org/resources/documentation/ObjectivCBook.pdf>).

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:

- <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>

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

2.4 COBOL Language

As of the GCC 15 release, GCC supports the ISO COBOL language standard (ISO/IEC 1989:2023). It includes some support for compatibility with other COBOL compilers via the `-dialect` option.

2.5 Go Language

As of the GCC 4.7.1 release, GCC supports the Go 1 language standard, described at <https://go.dev/doc/go1>.

2.6 D language

GCC supports the D 2.0 programming language. The D language itself is currently defined by its reference implementation and supporting language specification, described at <https://dlang.org/spec/spec.html>.

2.7 Modula-2 language

GCC supports the Modula-2 language and is compliant with the PIM2, PIM3, PIM4 and ISO dialects. Also implemented are a complete set of free ISO libraries. It also contains a collection of PIM libraries and some Logitech compatible libraries.

For more information on Modula-2 see <https://gcc.gnu.org/readings.html>. The on-line manual is available at <https://gcc.gnu.org/onlinedocs/gm2/index.html>.

2.8 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.

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. See Section 3.2 [Options Controlling the Kind of Output], page 34.

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

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

The usual way to run GCC is to run the executable called `gcc`, or `machine-gcc` when cross-compiling, or `machine-gcc-version` to run a specific version of GCC. When you compile C++ programs, you should invoke GCC as `g++` instead. See Section 3.3 [Compiling C++ Programs], page 44, for information about the differences in behavior between `gcc` and `g++` when 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` is `-fno-foo`. This manual documents only one of these two forms, whichever one is not the default.

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

See Section A.1 [Option Index], page 1165, 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 34.

```
-c -S -E -o file
-dumpbase dumpbase -dumpbase-ext auxdropsuf
-dumpdir dumpprfx -x language
-v -### --help[=class,...] --target-help --version
-pass-exit-codes -pipe -wrapper
@file -ffile-prefix-map=old=new -fcanon-prefix-map
-fplugin=file -fplugin-arg-name=arg
-fdump-ada-spec[-slim] -fada-spec-parent=unit
-fdump-go-spec=file
--assemble --compile --dumpbase dumpbase
--dumpbase-ext auxdropsuf --dumpdir dumpprfx
--language=language --output=file --pass-exit-codes
--pipe --preprocess --verbose
```

C Language Options

See Section 3.4 [Options Controlling C Dialect], page 45.

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

C++ Language Options

See Section 3.5 [Options Controlling C++ Dialect], page 52.

```
--compile-std-module
-fabi-compat-version=n -fabi-version=n
-fno-access-control -faligned-new=[n]
-fno-assume-sane-operators-new-delete
-fchar8_t -fcheck-new
-fconcepts -fconcepts-diagnostics-depth=n
-fconstexpr-depth=n -fconstexpr-cache-depth=n
-fconstexpr-loop-limit=n -fconstexpr-ops-limit=n
-fcontracts
-fcontract-evaluation-semantic=[ignore|observe|enforce|quick_enforce]
-fcontracts-conservative-ipa -fcontract-checks-outlined
-fcontract-disable-optimized-checks
-fcontracts-client-check=[none|pre|all]
-fcontracts-definition-check=[on|off]
-fcoroutines -fdiagnostics-all-candidates
-fno-elide-constructors
-fno-enforce-eh-specs
-fext-numeric-literals
-fno-gnu-keywords
-fno-immediate-escalation
-fno-implement-inlines
-fimplicit-constexpr
-fno-implicit-inline-templates
-fno-implicit-templates
-fmodule-header[=kind]
-fmodule-implicit-inline
-fno-module-lazy
-fmodule-mapper=specification
```

```

-fmodule-only
-fmodules
-fms-extensions
-fnew-inheriting-ctors
-fnew-ttp-matching
-fno-nonansi-builtins -fnothrow-opt -fno-operator-names
-fno-optional-diags
-fno-pretty-templates -frange-for-ext-temps -freflection
-fno-rtti -fsized-deallocation
-fstrict-enums -fstrong-eval-order[=kind]
-ftemplate-backtrace-limit=n
-ftemplate-depth=n
-fno-threadsafe-statics -fuse-cxa-atexit -fno-use-cxa-get-exception-ptr
-fno-weak -nostdinc++
-fvisibility-inlines-hidden
-fvisibility-ms-compat
-flang-info-include-translate[=header]
-flang-info-include-translate-not
-flang-info-module-cmi[=module]
-stdlib=libstdc++,libc++
-Wabbreviated-auto-in-template-arg
-Wabi-tag -Waligned-new[=kind]
-Wcatch-value -Wcatch-value=n
-Wno-class-conversion -Wclass-memaccess
-Wcomma-subscript -Wconditionally-supported
-Wno-conversion-null -Wctad-maybe-unsupported
-Wctor-dtor-privacy -Wdangling-reference
-Wno-defaulted-function-deleted
-Wno-delete-incomplete
-Wdelete-non-virtual-dtor -Wno-deprecated-array-compare
-Wdeprecated-copy -Wdeprecated-copy-dtor
-Wno-deprecated-enum-enum-conversion -Wno-deprecated-enum-float-conversion
-Wno-deprecated-literal-operator -Wdeprecated-variadic-comma-omission
-Weffc++ -Wno-elaborated-enum-base
-Wno-exceptions -Wno-expose-global-module-tu-local -Wno-external-tu-local
-Wextra-semi -Wno-global-module -Wno-inaccessible-base
-Wno-inherited-variadic-ctor -Wno-init-list-lifetime
-Winvalid-constexpr -Winvalid-imported-macros
-Wno-invalid-offsetof -Wno-literal-suffix
-Wmismatched-new-delete -Wmismatched-tags
-Wmultiple-inheritance -Wnamespaces -Wnarrowing
-Wnoexcept -Wnoexcept-type -Wnon-virtual-dtor
-Wpessimizing-move -Wno-placement-new -Wplacement-new=n
-Wrange-loop-construct -Wredundant-move -Wredundant-tags
-Wreorder -Wregister -Wno-sfinae-incomplete
-Wstrict-null-sentinel -Wno-subobject-linkage -Wtemplates
-Wno-non-c-typedef-for-linkage -Wno-non-template-friend -Wold-style-cast
-Woverloaded-virtual -Wno-pmf-conversions -Wself-move -Wsign-promo
-Wsized-deallocation -Wsuggest-final-methods
-Wsuggest-final-types -Wsuggest-override -Wno-template-body
-Wno-template-id-ctor -Wtemplate-names-tu-local
-Wno-terminate -Wno-vexing-parse -Wvirtual-inheritance
-Wno-virtual-move-assign -Wvolatile

```

Objective-C and Objective-C++ Language Options

See Section 3.6 [Options Controlling Objective-C and Objective-C++ Dialects], page 82.

```

-fconstant-string-class=class-name

```

```

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

```

OpenMP and OpenACC Options

See Section 3.7 [Options Controlling OpenMP and OpenACC], page 86.

```

-offload=arg -offload-options=arg
-fopenacc -fopenacc-dim=geom
-fopenmp -fopenmp-simd -fopenmp-target-simd-clone[=device-type]

```

Diagnostic Message Formatting Options

See Section 3.8 [Options to Control Diagnostic Messages Formatting], page 88.

```

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

```

Warning Options

See Section 3.9 [Options to Request or Suppress Warnings], page 101.

```
-fsyntax-only -fmax-errors=n -Wpedantic
-pedantic-errors -fpermissive
-w -Wextra -Wall -Wabi=n
-Waddress -Wno-address-of-packed-member -Waggregate-return
-Walloc-size -Walloc-size-larger-than=byte-size -Walloc-zero
-Walloca -Walloca-larger-than=byte-size -Wauto-profile
-Wno-aggressive-loop-optimizations
-Warith-conversion
-Warray-bounds -Warray-bounds=n -Warray-compare
-Warray-parameter -Warray-parameter=n
-Wno-attributes -Wattribute-alias=n -Wno-attribute-alias
-Wno-attribute-warning
-Wbidi-chars=[none|unpaired|any|ucn]
-Wbool-compare -Wbool-operation
-Wno-builtin-declaration-mismatch
-Wno-builtin-macro-redefined -Wc90-c99-compat -Wc99-c11-compat
-Wc11-c23-compat -Wc23-c2y-compat
-Wc++-compat -Wc++11-compat -Wc++14-compat -Wc++17-compat
-Wc++20-compat -Wc++26-compat
-Wno-c++11-extensions -Wno-c++14-extensions -Wno-c++17-extensions
-Wno-c++20-extensions -Wno-c++23-extensions
-Wcalloc-transposed-args -Wcannot-profile
-Wcast-align -Wcast-align=strict -Wcast-function-type -Wcast-qual
-Wchar-subscripts
-Wclobbered -Wcomment
-Wcompare-distinct-pointer-types
-Wno-complain-wrong-lang -Wconstant-logical-operand
-Wconversion -Wno-coverage-mismatch -Wno-cpp
-Wdangling-else -Wdangling-pointer -Wdangling-pointer=n
-Wdate-time
-Wno-deprecated -Wno-deprecated-declarations -Wno-designated-init
-Wno-deprecated-openmp
-Wdisabled-optimization
-Wno-discarded-array-qualifiers -Wno-discarded-qualifiers
-Wno-div-by-zero -Wdouble-promotion
-Wduplicated-branches -Wduplicated-cond
-Wempty-body -Wno-endif-labels -Wenum-compare -Wenum-conversion
-Wenum-int-mismatch
-Werror -Werror=* -Wexpansion-to-defined -Wfatal-errors
-Wflex-array-member-not-at-end
-Wfloat-conversion -Wfloat-equal -Wformat -Wformat=2
-Wno-format-contains-nul -Wno-format-diag -Wno-format-extra-args
-Wformat-nonliteral -Wformat-overflow=n
-Wformat-security -Wformat-signedness -Wformat-truncation=n
-Wformat-y2k -Wframe-address
-Wframe-larger-than=byte-size -Wno-free-nonheap-object
-Wheader-guard -Wno-if-not-aligned -Wno-ignored-attributes
-Wignored-qualifiers -Wno-incompatible-pointer-types -Whardened
-Wimplicit -Wimplicit-fallthrough -Wimplicit-fallthrough=n
-Wno-implicit-function-declaration -Wno-implicit-int
-Winfinite-recursion
-Winit-self -Winline -Wno-int-conversion -Wint-in-bool-context
-Wno-int-to-pointer-cast -Wno-invalid-memory-model
-Winvalid-pch -Winvalid-utf8 -Wno-unicode -Wjump-misses-init
-Wkeyword-macro
-Wlarger-than=byte-size -Wleading-whitespace=kind
```

```

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

```

```

-Wzero-length-bounds
-Wzero-init-padding-bits=value
--all-warnings --extra-warnings --no-warnings
--pedantic --pedantic-errors

```

Static Analyzer Options

```

-fanalyzer
-fanalyzer-assume-nothrow
-fanalyzer-call-summaries
-fanalyzer-checker=name
-fno-analyzer-feasibility
-fanalyzer-show-events-in-system-headers
-fno-analyzer-state-merge
-fno-analyzer-state-purge
-fno-analyzer-suppress-followups
-fanalyzer-transitivity
-fno-analyzer-undo-inlining
-fanalyzer-verbose-edges
-fanalyzer-verbose-state-changes
-fanalyzer-verbosity=level
-fdump-analyzer
-fdump-analyzer-callgraph
-fdump-analyzer-exploded-graph
-fdump-analyzer-exploded-nodes
-fdump-analyzer-exploded-nodes-2
-fdump-analyzer-exploded-nodes-3
-fdump-analyzer-exploded-paths
-fdump-analyzer-feasibility
-fdump-analyzer-infinite-loop
-fdump-analyzer-json
-fdump-analyzer-state-purge
-fdump-analyzer-stderr
-fdump-analyzer-supergraph
-fdump-analyzer-untracked
-Wno-analyzer-double-fclose
-Wno-analyzer-double-free
-Wno-analyzer-exposure-through-output-file
-Wno-analyzer-exposure-through-uninit-copy
-Wno-analyzer-fd-access-mode-mismatch
-Wno-analyzer-fd-double-close
-Wno-analyzer-fd-leak
-Wno-analyzer-fd-phase-mismatch
-Wno-analyzer-fd-type-mismatch
-Wno-analyzer-fd-use-after-close
-Wno-analyzer-fd-use-without-check
-Wno-analyzer-file-leak
-Wno-analyzer-free-of-non-heap
-Wno-analyzer-imprecise-fp-arithmetic
-Wno-analyzer-infinite-loop
-Wno-analyzer-infinite-recursion
-Wno-analyzer-jump-through-null
-Wno-analyzer-malloc-leak
-Wno-analyzer-mismatching-deallocation
-Wno-analyzer-mkostemp-redundant-flags
-Wno-analyzer-mktemp-missing-placeholder
-Wno-analyzer-mktemp-of-string-literal
-Wno-analyzer-null-argument
-Wno-analyzer-null-dereference

```

```

-Wno-analyzer-out-of-bounds
-Wno-analyzer-overlapping-buffers
-Wno-analyzer-possible-null-argument
-Wno-analyzer-possible-null-dereference
-Wno-analyzer-putenv-of-auto-var
-Wno-analyzer-shift-count-negative
-Wno-analyzer-shift-count-overflow
-Wno-analyzer-stale-setjmp-buffer
-Wno-analyzer-tainted-allocation-size
-Wno-analyzer-tainted-assertion
-Wno-analyzer-tainted-array-index
-Wno-analyzer-tainted-divisor
-Wno-analyzer-tainted-offset
-Wno-analyzer-tainted-size
-Wno-analyzer-throw-of-unexpected-type
-Wanalyzer-symbol-too-complex
-Wanalyzer-too-complex
-Wno-analyzer-undefined-behavior-ptrdiff
-Wno-analyzer-undefined-behavior-strtok
-Wno-analyzer-unsafe-call-within-signal-handler
-Wno-analyzer-use-after-free
-Wno-analyzer-use-of-pointer-in-stale-stack-frame
-Wno-analyzer-use-of-uninitialized-value
-Wno-analyzer-va-arg-type-mismatch
-Wno-analyzer-va-list-exhausted
-Wno-analyzer-va-list-leak
-Wno-analyzer-va-list-use-after-va-end
-Wno-analyzer-write-to-const
-Wno-analyzer-write-to-string-literal

```

C and Objective-C-only Warning Options

```

-Wbad-function-cast -Wdeprecated-non-prototype -Wfree-labels
-Wmissing-declarations -Wmissing-parameter-name -Wmissing-parameter-type
-Wdeclaration-missing-parameter-type -Wmissing-prototypes
-Wmissing-variable-declarations
-Wmultiple-parameter-fwd-decl-lists
-Wnested-externs -Wold-style-declaration
-Wold-style-definition -Wstrict-prototypes -Wtraditional
-Wtraditional-conversion -Wdeclaration-after-statement -Wpointer-sign

```

Debugging Options

See Section 3.11 [Options for Debugging Your Program], page 189.

```

-g -glevel -gdwarf -gdwarf-version
-gbtf -gctf -gctflevel
-gno-prune-btf -ggdb -gno-record-gcc-switches
-gstrict-dwarf -gas-loc-support -gas-locview-support
-gcodeview -gcolumn-info -gdwarf32 -gdwarf64
-gno-statement-frontiers
-gno-variable-location-views -gvariable-location-views=incompat5
-ginternal-reset-location-views -ginline-points
-gvms -gz[=type]
-gsplit-dwarf -gdescribe-dies
-fdebug-prefix-map=old=new -fdebug-types-section
-fno-eliminate-unused-debug-types
-femit-struct-debug-baseonly -femit-struct-debug-reduced
-femit-struct-debug-detailed[=spec-list]
-fno-eliminate-unused-debug-symbols -femit-class-debug-always
-fno-merge-debug-strings -fno-dwarf2-cfi-asm

```

```
-fvar-tracking -fvar-tracking-assignments -fvar-tracking-uninit
--debug
```

Optimization Options

See Section 3.12 [Options that Control Optimization], page 196.

```
-faggressive-loop-optimizations
-falign-functions[=n[:m:[n2[:m2]]]]
-falign-jumps[=n[:m:[n2[:m2]]]]
-falign-labels[=n[:m:[n2[:m2]]]]
-falign-loops[=n[:m:[n2[:m2]]]]
-fmin-function-alignment=[n]
-fno-allocation-dce -fallow-store-data-races
-fassociative-math -fauto-profile -fauto-profile[=path]
-fauto-profile-inlining -fauto-inc-dec -fbranch-probabilities
-fcaller-saves
-fcombine-stack-adjustments -fconserve-stack
-ffold-mem-offsets
-fcompare-elim -fcprop-registers -fcrossjumping
-fcse-follow-jumps -fcse-skip-blocks -fcx-fortran-rules
-fcx-limited-range -fcx-method
-fdata-sections -fdce -fdelayed-branch
-fdelete-null-pointer-checks -fdep-fusion -fdevirtualize
-fdevirtualize-speculatively -fdevirtualize-at-ltrans -fdse
-fearly-inlining -fexcess-precision=style
-fexpensive-optimizations -fext-dce
-ffast-math -ffat-lto-objects -ffinite-loops
-ffinite-math-only -ffloat-store
-fforward-propagate -ffp-contract=style -ffp-int-builtin-inexact
-ffunction-sections -ffuse-ops-with-volatile-access
-fgcse -fgcse-after-reload -fgcse-las -fgcse-lm -fgraphite-identity
-fgcse-sm -fhoist-adjacent-loads -fif-conversion
-fif-conversion2 -findirect-inlining
-finline-atomics -finline-functions -finline-functions-called-once
-finline-limit=n -finline-small-functions
-finline-stringops[=fn]
-fipa-modref -fipa-cp -fipa-cp-clone
-fipa-bit-cp -fipa-vrp -fipa-pta -fipa-profile -fipa-pure-const
-fipa-reference -fipa-reference-addressable -fipa-reorder-for-locality
-fipa-sra -fipa-stack-alignment
-fipa-icf -fipa-icf-functions -fipa-icf-variables
-fira-algorithm=algorithm
-flate-combine-instructions -flifetime-dse -flive-patching=level
-fira-region=region -fira-hoist-pressure
-fira-loop-pressure -fno-ira-share-save-slots
-fno-ira-share-spill-slots
-fisolate-erroneous-paths-dereference -fisolate-erroneous-paths-attribute
-fivopts -fkeep-inline-functions -fkeep-static-functions
-fkeep-static-consts -flimit-function-alignment -flive-range-shrinkage
-floop-block -floop-interchange -floop-strip-mine
-floop-unroll-and-jam -floop-nest-optimize
-floop-parallelize-all -flra-remat -flto -flto-compression-level=n
-flto-toplevel-asm-heuristics
-flto-partition=alg -flto-incremental=path
-flto-incremental-cache-size=n -fmalloc-dce -fmerge-all-constants
-fmerge-constants -fmodulo-sched -fmodulo-sched-allow-regmoves
-fmove-loop-invariants -fmove-loop-stores -fno-branch-count-reg
-fno-defer-pop -fno-function-cse
-fno-guess-branch-probability -fno-inline -fno-math-errno -fno-peephole
```

```

-fno-peephole2 -fno-printf-return-value -fno-sched-interblock
-fno-sched-spec -fno-signed-zeros
-fno-toplevel-reorder -fno-trapping-math -fno-zero-initialized-in-bss
-fomit-frame-pointer -foptimize-crc -foptimize-sibling-calls
-fpartial-inlining -fpeel-loops -fpredictive-commoning
-fprefetch-loop-arrays
-fprofile-correction
-fprofile-use -fprofile-use=path -fprofile-partial-training
-fprofile-values -fprofile-reorder-functions
-freciprocal-math -free -frename-registers -freorder-blocks
-freorder-blocks-algorithm=algorithm
-freorder-blocks-and-partition -freorder-functions
-frerun-cse-after-loop -freschedule-modulo-scheduled-loops
-frounding-math -fsave-optimization-record
-fsched2-use-superblocks -fsched-pressure
-fsched-spec-load -fsched-spec-load-dangerous
-fsched-stalled-insns-dep[=n] -fsched-stalled-insns[=n]
-fsched-group-heuristic -fsched-critical-path-heuristic
-fsched-spec-insn-heuristic -fsched-rank-heuristic
-fsched-last-insn-heuristic -fsched-dep-count-heuristic
-fschedule-fusion
-fschedule-insns -fschedule-insns2 -fsection-anchors
-fselective-scheduling -fselective-scheduling2
-fsel-sched-pipelining -fsel-sched-pipelining-outer-loops
-fsemantic-interposition -fshrink-wrap -fshrink-wrap-separate
-fsignaling-nans
-fsingle-precision-constant -fsplit-ivs-in-unroller -fsplit-loops
-fspeculatively-call-stored-functions -fsplit-paths
-fsplit-wide-types -fsplit-wide-types-early -fssa-backprop -fssa-phiopt
-fstdarg-opt -fstore-merging -fstrict-aliasing -fipa-strict-aliasing
-fthread-jumps -ftracer -ftree-bit-ccp
-ftree-builtin-call-dce -ftree-ccp -ftree-ch -ftree-coalesce-vars
-ftree-copy-prop -ftree-cselim -ftree-dce -ftree-dominator-opts
-ftree-dse -ftree-forwprop -ftree-fre -fcode-hoisting
-ftree-loop-if-convert -ftree-loop-im
-ftree-phi-prop -ftree-loop-distribution -ftree-loop-distribute-patterns
-ftree-loop-ivcanon -ftree-loop-linear -ftree-loop-optimize
-ftree-loop-vectorize
-ftree-parallelize-loops[=n] -ftree-pre -ftree-partial-pre -ftree-pta
-ftree-reassoc -ftree-scev-cprop -ftree-sink -ftree-slsr -ftree-sra
-ftree-switch-conversion -ftree-tail-merge
-ftree-ter -ftree-vectorize -ftree-vrp -ftrivial-auto-var-init
-funconstrained-commons -funit-at-a-time -funroll-all-loops
-funroll-loops -funsafe-math-optimizations -funswitch-loops
-fipa-ra -fvariable-expansion-in-unroller -fvect-cost-model -fvpt
-fweb -fwhole-program -fwpa -fuse-linker-plugin -fzero-call-used-regs
-0 -00 -01 -02 -03 -0s -Ofast -Og -Oz --optimize

```

Program Instrumentation Options

See Section 3.13 [Program Instrumentation Options], page 245.

```

-p -pg -fprofile-arcs -coverage -ftest-coverage
-fcondition-coverage
-fpath-coverage
-fprofile -fprofile-abs-path
-fprofile-dir=path -fprofile-generate -fprofile-generate=path
-fprofile-info-section -fprofile-info-section=name
-fprofile-note=path -fprofile-prefix-path=path
-fprofile-update=method -fprofile-filter-files=regex

```

```

-fprofile-exclude-files=regex
-fprofile-reproducible=[multithreaded|parallel-runs|serial]
-fsanitize=style -fsanitize-recover -fsanitize-recover=style
-fsanitize-trap -fsanitize-trap=style
-fasan-shadow-offset=number -fsanitize-sections=s1,s2,...
-fsanitize-undefined-trap-on-error -fcf-protection
-fcf-protection=[full|branch|return|none|check]
-fharden-compares -fharden-conditional-branches -fhardened
-fharden-control-flow-redundancy -fhardcfr-skip-leaf
-fhardcfr-check-exceptions -fhardcfr-check-returning-calls
-fhardcfr-check-noreturn-calls=[always|no-xthrow|nothrow|never]
-fstack-protector -fstack-protector-all -fstack-protector-strong
-fstack-protector-explicit -fstack-check
-fstack-limit-register=reg -fstack-limit-symbol=sym
-fno-stack-limit -fsplit-stack
-fstrub=disable -fstrub=strict -fstrub=relaxed
-fstrub=all -fstrub=at-calls -fstrub=internal
-fvtable-verify=[std|preinit|none]
-fvtv-counts -fvtv-debug
-finstrument-functions -finstrument-functions-once
-finstrument-functions-exclude-function-list=sym,sym,...
-finstrument-functions-exclude-file-list=file,file,...
-fprofile-prefix-map=old=new
-fpatchable-function-entry=N[,M]
--coverage --profile

```

Preprocessor Options

See Section 3.14 [Options Controlling the Preprocessor], page 267.

```

-C -CC -Dmacro[=defn]
-dD -dI -dM -dN -dU
-fdebug-cpp -fdirectives-only -fdollars-in-identifiers
-fexec-charset=charset -fextended-identifiers
-finput-charset=charset
-fmacro-prefix-map=old=new -fmax-include-depth=depth
-fno-canonical-system-headers -fpch-deps -fpch-preprocess
-fpreprocessed -ftabstop=width -ftrack-macro-expansion
-fwide-exec-charset=charset -fworking-directory
-H -imacros file -include file
-M -MD -MF -MG -MM -MMD -MP -MQ -MT -Mno-modules
-no-integrated-cpp -P -pthread -remap
-traditional -traditional-cpp -trigraphs
-Umacro -undef
-Wp,option -Xpreprocessor option
--comments --comments-in-macros
--define-macro=macro[=defn]
--dependencies --dump=letters
--imacros=file --include=file
--no-integrated-cpp --no-line-commands
--print-missing-file-dependencies
--traditional --traditional-cpp --trigraphs --trace-includes
--undefine-macro=macro
--user-dependencies --write-dependencies --write-user-dependencies

```

Assembler Options

See Section 3.15 [Passing Options to the Assembler], page 275.

```

-Wa,option -Xassembler option
--for-assembler=option

```

Linker Options

See Section 3.16 [Options for Linking], page 276.

```

object-file-name -flink-libatomic -fuse-ld=linker -llibrary
-nostartfiles -nodefaultlibs -nolibc -nostdlib -nostdlib++
-e entry
-pie -pthread -r -rdynamic
-s -static -static-pie -static-libgcc -static-libstdc++
-static-libasan -static-libhwasan -static-liblsan
-static-libtsan -static-libubsan
-shared -shared-libgcc -symbolic
-T script -Wl,option -Xlinker option
-u symbol
-Tbss=addr -Tdata=addr -Ttext=addr
-N -n -t -Z -z keyword
--entry=entry --for-linker=option
--force-link=symbol --no-standard-library
--pie --static --static-pie --symbolic

```

Directory Options

See Section 3.17 [Options for Directory Search], page 282.

```

-Bprefix -Idir -I-
-idirafter dir
-imacros file -imultilib dir -imultiarch dir
-ipluginidir=dir -iprefix file
-iquote dir -isysroot dir -isystem dir
-iwithprefix dir -iwithprefixbefore dir
-Ldir -no-canonical-prefixes --no-sysroot-suffix
-nostdinc -nostdinc++
--embed-dir=dir --embed-directory=dir
--include-barrier --include-directory=dir
--include-directory-after=dir --include-prefix=prefix
--include-with-prefix=prefix --include-with-prefix-after=prefix
--include-with-prefix-before=prefix
--no-canonical-prefixes --no-standard-includes
--prefix=prefix --sysroot=dir

```

Code Generation Options

See Section 3.18 [Options for Code Generation Conventions], page 286.

```

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

```

Developer Options

See Section 3.19 [GCC Developer Options], page 297.

```

-dletters -dumpspecs -dumpmachine -dumpversion
-dumpfullversion -fcallgraph-info[=su,da]
-fchecking -fchecking=n
-fdbg-cnt-list -fdbg-cnt=counter-value-list
-fdisable-ipa-pass_name
-fdisable-rtl-pass_name
-fdisable-rtl-pass-name=range-list
-fdisable-tree-pass_name
-fdisable-tree-pass-name=range-list
-fdump-debug -fdump-earlydebug
-fdump-noaddr -fdump-unnumbered -fdump-unnumbered-links
-fdump-final-insns[=file]
-fdump-internal-locations
-fdump-ipa-all -fdump-ipa-cgraph -fdump-ipa-inline
-fdump-lang-all
-fdump-lang-switch
-fdump-lang-switch-options
-fdump-lang-switch-options=filename
-fdump-passes
-fdump-rtl-pass -fdump-rtl-pass=filename
-fdump-statistics
-fdump-tree-all
-fdump-tree-switch
-fdump-tree-switch-options
-fdump-tree-switch-options=filename
-fcompare-debug[=opts] -fcompare-debug-second
-fenable-kind-pass
-fenable-kind-pass=range-list
-fira-verbose=n
-flto-report -flto-report-wpa -fmem-report-wpa
-fmem-report -fpref-headers -fpost-ipa-mem-report
-fopt-info -fopt-info-options[=file]
-fmultiflags -fprofile-report
-frandom-seed=string -fsched-verbose=n
-fsel-sched-verbose -fsel-sched-dump-cfg -fsel-sched-pipelining-verbose
-fstats -fstack-usage -ftime-report -ftime-report-details
-fvar-tracking-assignments-toggle -gtoggle
-print-autofdo-gcov-version
-print-file-name=library -print-libgcc-file-name
-print-multi-directory -print-multi-lib -print-multi-os-directory
-print-multiarch
-print-prog-name=program -print-search-dirs -Q
-print-sysroot -print-sysroot-headers-suffix
-save-temps -save-temps=cwd -save-temps=obj
-specs=file -time[=file]
--dump=letters
--print-autofdo-gcov-version
--print-file-name=library --print-libgcc-file-name
--print-multi-directory --print-multi-lib --print-multi-os-directory
--print-multiarch --print-prog-name=program
--print-search-dirs --print-sysroot --print-sysroot-headers-suffix
--save-temps --specs=file
--param name=value

```

Target-Specific Options

See Section 3.20 [Target-Specific Options], page 316.

AArch64 Options (Section 3.20.1 [AArch64 Options], page 317)

```
-mabi=name -mbig-endian -mlittle-endian
-menable-sysreg-checking
-mgeneral-regs-only
-mcmodel=tiny -mcmodel=small -mcmodel=large
-mstrict-align -momit-leaf-frame-pointer
-mtls-dialect=desc -mtls-dialect=traditional
-mtls-size=size -mtp=name
-mfix-cortex-a53-835769 -mfix-cortex-a53-843419
-mlow-precision-recip-sqrt -mlow-precision-sqrt -mlow-precision-div
-mmax-vectorization -mautovec-preference=name
-mpc-relative-literal-loads
-msign-return-address=scope
-mbranch-protection=features
-mharden-sls=opts
-march=name -mcpu=name -mtune=name
-moverride=string
-mstack-protector-guard=guard -mstack-protector-guard-reg=sysreg
-mstack-protector-guard-offset=offset -mtrack-speculation
-moutline-atomics -mearly-ra -mearly-ldp-fusion -mlate-ldp-fusion -mnarrow-gp-writes
-msve-vector-bits=bits
```

Adapteva Epiphany Options (Section 3.20.2 [Adapteva Epiphany Options], page 329)

```
-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 -msmall116
-mfp-mode=mode -mmay-round-for-trunc -mfp-iarith
-mvect-double -max-vect-align=num
-msplit-vecmove-early -mlreg-reg
```

AMD GCN Options (Section 3.20.3 [AMD GCN Options], page 331)

```
-march=gpu -mtune=gpu
-mgang-private-size=bytes
-msram-ecc=[on|off|any]
-mxnack=[on|off|any]
-Wopenacc-dims
```

ARC Options (Section 3.20.4 [ARC Options], page 333)

```
-mbarrel-shifter -mjli-always
-mcpu=cpu -mA6 -mARC600 -mA7 -mARC700
-mdpfp -mdpfp-compact -mdpfp-fast -mno-dpfp-lrsr
-mea -mmul32x16 -mmul64 -matomic
-mnorm -mspfp -mspfp-compact -mspfp-fast -msimd -msoft-float -mswap
-mlock -mswape
-mxy -misize -marclinux -marclinux_prof
-mlong-calls -mmedium-calls -msdata -mirq-ctrl-saved
-mrgf-banked-regs -mlpc-width=width -G num
-mvolatile-cache -mtp-regno=regno
-mbitops -mcmem -munaligned-access
-mauto-modify-reg -mno-brcc
-mcase-vector-pcrel -mno-cond-exec -mearly-cbranchsi
-mindexed-loads -mlra-priority-none
-mlra-priority-compact -mlra-priority-noncompact -mmillicode
-msize-level=level
```

```

-mtune=cpu -mmultcost=num -mcode-density-frame
-mmpy-option=multo
-mdiv-rem -mcode-density -mll64 -mfpu=fpu -mrf16 -mbranch-index

```

ARM Options (Section 3.20.5 [ARM Options], page 341)

```

-mapcs-frame -mapcs
-mabi=name
-mgeneral-regs-only -mno-sched-prolog
-mlittle-endian -mbig-endian
-mbe8 -mbe32
-mfloat-abi=name
-mfp16-format=name
-mthumb-interwork
-mcpu=name -march=name -mfpu=name -mtune=name
-mstructure-size-boundary=n
-mabort-on-noreturn -mlong-calls
-msingle-pic-base -mpic-register=reg
-mpic-data-is-text-relative
-mnop-fun-dllimport
-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
-mfix-cortex-a57-aes-1742098
-mfix-cortex-a72-aes-1655431
-munaligned-access
-mslow-flash-data
-masm-syntax-unified
-mrestrict-it
-mpure-code
-mcmse
-mfix-cmse-cve-2021-35465
-mstack-protector-guard=guard
-mstack-protector-guard-offset=offset
-mfdpic
-mbranch-protection=features
-mvectorize-with-neon-quad -mvectorize-with-neon-double

```

AVR Options (Section 3.20.6 [AVR Options], page 359)

```

-mmcu=mcu -mabsdata -maccumulate-args -masm-len-notes
-mcvt -mbranch-cost=cost -mfuse-add=level -mfuse-move=level
-mfuse-move2 -mcall-prologues -mgas-isr-prologues -mint8 -mflmap
-mdouble=bits -mlong-double=bits -mno-call-main
-mn_flash=size -mfraction-convert-truncate -mno-interrupts
-mmmain-is-OS_task -mrelax -mpmem-wrap-around
-mrmw -mstrict-X -mtiny-stack
-mrodata-in-ram -msplit-bit-shift -msplit-ldst -mshort-calls
-mskip-bug -muse-nonzero-bits -nodevicelib -nodevicespecs
-Wasm-len-notes -Waddr-space-convert -Wmisspelled-isr

```

Blackfin Options (Section 3.20.7 [Blackfin Options], page 372)

```

-mcpu=cpu[-sirevision]
-msim -momit-leaf-frame-pointer
-mspecld-anomaly -mcsync-anomaly
-mlow-64k -mstack-check-l1 -mid-shared-library
-mleaf-id-shared-library

```

```
-mshared-library-id=n
-msep-data -mlong-calls
-mfast-fp -minline-plt -mmulticore -mcorea -mcoreb -msdram
-micplb
```

C6X Options (Section 3.20.8 [C6X Options], page 374)

```
-mbig-endian -mlittle-endian -march=cpu
-msim -msdata=sdata-type -mdsbt -mlong-calls
```

CRIS Options (Section 3.20.9 [CRIS Options], page 375)

```
-mcpu=cpu -march=cpu
-mtune=cpu -mmax-stackframe=n
-metrax4 -metrax100 -mpdebug -mcc-init -mno-side-effects
-mstack-align -mdata-align -mconst-align
-m32-bit -m16-bit -m8-bit -mno-prologue-epilogue
-mbest-lib-options -moverride-best-lib-options
-mtrap-using-break8 -mtrap-unaligned-atomic
-munaligned-atomic-may-use-library
-sim -sim2
-mmul-bug-workaround
```

C-SKY Options (Section 3.20.10 [C-SKY Options], page 377)

```
-march=arch -mcpu=cpu
-mbig-endian -mlittle-endian
-mfpu=fpu -mdouble-float -mfdivdu
-mfloat-abi=name
-melrw -mistack -mmp -mcp -mcache -msecurity -mtrust
-mdsp -medsp -mvdsp
-mdiv -msmart -mhigh-registers -manchor
-mpushpop -mmultiple-stld -mconstpool -mstack-size -mccrt
-mbranch-cost=n -msched-prolog -msim
```

Cygwin and MinGW Options (Section 3.20.11 [Cygwin and MinGW Options], page 380)

```
-mconsole -mcrtdll=library -mdll
-mnop-fun-dllimport -mthreads
-municode -mwin32 -mwindows -fno-set-stack-executable
-fwritable-relocated-rdata -mpe-aligned-commons
-muse-libstdc-wrappers
```

Darwin Options (Section 3.20.12 [Darwin Options], page 381)

```
-all_load -allowable_client -arch name
-arch_errors_fatal -asm_macosx_version_min=version
-bind_at_load -bundle -bundle_loader
-client_name -compatibility_version -current_version
-dead_strip
-dependency-file -dylib_file -dylinker -dylinker_install_name
-dynamic -dynamiclib -exported_symbols_list
-fapple-kext -fconstant-cfstrings -ffix-and-continue
-filelist -findirect-data -flat_namespace -force_cpusubtype_ALL
-force_flat_namespace -framework name -gfull -gused
-headerpad_max_install_names -iframework
-image_base -init symbol-name -install_name -keep_private_externs
-matt-stubs -mconstant-cfstrings -mdynamic-no-pic
-mfix-and-continue -mkernel -mmacosx-version-min=version
-mone-byte-bool -msymbol-stubs -mtarget-linker[=]version
-nofaultexport -nofaulttrpaths
-pagezero_size -preload -read_only_relocs
-sectalign -sectcreate
```

```

-seg_addr_table
-segladdr -segaddr
-segprot -segs_read_only_addr -segs_read_write_addr
-sub_library -sub_umbrella
-twolevel_namespace -twolevel_namespace_hints
-umbrella -undefined -unexported_symbols_list
-weak_framework_name -weak_reference_mismatches
-whatsloaded -whyload
-F -ObjC -ObjC++ -Wnonportable-cfstrings

```

DEC Alpha Options (Section 3.20.13 [DEC Alpha Options], page 386)

```

-mno-fp-regs -msoft-float
-mieee -mieee-with-inexact -mieee-conformant
-mfp-trap-mode=mode -mfp-rounding-mode=mode
-mtrap-precision=mode -mbuild-constants
-mcpu=cpu-type -mtune=cpu-type
-mbwx -mmax -mfix -mcix
-msafe-bwa -msafe-partial
-mfloat-vax -mfloat-ieee
-mexplicit-relocs -msmall-data -mlarge-data
-msmall-text -mlarge-text
-mmemory-latency=time
-mtls-kernel -mtls-size=bitsize
-mlong-double-128 -mlong-double-64

```

eBPF Options (Section 3.20.14 [eBPF Options], page 391)

```

-mbig-endian -mlittle-endian
-mframe-limit=bytes -mxbpf -mco-re -mjmpext -mjmp32
-malu32 -mv3-atomics -mbswap -msdiv -msmov -mcpu=version
-masm=dialect -minline-memops-threshold=bytes
-Wco-re

```

FR30 Options (Section 3.20.15 [FR30 Options], page 393)

```

-msmall-model -mno-lsim

```

FRV Options (Section 3.20.16 [FRV Options], page 393)

```

-mgpr-32 -mgpr-64 -mfpr-32 -mfpr-64
-mhard-float -msoft-float
-malloc-cc -mfixed-cc -mdword -mdouble -mmedia -mmuladd
-mfdpic -minline-plt -mgprel-ro -multilib-library-pic
-mlinked-fp -mlong-calls -malign-labels
-mlibrary-pic -macc-4 -macc-8
-mpack -mno-eflags -mno-cond-move
-mno-optimize-membar -mno-scc -mno-cond-exec
-mno-vliw-branch -mno-multi-cond-exec -mno-nested-cond-exec
-mtomcat-stats
-mTLS -mtls
-mcpu=cpu

```

FT32 Options (Section 3.20.17 [FT32 Options], page 396)

```

-msim -mnodiv -mft32b -mcompress -mnopm

```

GNU/Linux Options (Section 3.20.18 [GNU/Linux Options], page 396)

```

-mglibc -muclibc -mmusl -mbionic -mandroid
-tno-android-cc -tno-android-ld

```

H8/300 Options (Section 3.20.19 [H8/300 Options], page 397)

```

-mrelax -mh -ms -mn -msx -ms2600
-mquickcall -mslowbyte -mexr -mint32 -malign-300

```

HPPA Options (Section 3.20.20 [HPPA Options], page 397)

```

-march=architecture-type
-mno-atomic-libcalls
-mcaller-copies -mdisable-fpregs -mdisable-indexing
-mordered -mfast-indirect-calls -mgas -mgnu-ld -mhp-ld
-mfixed-range=register-range
-mcoherent-ldcw -mlinker-opt -mlong-calls
-mlong-load-store
-mno-space-regs -msoft-float -mpa-risc-1-0
-mpa-risc-1-1 -mpa-risc-2-0 -mportable-runtime
-mschedule=cpu-type -msoft-mult -msio -mwsio
-munix=unix-std -nolibdld -static -threads

```

IA-64 Options (Section 3.20.21 [IA-64 Options], page 401)

```

-mbig-endian -mlittle-endian -mgnu-as -mgnu-ld -mno-pic
-mvolatile-asm-stop -mregister-names -msdata
-mconstant-gp -mauto-pic
-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-stop-bits-after-every-cycle -msched-count-spec-in-critical-path
-msel-sched-dont-check-control-spec -msched-fp-mem-deps-zero-cost
-msched-max-memory-insns-hard-limit -msched-max-memory-insns=max-insns

```

LM32 Options (Section 3.20.22 [LM32 Options], page 404)

```

-mbarrel-shift-enabled -mdivide-enabled -mmultiply-enabled
-msign-extend-enabled -muser-enabled

```

LoongArch Options (Section 3.20.23 [LoongArch Options], page 405)

```

-march=arch-type -mtune=tune-type -mabi=base-abi-type
-mfpu=fpu-type -msimd=simd-type
-msoft-float -msingle-float -mdouble-float -mlsx -mlasx
-mbranch-cost=n -maddr-reg-reg-cost=n -mcheck-zero-division
-mbreak-code=code
-mcond-move-int -mcond-move-float
-memcpy -mstrict-align -G num
-mmax-inline-memcpy-size=n
-mexplicit-relocs=style -mexplicit-relocs -mno-explicit-relocs
-mdirect-extern-access
-mcmodel=code-model -mrelax -mpass-mrelax-to-as
-mrecip -mrecip=opt -mfrecipe -mdiv32
-mlam-bh -mlamcas -mld-seq-sa
-mscq -mtls-dialect=opt
-mannotate-tablejump

```

LynxOS Options (Section 3.20.24 [LynxOS Options], page 411)

```

-mshared -mthreads -mlegacy-threads

```

M32R/D Options (Section 3.20.25 [M32R/D Options], page 411)

```

-m32r2 -m32rx -m32r
-mdebug
-malign-loops
-missue-rate=number
-mbranch-cost=number
-mmodel=code-size-model-type
-msdata=sdata-type
-mno-flush-func -mflush-func=name
-mno-flush-trap -mflush-trap=number
-G num

```

M680x0 Options (Section 3.20.26 [M680x0 Options], page 412)

```

-march=arch -mcpu=cpu -mtune=tune
-m68000 -m68020 -m68020-40 -m68020-60 -m68030 -m68040
-m68060 -m68302 -m68332 -m68851
-mcpu32 -mfidoa -m5200 -m5206e -m528x -m5307 -m5407
-mcfv4e -mbitfield -mc68000 -mc68020
-mrtd -mdiv -mshort
-mhard-float -m68881 -msoft-float -mpcrel
-malign-int -mstrict-align -msep-data
-mshared-library-id=n -mid-shared-library
-mxgot -mxtls -mlong-jump-table-offsets

```

MCore Options (Section 3.20.27 [MCore Options], page 418)

```

-mhardlit -mdiv -mrelax-immediates
-mwide-bitfields
-m4byte-functions -mcallgraph-data
-mslow-bytes -mno-lsim
-mlittle-endian -mbig-endian -m210 -m340 -mstack-increment

```

MicroBlaze Options (Section 3.20.28 [MicroBlaze Options], page 419)

```

-msoft-float -mhard-float -msmall-divides -mcpu=cpu
-mmemcpy -mxl-soft-mul -mxl-soft-div -mxl-barrel-shift
-mxl-pattern-compare -mxl-gp-opt
-mxl-multiply-high -mxl-float-convert -mxl-float-sqrt
-mbig-endian -mlittle-endian -mxl-reorder -mxl-mode-app-model
-mxl-prefetch -mpic-data-is-text-relative

```

MIPS Options (Section 3.20.29 [MIPS Options], page 420)

```

-EL -EB -mel -meb -march=arch -mtune=arch
-mips1 -mips2 -mips3 -mips4 -mips32 -mips32r2 -mips32r3 -mips32r5
-mips32r6 -mips64 -mips64r2 -mips64r3 -mips64r5 -mips64r6
-mips16 -mmips16e2 -mflip-mips16
-minterlink-compressed -minterlink-mips16
-mabi=abi -mabicalls -mshared -mplt -mxgot
-mgp32 -mgp64 -mfp32 -mfpxx -mfp64 -mhard-float -msoft-float
-mno-float -msingle-float -mdouble-float -modd-spreg
-mabs=mode -mnan=encoding
-mdsp -mdspr2 -mmcu -meva -mvirt -mxpa -mcrc -mginv
-mmips -mmips
-mloongson-mmi -mloongson-ext -mloongson-ext2
-mfpu=fpu-type
-msmartmips -mpaired-single -mdmx -mips3d -mmt -mllsc
-mlong64 -mlong32 -msym32
-Gnum -mno-local-sdata -mno-extern-sdata -mno-gopt
-membedded-data -muninit-const-in-rodata
-mcode-readable=setting -mno-data-in-code -mcode-xonly
-msplit-addresses -mexplicit-relocs -mexplicit-relocs=release
-mno-check-zero-division -mdivide-traps -mdivide-breaks
-mno-load-store-pairs

```

```

-mstrict-align -mno-unaligned-access
-mmemcpy -mlong-calls
-mmadd -mimadd -mno-fused-madd -nocpp
-mfix-24k -mfix-r4000 -mfix-r4400 -mfix-r5900
-mfix-r10000 -mfix-rm7000 -mfix-vr4120 -mfix-vr4130 -mfix-sb1
-mfix4300 -mr10k-cache-barrier=setting
-mflush-func=func -mno-flush-func
-mbranch-cost=num -mbranch-likely
-mcompact-branches=policy
-mno-fp-exceptions -mvr4130-align -msynci -mno-lxc1-sxc1 -mno-madd4
-mno-relax-pic-calls -mmcount-ra-address
-mframe-header-opt

```

MMIX Options (Section 3.20.30 [MMIX Options], page 435)

```

-mlibfuncs -mepsilon -mabi=gnu -mabi=mmixware
-mzero-extend -mknuthdiv -mtoplevel-symbols
-melf -mbranch-predict -mbase-addresses
-msingle-exit -mset-data-start=address
-mset-program-start=address -mno-set-program-start

```

MN10300 Options (Section 3.20.31 [MN10300 Options], page 437)

```

-mmult-bug -mno-mult-bug
-mam33 -mam33-2 -mam34
-mtune=cpu-type
-mno-return-pointer-on-d0
-mno-crt0 -mrelax -mno-liw -mno-setlb

```

Moxie Options (Section 3.20.32 [Moxie Options], page 438)

```

-meb -mel -mmul.x -mno-crt0

```

MSP430 Options (Section 3.20.33 [MSP430 Options], page 438)

```

-msim -masm-hex -mmc=name -mlarge -msmall -mrelax
-mwarn-mcu -mwarn-devices-csv
-mcode-region=where -mdata-region=where
-muse-lower-region-prefix
-msilicon-errata=name[,name...]
-msilicon-errata-warn=name[,name...]
-mhwmult=type -minrt -mtiny-printf -mmax-inline-shift=n

```

NDS32 Options (Section 3.20.34 [NDS32 Options], page 441)

```

-mbig-endian -mlittle-endian -EB -EL
-mabi=name -mfloat-abi=name
-mreduced-regs -mfull-regs
-malways-align -malign-functions
-mfp-as-gp -mcmov -mhw-abs
-mext-perf -mext-perf2 -mext-string -mext-dsp
-mext-fpu-fma -mext-fpu-sp -mext-fpu-dp
-mv3push -m16-bit -mvh
-misr-vector-size=num -misr-secure=num
-mcache-block-size=num
-march=arch -mcpu=cpu
-mconfig-fpu=num -mconfig-mul=type
-mconfig-register-ports=kind
-mcmodel=code-model
-mctor-dtor -mrelax -mrelax-hint
-msched-prolog-epilog -mno-ret-in-naked-func
-malways-save-lp -munaligned-access -minline-asm-r15

```

Nvidia PTX Options (Section 3.20.35 [Nvidia PTX Options], page 444)

```

-m64 -march=arch -misa=arch -march-map=arch

```

```
-mptx=version -mmainkernel -moptimize
-msoft-stack -msoft-stack-reserve-local=size
-muniform-simt -mgomp
```

OpenRISC Options (Section 3.20.36 [OpenRISC Options], page 445)

```
-mboard=name -mhard-mul -mhard-div
-msoft-mul -msoft-div
-msoft-float -mhard-float -mdouble-float -munordered-float
-mcmov -mrdr -mrdr -msex -msfimm -mshftimm
-mcmodel=code-model
```

PDP-11 Options (Section 3.20.37 [PDP-11 Options], page 447)

```
-mfpu -msoft-float -mac0 -m40 -m45 -m10
-mint32 -mint16
-msplit -munix-asm -mdec-asm -mgnu-asm -mlra
```

Picolibc Options (Section 3.20.38 [Picolibc Options], page 447)

```
--oslib=library --crt0=[none|minimal|hosted|semihost]
--printf=[d|f|l|i|m] --scanf=[d|f|l|i|m]
```

PowerPC Options See RS/6000 and PowerPC Options.

PRU Options (Section 3.20.40 [PRU Options], page 448)

```
-mmcuc=mcu -minrt -mno-relax -mloop
-mmul -mfillzero -mabi=variant
```

RISC-V Options (Section 3.20.41 [RISC-V Options], page 450)

```
-mbranch-cost=N-instruction
-mabi=ABI-string
-mfddiv -mdiv
-mno-fence-tso
-misa-spec=ISA-spec-string
-march=[ISA|Profile|Profile_ISA|processor-string]
-mcpu=processor-string -mtune=processor-string
-mpreferred-stack-boundary=num
-msmall-data-limit=N-bytes
-msave-restore -mno-shorten-memrefs
-mstrict-align -mscalar-strict-align -mno-vector-strict-align
-mzilsd-word-align -mzilsd-strict-align
-mcmodel=medlow -mcmodel=medany -mcmodel=large
-mexplicit-relocs -mrelax -mriscv-attribute
-malign-data=type
-mbig-endian -mlittle-endian
-mstack-protector-guard=guard -mstack-protector-guard-reg=reg
-mstack-protector-guard-offset=offset
-mcsr-check -momit-leaf-frame-pointer -mmovcc
-mno-inline-atomics -mno-inline-strlen
-mno-inline-strcmp -mno-inline-strncmp
-mstringop-strategy=strategy
-mtls-dialect=desc -mtls-dialect=trad
-mrvv-vector-bits=value -mrvv-max-lmul=value
-madjust-lmul-cost -mmax-vectorization -mno-autovec-segment
```

RL78 Options (Section 3.20.42 [RL78 Options], page 467)

```
-msim -mallregs -mrelax -mes0
-mmul=none -mmul=g13 -mmul=g14 -mmul=rl78
-mcpu=g10 -mcpu=g13 -mcpu=g14 -mcpu=rl78
-mg10 -mg13 -mg14 -mrl78
-msave-mduc-in-interrupts
```

RS/6000 and PowerPC Options (Section 3.20.43 [RS/6000 and PowerPC Options], page 469)

```

-mcpu=cpu-type
-mtune=cpu-type
-mcmodel=code-model -mprofile-kernel
-mpowerpc64
-maltivec
-mpowerpc-gpopt -mpowerpc-gfxopt -mmfcrf -mpopcmtb -mpopcmtb
-mfprnd -mcmpb -mhard-dfp
-mfull-toc -mminimal-toc -mno-fp-in-toc -mno-sum-in-toc
-maix64 -maix32 -m64 -m32 -mxl-compat -mpe
-malign-power -malign-natural
-msoft-float -mhard-float -mmultiple -mupdate
-mavoid-indexed-addresses
-mfused-madd -mbit-align -mbit-word
-mstrict-align -mrelocatable -mrelocatable-lib
-mlittle -mlittle-endian -mbig -mbig-endian
-mdynamic-no-pic -msingle-pic-base
-mprioritize-restricted-insns=priority
-msched-costly-dep=dependence_type
-minsert-sched-nops=scheme
-mcall-aixdesc -mcall-eabi -mcall-freebsd
-mcall-linux -mcall-netbsd -mcall-openbsd
-mcall-sysv -mcall-sysv-eabi -mcall-sysv-noeabi
-mtraceback=traceback_type
-maix-struct-return -msvr4-struct-return
-mabi=abi-type -msecure-plt -mbss-plt
-msplit-patch-nops
-mregnames -mlongcall -mpltseq -mtls-markers
-mblock-move-inline-limit=num
-mblock-compare-inline-limit=num
-mblock-compare-inline-loop-limit=num
-mblock-ops-unaligned-vsx
-mstring-compare-inline-limit=num
-misel -mvsx -mvrsave -mmulhw -mdlmzb -mprototype
-msim -mmvme -mads -myellowknife -memb -meabi -msdata
-msdata=opt -mreadonly-in-sdata -mvxworks -G num
-mrecip -mrecip=opt -mno-recip -mrecip-precision
-mveclibabi=type -mfriz
-mpointers-to-nested-functions -msave-toc-indirect -mpower8-fusion
-mcrypto -mhtm -mqquad-memory -mqquad-memory-atomic
-mcompat-align-parm
-mfloat128 -mfloat128-hardware
-mgnu-attribute
-mstack-protector-guard=guard -mstack-protector-guard-reg=reg
-mstack-protector-guard-offset=offset -mprefixed
-mpcrel -mma -mrop-protect -mprivileged
-mno-splat-word-constant -mno-splat-float-constant
-mno-ieee128-constant -mno-warn-altivec-long

```

RX Options (Section 3.20.44 [RX Options], page 485)

```

-m64bit-doubles -m32bit-doubles -fpu -nofpu
-mcpu=name
-mbig-endian-data -mlittle-endian-data
-msmall-data-limit=N
-msim -mas100-syntax -mrelax
-mmax-constant-size=N -mint-register=N
-mpid -mno-allow-string-insns -mjsr

```

```
-mno-warn-multiple-fast-interrupts -msave-acc-in-interrupts
-mlra
```

S/390 and zSeries Options (Section 3.20.45 [S/390 and zSeries Options], page 488)

```
-mtune=cpu-type -march=cpu-type
-mhard-float -msoft-float -mhard-dfp
-mlong-double-64 -mlong-double-128
-mbackchain -mpacked-stack
-msmall-exec -mmvcle
-m64 -m31 -mdebug -mesa -mzarch
-mhtm -mvx -mzvector
-mtpf-trace -mtpf-trace-skip -mmain
-mfused-madd -mno-fused-madd
-mwarn-framesize=framesize -mwarn-dynamicstack
-mstack-size=stack-size -mno-stack-size
-mstack-guard=stack-guard -mno-stack-guard
-mstack-protector-guard=guard
-mstack-protector-guard-record
-mhotpatch=pre-halfwords,post-halfwords
-mno-pic-data-is-text-relative
-mindirect-branch=choice
-mindirect-branch-jump=choice -mindirect-branch-call=choice
-mfunction-return=choice
-mfunction-return-mem=choice -mfunction-return-reg=choice
-mindirect-branch-table
-mfentry -mrecord-mcount -mnop-mcount
-mpreserve-args -munaligned-symbols
```

SH Options (Section 3.20.46 [SH Options], page 494)

```
-m1 -m2 -m2e
-m2a -m2a-nofpu -m2a-single -m2a-single-only
-m3 -m3e
-m4 -m4-nofpu -m4-single -m4-single-only
-m4-100 -m4-100-nofpu -m4-100-single -m4-100-single-only
-m4-200 -m4-200-nofpu -m4-200-single -m4-200-single-only
-m4-300 -m4-300-nofpu -m4-300-single -m4-300-single-only
-m4-340 -m4-400 -m4-500
-m4a -m4a1 -m4a-nofpu -m4a-single -m4a-single-only
-mb -ml -mdalign -mrelax
-mbigtable -mbitops -mfmovd -mrenesas -mnomacsave
-mieee -misize -minline-ic_invalidate
-mprefergot -musermode -multcost=number -mdiv=strategy
-mdivsi3_libfunc=name -mfixed-range=register-range
-maccumulate-outgoing-args
-matomic-model=atomic-model
-mbranch-cost=num -mzdcbranch
-mcbranch-force-delay-slot
-mfsca -mfsrra
-mpretend-cmove -mfdpic -mtas -mlra
```

Solaris 2 Options (Section 3.20.47 [Solaris 2 Options], page 500)

```
-mclear-hwcap -mno-clear-hwcap -mimpure-text -mno-impure-text
-gsctf
```

SPARC Options (Section 3.20.48 [SPARC Options], page 501)

```
-mcpu=cpu-type
-mtune=cpu-type
-mcmodel=code-model
```

```

-mmemory-model=mem-model
-m32 -m64 -mptr32 -mptr64 -mapp-regs
-mfaster-structs -mflat
-mfpu -mhard-float -msoft-float
-mhard-quad-float -msoft-quad-float
-mstack-bias -mstd-struct-return
-munaligned-doubles -muser-mode
-mv8plus -mvis
-mvis2 -mvis3 -mvis3b -mvis4 -mvis4b
-mcbcond -mfmaf -mfsmuld -mpopc -msubxc
-mfix-at697f -mfix-ut699 -mfix-ut700 -mfix-gr712rc

```

System V Options (Section 3.20.49 [System V Options], page 507)

```
-YP,paths -Ym,dir
```

V850 Options (Section 3.20.50 [V850 Options], page 507)

```

-mlong-calls -mep
-mprolog-function -mspace
-mtda=n -msda=n -mzda=n
-mv850 -mv850e3v5 -m850e2v4 -mv850e2v3
-mv850e2 -mv850e1 -mv850es -mv850e
-mdisable-callt -mrelax -mlong-jumps
-msoft-float -mhard-float -mloop
-mrh850-abi -mghs -mgcc-abi
-m8byte-align -mbig-switch -mapp-regs -msmall-sld
-mno-strict-align -mjump-tables-in-data-section

```

VAX Options (Section 3.20.51 [VAX Options], page 510)

```

-munix -mgnu -md -md-float -mg -mg-float -mlra
-mvaxc-alignment -mqmath

```

Visium Options (Section 3.20.52 [Visium Options], page 511)

```

-mdebug -msim -mfpu -mhard-float -msoft-float
-mcpu=cpu-type -mtune=cpu-type -msv-mode -muser-mode

```

VMS Options (Section 3.20.53 [VMS Options], page 511)

```

-mvms-return-codes -mdebug-main=prefix -mmalloc64
-mpointer-size=size

```

VxWorks Options (Section 3.20.54 [VxWorks Options], page 512)

```

-mrtp -msmp -mvthreads -non-static -Bstatic -Bdynamic
-Xbind-lazy -Xbind-now

```

x86 Options (Section 3.20.55 [x86 Options], page 512)

```

-mtune=cpu-type -march=cpu-type
-mtune-ctrl=feature-list -mdump-tune-features -mno-default
-mfpmath=unit
-masm=diect -mno-fancy-math-387
-mno-fp-ret-in-387 -m80387 -mhard-float -msoft-float -mieee-fp
-mrtd -malign-double
-mpreferred-stack-boundary=num
-mincoming-stack-boundary=num
-mcld -mcx16 -msahf -mmovbe -mcrc32 -mmwait
-mrecip -mrecip=opt
-mvzeroupper -mstv -mprefer-avx128 -mprefer-vector-width=opt
-mpartial-vector-fp-math
-mmove-max=bits -mstore-max=bits
-mnoreturn-no-callee-saved-registers
-mmmx -msse -msse2 -msse3 -mssse3 -msse4.1 -msse4.2 -msse4 -mavx
-mavx2 -mavx512f -mavx512cd -mavx512vl

```

```

-mavx512bw -mavx512dq -mavx512ifma -mavx512vbmi -msha -maes
-mpclmul -mfsqgsbase -mrdrnd -mf16c -mfma -mpconfig -mwbnoinvd
-mptwrite -mclflushopt -mclwb -mxsavec -mxsaves
-msse4a -m3dnow -m3dnowa -mpopcmt -mabm -mbmi -mtbm -mfma4 -mxop
-madx -mlzcnt -mbmi2 -mfxsr -mxsave -mxsaveopt -mrtm -mhle -mlwp
-mmwaitx -mclzero -mpku -mgfni -mvaes -mwaitpkg
-mshstk -mmanual-endbr -mcet-switch -mforce-indirect-call
-mavx512vbmi2 -mavx512bf16 -menqcmd
-mvpclmulqdq -mavx512bitalg -mmovdiri -mmovdir64b -mavx512vpopcntdq
-mavx512vnni -mprfchw -mrdpid
-mrdseed -msgx -mavx512vp2intersect -mserialize -mtsxlctrk
-mamx-tile -mamx-int8 -mamx-bf16 -muint8 -mhreset
-mavxvnni -mamx-fp8 -mavx512fp16 -mavxifma -mavxvnniint8
-mavxneconvert -mcmpccxadd -mamx-fp16 -mprefetchi -mraoint
-mamx-complex -mavxvnniint16 -msm3 -msha512 -msm4 -mapxf
-musermsr -mavx10.1 -mavx10.2 -mamx-avx512 -mamx-tf32 -mmovrs
-mamx-movrs -mavx512bmm -mcldemote -mms-bitfields
-mno-align-stringops -minline-all-stringops
-minline-stringops-dynamically -mstringop-strategy=alg
-mkl -mwidekl
-mmemcpy-strategy=strategy -mmemset-strategy=strategy
-mpush-args -maccumulate-outgoing-args -m128bit-long-double
-m96bit-long-double -mlong-double-64 -mlong-double-80 -mlong-double-128
-mregparm=num -msseregparm
-mveclibabi=type -mvect8-ret-in-mem
-mpc32 -mpc64 -mpc80 -mdaz-ftz -mstackrealign -mstack-arg-probe
-momit-leaf-frame-pointer -mno-red-zone -mno-tls-direct-seg-refs
-mcmodel=code-model -mabi=name -maddress-mode=mode
-m32 -m64 -mx32 -m16 -miamcu -mlarge-data-threshold=num
-msse2avx -mfentry -mrecord-mcount -mnop-mcount -m8bit-idiv
-minstrument-return=type -mrecord-return
-mfentry-name=name -mfentry-section=name
-mskip-rax-setup
-mavx256-split-unaligned-load -mavx256-split-unaligned-store
-malign-data=type -mstack-protector-guard=guard
-mstack-protector-guard-reg=reg
-mstack-protector-guard-offset=offset
-mstack-protector-guard-symbol=symbol
-mgeneral-regs-only -mcall-ms2sysv-xlogues -mtls-dialect=type
-mrelax-cmpxchg-loop
-mindirect-branch=choice -mfunction-return=choice
-mindirect-branch-register -mharden-sls=choice
-mindirect-branch-cs-prefix -mapx-inline-asm-use-gpr32
-mgather -mscatter
-mneeded -mno-direct-extern-access
-munroll-only-small-loops -mdispatch-scheduler -mlam=choice

```

x86 Windows Options See Cygwin and MinGW Options.

Xstormy16 Options (Section 3.20.57 [Xstormy16 Options], page 549)

```
-msim
```

Xtensa Options (Section 3.20.58 [Xtensa Options], page 549)

```

-mconst16 -mforce-no-pic -mno-serialize-volatile
-mtext-section-literals -mauto-litpools -mno-target-align
-mlongcalls -mabi=abi-type
-mextra-l32r-costs=cycles -mstrict-align -mforce-l32

```

zSeries Options See S/390 and zSeries Options.

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:

<i>file.c</i>	C source code that must be preprocessed.
<i>file.i</i>	C source code that should not be preprocessed.
<i>file.ii</i>	C++ source code that should not be preprocessed.
<i>file.m</i>	Objective-C source code. Note that you must link with the <code>libobjc</code> library to make an Objective-C program work.
<i>file.mi</i>	Objective-C source code that should not be preprocessed.
<i>file.mm</i>	
<i>file.M</i>	Objective-C++ source code. Note that you must link with the <code>libobjc</code> library to make an Objective-C++ program work. Note that ‘.M’ refers to a literal capital M.
<i>file.mii</i>	Objective-C++ source code that should not be preprocessed.
<i>file.h</i>	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 <code>-fdump-ada-spec</code> switch).
<i>file.cc</i>	
<i>file.cp</i>	
<i>file.cxx</i>	
<i>file.cpp</i>	
<i>file.CPP</i>	
<i>file.c++</i>	
<i>file.C</i>	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.
<i>file.mm</i>	
<i>file.M</i>	Objective-C++ source code that must be preprocessed.
<i>file.mii</i>	Objective-C++ source code that should not be preprocessed.
<i>file.hh</i>	
<i>file.H</i>	
<i>file.hp</i>	
<i>file.hxx</i>	
<i>file.hpp</i>	
<i>file.HPP</i>	
<i>file.h++</i>	
<i>file.tcc</i>	C++ header file to be turned into a precompiled header or Ada spec.

<i>file.f</i>	
<i>file.for</i>	
<i>file.ftn</i>	
<i>file.fi</i>	Fixed form Fortran source code that should not be preprocessed.
<i>file.F</i>	
<i>file.FOR</i>	
<i>file.fpp</i>	
<i>file.FPP</i>	
<i>file.FTN</i>	Fixed form Fortran source code that must be preprocessed (with the traditional preprocessor).
<i>file.f90</i>	
<i>file.f95</i>	
<i>file.f03</i>	
<i>file.f08</i>	
<i>file.fii</i>	Free form Fortran source code that should not be preprocessed.
<i>file.F90</i>	
<i>file.F95</i>	
<i>file.F03</i>	
<i>file.F08</i>	Free form Fortran source code that must be preprocessed (with the traditional preprocessor).
<i>file.cob</i>	
<i>file.COB</i>	
<i>file.cbl</i>	
<i>file.CBL</i>	COBOL source code.
<i>file.go</i>	Go source code.
<i>file.d</i>	D source code.
<i>file.di</i>	D interface file.
<i>file.dd</i>	D documentation code (Ddoc).
<i>file.ads</i>	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 <i>specs</i> .
<i>file.adb</i>	Ada source code file containing a library unit body (a subprogram or package body). Such files are also called <i>bodies</i> .
<i>file.s</i>	Assembler code.
<i>file.S</i>	
<i>file.sx</i>	Assembler code that must be preprocessed.
<i>other</i>	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`

`--language=language`

`--language language`

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

```
c c-header cpp-output
c++ c++-header c++-system-header c++-user-header c++-cpp-output
c++-system-module
objective-c objective-c-header objective-c-cpp-output objc-cpp-output
objective-c++ objective-c++-header objective-c++-cpp-output
objc++-cpp-output
assembler assembler-with-cpp
ada adascil adaway
cobol
d
f77 f77-cpp-input f95 f95-cpp-input
go
modula-2 modula-2-cpp-output
rust
algol68
lto
```

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

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

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

`-c`

`--compile`

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`

`--assemble`

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

--preprocess

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

--output=file

--output file

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

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

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

```
gcc -c foo.c ...
```

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

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

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

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

A linker output will instead prefix aux and dump outputs:

```
gcc foo.c bar.c -o dir/foobar ...
```

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

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

```
gcc foo.c -o dir/foo ...
```

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

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

`-dumpbase` *dumpbase*

`--dumpbase` *dumpbase*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-dumpbase-ext *auxdropsuf*

--dumpbase-ext *auxdropsuf*

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

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

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

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

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

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

-dumpdir *dumpppfx*

--dumpdir *dumpppfx*

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

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

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

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

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

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

It defaults to the location of the output file, unless the output file is a special file like */dev/null*. Options *-save-temps=cwd* and *-save-temps=obj* override

this default, just like an explicit `-dumpdir` option. In case multiple such options are given, the last one prevails:

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

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

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

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

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

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

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

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

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

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

When compiling and then linking from a single input file, the linker output base name will only be appended to the default *dumpppfx* as above if it does not share the base name with the single input file name. This has been covered in single-input linking cases above, but not with an explicit `-dumpdir` that inhibits the combination, even if overridden by `-save-temps=*`:

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

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

Summing it all up for an intuitive though slightly imprecise data flow: the primary output name is broken into a directory part and a basename part; *dumpppfx* is set to the former, unless overridden by `-dumpdir` or `-save-temps=*`, and *dumpbase* is set to the latter, unless overridden by `-dumpbase`. If there are

multiple inputs or linking, this *dumpbase* may be combined with *dumppfx* and taken from each input file. Auxiliary output names for each input are formed by combining *dumppfx*, *dumpbase* minus suffix, and the auxiliary output suffix; dump output names are only different in that the suffix from *dumpbase* is retained.

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

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

- `-v`
- `--verbose` Print (on standard error output) the commands executed to run the stages of compilation. Also print the version number of the compiler driver program and of the preprocessor and the compiler proper.
- `###` Like `-v` except the commands are not executed and arguments are quoted unless they contain only alphanumeric characters or `./-_`. This is useful for shell scripts to capture the driver-generated command lines.
- `--help` Print (on the standard output) a description of the command-line options understood by `gcc`. If the `-v` option is also specified then `--help` is also passed on to the various processes invoked by `gcc`, so that they can display the command-line options they accept. If the `-Wextra` option has also been specified (prior to the `--help` option), then command-line options that have no documentation associated with them are also displayed.
- `--target-help` Print (on the standard output) a description of target-specific command-line options for each tool. For some targets extra target-specific information may also be printed.
- `--help={class|[^]qualifier}[,...]`
 Print (on the standard output) a description of the command-line options understood by the compiler that fit into all specified classes and qualifiers. These are the supported classes:
 - `'optimizers'` Display all of the optimization options supported by the compiler.
 - `'warnings'` Display all of the options controlling warning messages produced by the compiler.
 - `'target'` Display target-specific options. Unlike the `--target-help` option however, target-specific options of the linker and assembler are not

displayed. This is because those tools do not currently support the extended `--help=` syntax.

- `'params'` Display the values recognized by the `--param` option.
- `language` Display the options supported for *language*, where *language* is the name of one of the languages supported in this version of GCC. If an option is supported by all languages, one needs to select `'common'` class.
- `'common'` Display the options that are common to all languages.

These are the supported qualifiers:

- `'undocumented'`
Display only those options that are undocumented.
- `'joined'` Display options taking an argument that appears after an equal sign in the same continuous piece of text, such as: `'--help=target'`.
- `'separate'`
Display options taking an argument that appears as a separate word following the original option, such as: `'-o output-file'`.

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

```
--help=target,undocumented
```

The sense of a qualifier can be inverted by prefixing it with the `^` 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 so much that there is nothing to display. One case where it does work, however, is when one of the classes is *target*. For example, to display all the target-specific optimization options, use:

```
--help=target,optimizers
```

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

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

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

```
% gcc -Q -mabi=2 --help=target -c
The following options are target specific:
-mabi=
```

```

-mabort-on-noreturn      [disabled]
-mapcs                   [disabled]

```

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

```
-Q -O2 --help=optimizers
```

Alternatively you can discover which binary optimizations are enabled by `-O3` by using:

```

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

```

`--version`

Display the version number and copyrights of the invoked GCC.

`-pass-exit-codes`

`--pass-exit-codes`

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

`-pipe`

`--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.

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

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

This invokes all subprograms of `gcc` under ‘`gdb --args`’, thus the invocation of `cc1` is ‘`gdb --args cc1 ...`’.

`-ffile-prefix-map=old=new`

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

`-fcanon-prefix-map`

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

-fplugin=*name.so*

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

-fplugin-arg-*name-key=value*

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

-fdump-ada-spec[*-slim*]

For C and C++ source and include files, generate corresponding Ada specs. See Section “Generating Ada Bindings for C and C++ headers” in *GNAT User's Guide*, which provides detailed documentation on this feature.

-fada-spec-parent=*unit*

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

-fdump-go-spec=*file*

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

@*file*

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

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

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’`; preprocessed C++ files use the suffix `‘.ii’`; and C++20 module interface units sometimes use `‘.ixx’`, `‘.cppm’`, `‘.cxxm’`, `‘.c++m’`, or `‘.ccm’`.

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

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

When you compile C++ programs, you may specify many of the same command-line options that you use for compiling programs in any language; or command-line options

meaningful for C and related languages; or options that are meaningful only for C++ programs. See Section 3.4 [Options Controlling C Dialect], page 45, for explanations of options for languages related to C. See Section 3.5 [Options Controlling C++ Dialect], page 52, 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
--ansi In C mode, this is equivalent to **-std=c90**. In C++ mode, it is equivalent to **-std=c++98**.

-std= Determine the language standard. See Chapter 2 [Language Standards Supported by GCC], page 3, 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’. When a base standard is specified, the compiler accepts all programs following that standard plus those using GNU extensions that do not contradict it. For example, **-std=c90** turns off certain features of GCC that are incompatible with ISO C90, such as the **asm** and **typeof** keywords, but not other GNU extensions that do not have a meaning in ISO C90, such as omitting the middle term of a **?:** expression. On the other hand, when a GNU dialect of a standard is specified, all features supported by the compiler are enabled, even when those features change the meaning of the base standard. As a result, some strict-conforming programs may be rejected. The particular standard is used by **-Wpedantic** to identify which features are GNU extensions given that version of the standard. For example **-std=gnu90 -Wpedantic** warns about C++ style ‘//’ comments, while **-std=gnu99 -Wpedantic** does not.

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

‘c90’

‘c89’

‘iso9899:1990’

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

‘iso9899:199409’

ISO C90 as modified in amendment 1.

‘c99’

‘c9x’

‘iso9899:1999’

‘iso9899:199x’

ISO C99. This standard is substantially completely supported, modulo bugs and floating-point issues (mainly but not entirely relating to optional C99 features from Annexes F and G). See

<https://gcc.gnu.org/projects/c-status.html> for more information. The names ‘c9x’ and ‘iso9899:199x’ are deprecated.

‘c11’	
‘c1x’	
‘iso9899:2011’	ISO C11, the 2011 revision of the ISO C standard. This standard is substantially completely supported, modulo bugs, floating-point issues (mainly but not entirely relating to optional C11 features from Annexes F and G) and the optional Annexes K (Bounds-checking interfaces) and L (Analyzability). The name ‘c1x’ is deprecated.
‘c17’	
‘c18’	
‘iso9899:2017’	
‘iso9899:2018’	ISO C17, the 2017 revision of the ISO C standard (published in 2018). This standard is same as C11 except for corrections of defects (all of which are also applied with <code>-std=c11</code>) and a new value of <code>__STDC_VERSION__</code> , and so is supported to the same extent as C11.
‘c23’	
‘c2x’	
‘iso9899:2024’	ISO C23, the 2023 revision of the ISO C standard (published in 2024). The name ‘c2x’ is deprecated.
‘c2y’	The next version of the ISO C standard, still under development. The support for this version is experimental and incomplete.
‘gnu90’	
‘gnu89’	GNU dialect of ISO C90 (including some C99 features).
‘gnu99’	
‘gnu9x’	GNU dialect of ISO C99. The name ‘gnu9x’ is deprecated.
‘gnu11’	
‘gnu1x’	GNU dialect of ISO C11. The name ‘gnu1x’ is deprecated.
‘gnu17’	
‘gnu18’	GNU dialect of ISO C17.
‘gnu23’	
‘gnu2x’	GNU dialect of ISO C23. This is the default for C code. The name ‘gnu2x’ is deprecated.
‘gnu2y’	The next version of the ISO C standard, still under development, plus GNU extensions. The support for this version is experimental and incomplete. The name ‘gnu2x’ is deprecated.
‘c++98’	
‘c++03’	The 1998 ISO C++ standard plus the 2003 technical corrigendum and some additional defect reports. Same as <code>-ansi</code> for C++ code.

<code>'gnu++98'</code> <code>'gnu++03'</code>	GNU dialect of <code>-std=c++98</code> .
<code>'c++11'</code> <code>'c++0x'</code>	The 2011 ISO C++ standard plus amendments. The name <code>'c++0x'</code> is deprecated.
<code>'gnu++11'</code> <code>'gnu++0x'</code>	GNU dialect of <code>-std=c++11</code> . The name <code>'gnu++0x'</code> is deprecated.
<code>'c++14'</code> <code>'c++1y'</code>	The 2014 ISO C++ standard plus amendments. The name <code>'c++1y'</code> is deprecated.
<code>'gnu++14'</code> <code>'gnu++1y'</code>	GNU dialect of <code>-std=c++14</code> . The name <code>'gnu++1y'</code> is deprecated.
<code>'c++17'</code> <code>'c++1z'</code>	The 2017 ISO C++ standard plus amendments. The name <code>'c++1z'</code> is deprecated.
<code>'gnu++17'</code> <code>'gnu++1z'</code>	GNU dialect of <code>-std=c++17</code> . The name <code>'gnu++1z'</code> is deprecated.
<code>'c++20'</code> <code>'c++2a'</code>	The 2020 ISO C++ standard plus amendments. C++20 modules support is still experimental and needs to be enabled with <code>-fmodules</code> option. The name <code>'c++2a'</code> is deprecated.
<code>'gnu++20'</code> <code>'gnu++2a'</code>	GNU dialect of <code>-std=c++20</code> . This is the default for C++ code. C++20 modules support is still experimental and needs to be enabled with <code>-fmodules</code> option. The name <code>'gnu++2a'</code> is deprecated.
<code>'c++23'</code> <code>'c++2b'</code>	The 2023 ISO C++ standard plus amendments (published in 2024). Support is experimental, and could change in incompatible ways in future releases. The name <code>'c++2b'</code> is deprecated.
<code>'gnu++23'</code> <code>'gnu++2b'</code>	GNU dialect of <code>-std=c++23</code> . Support is experimental, and could change in incompatible ways in future releases. The name <code>'gnu++2b'</code> is deprecated.
<code>'c++2c'</code> <code>'c++26'</code>	The next revision of the ISO C++ standard, planned for 2026. Support is highly experimental, and will almost certainly change in incompatible ways in future releases.
<code>'gnu++2c'</code> <code>'gnu++26'</code>	GNU dialect of <code>-std=c++2c</code> . Support is highly experimental, and will almost certainly change in incompatible ways in future releases.

-aux-info filename

Output to the given filename prototyped declarations for all functions declared and/or defined in a translation unit, including those in header files. This option is silently ignored in any language other than C.

Besides declarations, the file indicates, in comments, the origin of each declaration (source file and line), whether the declaration was implicit, prototyped or unprototyped ('I', 'N' for new or 'O' for old, respectively, in the first character after the line number and the colon), and whether it came from a declaration or a definition ('C' or 'F', respectively, in the following character). In the case of function definitions, a K&R-style list of arguments followed by their declarations is also provided, inside comments, after the declaration.

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

In C++, `inline` is a standard keyword and is not affected by this switch. You may want to use the `-fno-gnu-keywords` flag instead, which disables `typeof` but not `asm` and `inline`. In C99 mode (`-std=c99` or `-std=gnu99`), this switch only affects the `asm` and `typeof` keywords, since `inline` is a standard keyword in ISO C99. In C23 mode (`-std=c23` or `-std=gnu23`), this switch only affects the `asm` keyword, since `typeof` is a standard keyword in ISO C23.

-fno-builtin**-fno-builtin-function**

Don't recognize built-in functions that do not begin with `'__builtin_'` as prefix. See Section 7.1 [Library Builtins], page 797, 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))
```

-fcond-mismatch

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

-ffreestanding

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

See Chapter 2 [Language Standards Supported by GCC], page 3, for details of freestanding and hosted environments.

-fgimple

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

-fgnu-tm

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

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

-fgnu89-inline

The option **-fgnu89-inline** tells GCC to use the traditional GNU semantics for **inline** functions when in C99 mode. See Section 6.9 [An Inline Function is As Fast As a Macro], page 718. Using this option is roughly equivalent to adding the **gnu_inline** function attribute to all inline functions (see Section 6.4.1 [Common Attributes], page 595).

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

The preprocessor macros **__GNUC_GNU_INLINE__** and **__GNUC_STDC_INLINE__** may be used to check which semantics are in effect for **inline** functions. See Section "Common Predefined Macros" in *The C Preprocessor*.

-fhosted

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

-flax-vector-conversions

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

-fms-extensions

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

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

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

Some cases of unnamed fields in structures and unions are only accepted with this option. See Section 6.2.6 [Unnamed struct/union fields within structs/unions], page 584, for details.

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

-fpermitted-float-eval-methods=style

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

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

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

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

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

-fdeps-file=file

Where to write structured dependency information.

-fdeps-format=format

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

-fdeps-target=file

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

-fplan9-extensions

Accept some non-standard constructs used in Plan 9 code.

This enables **-fms-extensions**, permits passing pointers to structures with anonymous fields to functions that expect pointers to elements of the type of the field, and permits referring to anonymous fields declared using a typedef. See Section 6.2.6 [Unnamed struct/union fields within structs/unions], page 584, for details. This is only supported for C, not C++.

-fsigned-bitfields**-funsigned-bitfields****-fno-signed-bitfields****-fno-unsigned-bitfields**

These options control whether a bit-field is signed or unsigned, when the declaration does not use either **signed** or **unsigned**. By default, such a bit-field is signed, because this is consistent: the basic integer types such as **int** are signed types.

-fsigned-char**-funsigned-char**

-fsigned-char lets the type **char** be signed, like **signed char**, while **-funsigned-char** lets the type **char** be unsigned, like **unsigned char**.

Note that **-fsigned-char** 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**.

Each target supported by GCC has a default for what **char** should be, which is typically specified in the processor-specific ABI document for that target. 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. These options let you make such a program work with the opposite default.

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

-fstrict-flex-arrays (C and C++ only)**-fstrict-flex-arrays=level** (C and C++ only)

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

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

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

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

You can control this behavior for a specific trailing array field of a structure by using the variable attribute `strict_flex_array` attribute (see Section 6.4.1 [Common Attributes], page 595).

The `-fstrict_flex_arrays` option interacts with the `-Wstrict-flex-arrays` option. See Section 3.9 [Warning Options], page 101, for more information.

`-fsso-struct=endianness`

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

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

3.5 Options Controlling C++ Dialect

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

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

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

Some options for compiling C programs, such as `-std`, are also relevant for C++ programs. See Section 3.4 [Options Controlling C Dialect], page 45.

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

`--compile-std-module`

Build the compiled module interfaces (see Section 3.23 [C++ Modules], page 557) for the `<bits/stdc++.h>` header unit and the `'std'` and `'std.compat'` modules before compiling any source files explicitly specified on the command line. This is intended to be useful for building simple programs that use `'import std;'` with a single command.

`-fabi-version=n`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Version 19, which first appeared in G++ 14, fixes manglings of structured bindings to include ABI tags, handling of cv-qualified `[[no_unique_address]]` members, and adds mangling of C++20 constraints on function templates.

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

Version 21, which first appeared in G++ 16, fixes unnecessary captures in noexcept lambdas (c++/119764), layout of a base class with all explicitly defaulted constructors (c++/120012), and mangling of class and array objects with implicitly zero-initialized non-trailing subobjects (c++/121231).

See also `-Wabi`.

`-fabi-compat-version=n`

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

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

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

`-fno-access-control`

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

`-faligned-new`

`-faligned-new=alignment`

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

This flag is enabled by default for `-std=c++17`.

`-fno-assume-sane-operators-new`

The C++ standard allows replacing the global `new`, `new[]`, `delete` and `delete[]` operators, though a lot of C++ programs don't replace them and just use the implementation provided version. Furthermore, the C++ standard allows omitting those calls if they are made from `new` or `delete` expressions (and by extension the same is assumed if `__builtin_operator_new` or `__builtin_operator_delete` functions are used). This option allows control over some optimizations around calls to those operators. With `-fassume-sane-operators-new-delete` option GCC may assume that calls to the replaceable global operators from `new` or `delete` expressions or from `__builtin_operator_new` or `__builtin_operator_delete` calls don't read or modify any global variables or variables whose address could escape to the operators (global state; except for `errno` for the `new` and `new[]` operators). This allows most optimizations across those calls and is something that the implementation provided operators satisfy unless `malloc` implementation details are observable in the code or unless `malloc` hooks are

used, but might not be satisfied if a program replaces those operators. This behavior is enabled by default. With `-fno-assume-sane-operators-new-delete` option GCC must assume all these calls (whether from new or delete expressions or called directly) may read and write global state unless proven otherwise (e.g. when GCC compiles their implementation). Use this option if those operators are or may be replaced and code needs to expect such behavior.

`-fchar8_t`
`-fno-char8_t`

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

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

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

and introduces new signatures for user-defined literals:

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

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

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

`-fcheck-new`

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

-fconcepts

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

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

-fconcepts-diagnostics-depth=n

Specify maximum error replay depth during recursive diagnosis of a constraint satisfaction failure. The default is 1.

-fconstexpr-depth=n

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

-fconstexpr-cache-depth=n

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

-fconstexpr-fp-exception

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

```
constexpr float inf = 1./0.; // OK with -fconstexpr-fp-exception
```

-fconstexpr-loop-limit=n

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

-fconstexpr-ops-limit=n

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

-fcontracts

Enable support for the C++ Contracts feature, as specified in the C++26 working draft (N5003).

On violation of an enforced or observed contract (see **-fcontract-evaluation-semantic** below), a violation handler is called; the standard library provides a default handler that emits information about the contract that failed.

Users can replace the default violation handler by defining

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

-fcontract-evaluation-semantic=semantic

Set the semantic with which contracts will be evaluated to *semantic*.

Available values for the evaluation mode are:

'**ignored**' The contract checks will be elided, with no checking added at either compile or runtime.

'**observed**' The contract checks are performed (at runtime) and, if one fails, a handler is called that allows actions such as logging or emitting run-time warnings. When the handler returns, the execution of the code will continue. At compile-time the checks are performed for **constexpr** evaluations, in this case a diagnostic is emitted if the check fails.

'**enforce**' The contract checks are performed and the handler is called if one fails. When the handler returns the execution of the code is terminated preventing it from continuing past the failed case. At compile-time the checks are applied to **constexpr** and, if one fails, the program is ill-formed; an error will be emitted.

'**quick_enforce**' The contract checks are performed (at runtime) and, if one fails, the execution is terminated immediately (without calling any handler). At compile-time there is no difference in behaviour between this option and the **enforce** case, since no handler is invoked at compile time.

-fcontracts-conservative-ipa

This prevents inter-procedural analysis from taking action when the body of an inline function is visible in a given TU. This allows for the case that contract evaluation conditions are permitted to differ between TUs which means that such actions would be potentially incorrect.

-fcontract-checks-outlined

By default, contract checks for **pre** and **post** condition checks are expanded in-line into function bodies. This option causes a small function to be created instead (containing the checks) that is called in the relevant position. This permits different criteria to be applied to the compilation of the checking and the remainder of the function body.

-fcontract-disable-optimized-checks

When **pre** and **post** condition checks are outlined (using **-fcontract-checks-outlined**), this option disables the optimisation steps for those functions which can avoid unwanted elision of checking steps.

-fcontracts-client-check=mode

This option enables insertion of checks at call sites (caller-side checking).

The value of *mode* determines which contract checks are made: **'none'** No caller/client side checking (default)

'pre' Preconditions are checked at the caller/client side.

'all' Both pre and post-conditions are checked at the caller/client side.

-fcontracts-definition-check=mode

This option introduces the ability to disable callee/definition-side checks, which are otherwise enabled by default when the contracts feature is active.

The variable *mode* has the values **'on'** and **'off'**.

-fcoroutines

Enable support for the C++ coroutines extension. With **-std=c++20** and above, coroutines are part of the language standard, so **-fcoroutines** defaults to on.

-fdiagnostics-all-candidates

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

-fdump-lang-**-fdump-lang-switch****-fdump-lang-switch-options****-fdump-lang-switch-options=filename**

Control the dumping of C++-specific information. The *options* and *filename* portions behave as described in the **-fdump-tree** option. The following *switch* values are accepted:

'all' Enable all of the below.

'class' Dump class hierarchy information. Virtual table information is emitted unless **'slim'** is specified.

'module' Dump module information. Options **lineno** (locations), **graph** (reachability), **blocks** (clusters), **uid** (serialization), **alias** (mergeable), **asmname** (Elrond), **eh** (mapper) & **vops** (macros) may provide additional information.

'raw' Dump the raw internal tree data.

'tinst' Dump the sequence of template instantiations, indented to show the depth of recursion. The **lineno** option adds the source location where the instantiation was triggered, and the **details** option also dumps pre-instantiation substitutions such as those performed during template argument deduction.

Lines in the .tinst dump start with **'I'** for an instantiation, **'S'** for another substitution, and **'R[IS]'** for the reopened context of a deferred instantiation.

-fno-elide-constructors

The C++ standard allows an implementation to omit creating a temporary that is only used to initialize another object of the same type. Specifying this option

disables that optimization, and forces G++ to call the copy constructor in all cases. This option also causes G++ to call trivial member functions which otherwise would be expanded inline.

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

-fno-enforce-eh-specs

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

-fextern-tls-init

-fno-extern-tls-init

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

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

-ffold-simple-inlines

-fno-fold-simple-inlines

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

-fno-gnu-keywords

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

-fno-immediate-escalation

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

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

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

void g(int i)
{
    f (3);
}
```

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

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

`-fimplicit-constexpr`

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

`-fno-implicit-templates`

Never emit code for non-inline templates that are instantiated implicitly (i.e. by use); only emit code for explicit instantiations. If you use this option, you must take care to structure your code to include all the necessary explicit instantiations to avoid getting undefined symbols at link time. See Section 8.5 [Template Instantiation], page 1034, 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 need the same set of explicit instantiations.

`-fno-implement-inlines`

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

`-fmodules`

`-fno-modules`

Enable support for C++20 modules (see Section 3.23 [C++ Modules], page 557). The `-fno-modules` is usually not needed, as that is the default. Even though this is a C++20 feature, it is not currently implicitly enabled by selecting that standard version.

`-fmodule-header`

`-fmodule-header=user`

`-fmodule-header=system`

Compile a header file to create an importable header unit.

`-fmodule-implicit-inline`

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

`-fno-module-lazy`

Disable lazy module importing and module mapper creation.

`-fmodule-mapper=[hostname]:port[?ident]`

`-fmodule-mapper=|program[?ident] args...`

`-fmodule-mapper==socket[?ident]`

`-fmodule-mapper=<>[inout][?ident]`

`-fmodule-mapper=<in>out[?ident]`

`-fmodule-mapper=file[?ident]`

An oracle to query for module name to filename mappings. If unspecified the `CXX_MODULE_MAPPER` environment variable is used, and if that is unset, an in-process default is provided.

`-fmodule-only`

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

`-fms-extensions`

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

`-fnew-inheriting-ctors`

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

`-fnew-ttp-matching`

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

`-fno-nonansi-builtins`

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

`-fnothrow-opt`

Treat a `throw()` exception specification as if it were a `noexcept` specification to reduce or eliminate the text size overhead relative to a function with no excep-

tion specification. If the function has local variables of types with non-trivial destructors, the exception specification actually makes the function smaller because the EH cleanups for those variables can be optimized away. The semantic effect is that an exception thrown out of a function with such an exception specification results in a call to `terminate` rather than `unexpected`.

-fno-operator-names

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

-fno-optional-diags

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

-fno-pretty-templates

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

-frange-for-ext-temps

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

-freflection

Enable experimental C++26 Reflection.

-fno-rtti

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

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

-fsized-deallocation

Enable the built-in global declarations

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

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

`-fstrict-enums`

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

`-fstrong-eval-order`

`-fstrong-eval-order=kind`

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

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

`-fstrong-eval-order=none` is equivalent to `-fno-strong-eval-order`.

`-ftemplate-backtrace-limit=n`

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

`-ftemplate-depth=n`

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

`-fno-threadsafe-statics`

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

`-fuse-cxa-atexit`

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

`-fno-use-cxa-get-exception-ptr`

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

-fvisibility-inlines-hidden

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

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

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

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

Explicitly instantiated inline methods are unaffected by this option as their linkage might otherwise cross a shared library boundary. See Section 8.5 [Template Instantiation], page 1034.

-fvisibility-ms-compat

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

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

1. It sets the default visibility to `hidden`, like `-fvisibility=hidden`.
2. Types, but not their members, are not hidden by default.
3. The One Definition Rule is relaxed for types without explicit visibility specifications that are defined in more than one shared object: those declarations are permitted if they are permitted when this option is not used.

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

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

-fno-weak

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

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

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

-nostdinc++

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

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

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

-flang-info-module-cmi**-flang-info-module-cmi=module**

Inform of Compiled Module Interface pathnames. The first will note all read CMI pathnames. The *module* form will note reading a specific module's CMI. *module* may be a named module or a header-unit (the latter indicated by either being a pathname containing directory separators or enclosed in <> or "").

-stdlib=libstdc++,libc++

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

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

-Wabi-tag (C++ and Objective-C++ only)

Warn when a type with an ABI tag is used in a context that does not have that ABI tag. See Section 8.7 [C++ Attributes], page 1037, for more information about ABI tags.

-Wno-abbreviated-auto-in-template-arg

Disable the error for an `auto` placeholder type used within a template argument list to declare a C++20 abbreviated function template, e.g.

```
void f(S<auto>);
```

This feature was proposed in the Concepts TS, but was not adopted into C++20; in the standard, a placeholder in a parameter declaration must appear as a

decl-specifier. The error can also be reduced to a warning by `-fpermissive` or `-Wno-error=abbreviated-auto-in-template-arg`.

-Wcomma-subscript (C++ and Objective-C++ only)

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

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

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

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

This warning is upgraded to an error by `-pedantic-errors` in C++23 mode or later.

-Wctad-maybe-unsupported (C++ and Objective-C++ only)

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

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

-Wctor-dtor-privacy (C++ and Objective-C++ only)

Warn when a class seems unusable because all the constructors or destructors in that class are private, and it has neither friends nor public static member functions. Also warn if there are no non-private methods, and there's at least one private member function that isn't a constructor or destructor.

-Wdangling-reference (C++ and Objective-C++ only)

Warn when a reference is bound to a temporary whose lifetime has ended. For example:

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

In the example above, two temporaries are created, one for each argument, and a reference to one of the temporaries is returned. However, both temporaries

are destroyed at the end of the full expression, so the reference `r` is dangling. This warning also detects dangling references in member initializer lists:

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

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

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

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

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

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

`-Wdangling-reference` also warns about code like

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

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

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

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

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

The warning can be disabled by using the `gnu::no_dangling` attribute (see Section 8.7 [C++ Attributes], page 1037).

This warning is enabled by `-Wextra`.

`-Wdelete-non-virtual-dtor` (C++ and Objective-C++ only)

Warn when `delete` is used to destroy an instance of a class that has virtual functions and non-virtual destructor. It is unsafe to delete an instance of a derived class through a pointer to a base class if the base class does not have a virtual destructor. This warning is enabled by `-Wall`.

`-Wdeprecated-copy` (C++ and Objective-C++ only)

Warn that the implicit declaration of a copy constructor or copy assignment operator is deprecated if the class has a user-provided copy constructor or copy assignment operator, in C++11 and up. This warning is enabled by `-Wextra`.

-Wdeprecated-copy-dtor (C++ and Objective-C++ only)

Similar to **-Wdeprecated-copy**, but also deprecate if the class has a user-provided destructor.

-Wno-deprecated-enum-enum-conversion (C++ and Objective-C++ only)

Disable the warning about the case when the usual arithmetic conversions are applied on operands where one is of enumeration type and the other is of a different enumeration type. This conversion was deprecated in C++20. For example:

```
enum E1 { e };
enum E2 { f };
int k = f - e;
```

-Wdeprecated-enum-enum-conversion is enabled by default with **-std=c++20**. In pre-C++20 dialects, this warning can be enabled by **-Wenum-conversion** or **-Wdeprecated**.

-Wno-deprecated-enum-float-conversion (C++ and Objective-C++ only)

Disable the warning about the case when the usual arithmetic conversions are applied on operands where one is of enumeration type and the other is of a floating-point type. This conversion was deprecated in C++20. For example:

```
enum E1 { e };
enum E2 { f };
bool b = e <= 3.7;
```

-Wdeprecated-enum-float-conversion is enabled by default with **-std=c++20**. In pre-C++20 dialects, this warning can be enabled by **-Wenum-conversion** or **-Wdeprecated**.

-Wdeprecated-literal-operator (C++ and Objective-C++ only)

Warn that the declaration of a user-defined literal operator with a space before the suffix is deprecated. This warning is enabled by default in C++23, or with explicit **-Wdeprecated**.

```
string operator "" _i18n(const char*, std::size_t); // deprecated
string operator ""_i18n(const char*, std::size_t); // preferred
```

-Wdeprecated-variadic-comma-omission (C++ and Objective-C++ only)

Warn that omitting a comma before the varargs ... at the end of a function parameter list is deprecated. This warning is enabled by default in C++26, or with explicit **-Wdeprecated**.

```
void f1(int...); // deprecated
void f1(int, ...); // preferred
template <typename ...T>
void f2(T...); // ok
template <typename ...T>
void f3(T.....); // deprecated
```

-Wno-elaborated-enum-base

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

-Wno-init-list-lifetime (C++ and Objective-C++ only)

Do not warn about uses of `std::initializer_list` that are likely to result in dangling pointers. Since the underlying array for an `initializer_list` is handled like a normal C++ temporary object, it is easy to inadvertently keep a pointer to the array past the end of the array's lifetime. For example:

- If a function returns a temporary `initializer_list`, or a local `initializer_list` variable, the array's lifetime ends at the end of the return statement, so the value returned has a dangling pointer.
- If a new-expression creates an `initializer_list`, the array only lives until the end of the enclosing full-expression, so the `initializer_list` in the heap has a dangling pointer.
- When an `initializer_list` variable is assigned from a brace-enclosed initializer list, the temporary array created for the right side of the assignment only lives until the end of the full-expression, so at the next statement the `initializer_list` variable has a dangling pointer.

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

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

-Winvalid-constexpr

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

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

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

-Winvalid-imported-macros

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

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

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

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

int main() {
    int64_t i64 = 123;
    printf("My int64: %" PRIu64"\n", i64);
}
```

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

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

These warnings are enabled by default.

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

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

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

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

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

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

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

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

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

As an example:

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

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

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

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

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

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

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

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

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

Warn on uses of the `register` storage class specifier, except when it is part of the GNU Section 6.11.6 [Explicit Register Variables], page 774, extension. The use of the `register` keyword as storage class specifier has been deprecated in C++11 and removed in C++17. Enabled by default with `-std=c++17`.

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

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

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

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

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

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

```
struct T {
    ...
```

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

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

This warning is enabled by `-Wall`.

`-Wno-redundant-move` (C++ and Objective-C++ only)

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

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

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

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

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

This warning is enabled by `-Wextra`.

`-Wrange-loop-construct` (C++ and Objective-C++ only)

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

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

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

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

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

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

This warning is enabled by `-Wall`.

`-Wredundant-tags` (C++ and Objective-C++ only)

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

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

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

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

`-Wno-subobject-linkage` (C++ and Objective-C++ only)

Do not warn if a class type has a base or a field whose type uses the anonymous namespace or depends on a type with no linkage. If a type A depends on a type B with no or internal linkage, defining it in multiple translation units would be an ODR violation because the meaning of B is different in each translation unit. If A only appears in a single translation unit, the best way to silence the warning is to give it internal linkage by putting it in an anonymous namespace as well. The compiler doesn't give this warning for types defined in the main .C file, as those are unlikely to have multiple definitions. `-Wsubobject-linkage` is enabled by default.

`-Weffc++` (C++ and Objective-C++ only)

Warn about violations of the following style guidelines from Scott Meyers' *Effective C++* series of books:

- Define a copy constructor and an assignment operator for classes with dynamically-allocated memory.
- Prefer initialization to assignment in constructors.
- Have `operator=` return a reference to `*this`.
- Don't try to return a reference when you must return an object.
- Distinguish between prefix and postfix forms of increment and decrement operators.
- Never overload `&&`, `||`, or `,.`

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

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

-Wno-exceptions (C++ and Objective-C++ only)

Disable the warning about the case when an exception handler is shadowed by another handler, which can point out a wrong ordering of exception handlers.

-Wsfinae-incomplete (C++ and Objective-C++ only)

Warn about a class that is found to be incomplete, or a function with auto return type that has not yet been deduced, in a context where that causes substitution failure rather than an error, and then the class or function is defined later in the translation unit. This is problematic because template instantiations or concept checks could have different results if they first occur either before or after the definition.

This warning is enabled by default. `-Wsfinae-incomplete=2` adds a warning at the point of substitution failure, to make it easier to track down problems flagged by the default mode.

-Wstrict-null-sentinel (C++ and Objective-C++ only)

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

-Wno-non-c-typedef-for-linkage (C++ and Objective-C++ only)

Disable pedwarn for unnamed classes with a typedef name for linkage purposes containing C++ specific members, base classes, default member initializers or lambda expressions, including those on nested member classes.

```
typedef struct {
    int a; // non-static data members are ok
    struct T { int b; }; // member classes too
    enum E { E1, E2, E3 }; // member enumerations as well
    int c = 42; // default member initializers are not ok
    struct U : A { int c; }; // classes with base classes are not ok
    typedef int V; // typedef is not ok
    using W = int; // using declaration is not ok
    decltype([](){} ) x; // lambda expressions not ok
} S;
```

In all these cases, the tag name S should be added after the struct keyword.

-Wno-non-template-friend (C++ and Objective-C++ only)

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

-Wold-style-cast (C++ and Objective-C++ only)

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

-Woverloaded-virtual (C++ and Objective-C++ only)**-Woverloaded-virtual=*n***

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

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

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

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

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

fails to compile.

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

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

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

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

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

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

-Wno-pmf-conversions (C++ and Objective-C++ only)

Disable the diagnostic for converting a bound pointer to member function to a plain pointer.

-Wsign-promo (C++ and Objective-C++ only)

Warn when overload resolution chooses a promotion from unsigned or enumerated type to a signed type, over a conversion to an unsigned type of the same size. Previous versions of G++ tried to preserve unsignedness, but the standard mandates the current behavior.

-Wtemplates (C++ and Objective-C++ only)

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

-Wmismatched-new-delete (C++ and Objective-C++ only)

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

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

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

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

The related option `-Wmismatched-dealloc` diagnoses mismatches involving allocation and deallocation functions other than `operator new` and `operator delete`.

`-Wmismatched-new-delete` is included in `-Wall`.

-Wmismatched-tags (C++ and Objective-C++ only)

Warn for declarations of structs, classes, and class templates and their specializations with a class-key that does not match either the definition or the first declaration if no definition is provided.

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

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

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

-Wmultiple-inheritance (C++ and Objective-C++ only)

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

-Wvirtual-inheritance

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

-Wno-virtual-move-assign

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

-Wnamespaces

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

-Wno-template-body (C++ and Objective-C++ only)

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

-Wno-template-id-ctor (C++ and Objective-C++ only)

Disable the warning about the use of `simple-template-id` as the declarator-id of a constructor or destructor, which became invalid in C++20 via DR 2237. For example:

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

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

-Wtemplate-names-tu-local

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

This can cause variable or function templates to accidentally become unusable if they reference such an entity, because other translation units that import the

template will never be able to instantiate it. This warning attempts to detect cases where this might occur. The presence of an explicit instantiation silences the warning.

This flag is enabled by `-Wextra`.

`-Wno-expose-global-module-tu-local`

An exposure of a translation-unit-local entity from a module interface is invalid, as this may cause ODR violations and manifest in link errors or other unexpected behaviour. However, many existing libraries declare TU-local entities in their interface, and avoiding exposures of these entities may be difficult in some cases.

As an extension, GCC allows exposures of internal variables and functions that were declared in the global module fragment. This warning indicates when such an invalid exposure has occurred, and can be silenced using diagnostic pragmas either at the site of the exposure, or at the point of declaration of the internal declaration. This also applies to `export using` declarations naming such entities.

When combined with `-Wtemplate-names-tu-local`, GCC will also warn about non-exposure references to TU-local entities in template bodies. Such templates can still be instantiated in other TUs but the above risks regarding exposures of translation-unit-local entities apply.

This warning is enabled by default, and is upgraded to an error by `-pedantic-errors`.

`-Wno-external-tu-local`

Warn when naming a TU-local entity outside of the translation unit it was declared in. Such declarations will be ignored during name lookup. This can occur when performing ADL from a template declared in the same TU as the internal function:

```
export module M;
template <typename T> void foo(T t) {
    bar(t);
}
struct S {} s;
static void bar(S) {} // internal linkage

// instantiating foo(s) from outside this TU can see ::bar,
// but naming it there is ill-formed.
```

This can be worked around by making `bar` attached to the global module, using `extern "C++"`.

This warning is enabled by default, and is upgraded to an error by `-pedantic-errors`.

`-Wno-terminate` (C++ and Objective-C++ only)

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

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

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

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

Another example:

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

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

This warning is enabled by default.

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

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

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

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

Enabled by default with `-std=c++20`. Before `-std=c++20`, enabled with explicit `-Wdeprecated`.

-Waligned-new**-Waligned-new=[none|global|all]**

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

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

-Wno-placement-new**-Wplacement-new=n**

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

```
char buf [64];
```

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

This warning is enabled by default.

-Wplacement-new=1

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

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

-Wplacement-new=2

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

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

-Wcatch-value

-Wcatch-value=n (C++ and Objective-C++ only)

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

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

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

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

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

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

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

-Wextra-semi (C++, Objective-C++ only)

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

```
struct S {
```

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

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

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

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

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

-Wno-global-module (C++ and Objective-C++ only)

Disable the diagnostic for when the global module fragment of a module unit does not consist only of preprocessor directives.

-Wno-inaccessible-base (C++, Objective-C++ only)

This option controls warnings when a base class is inaccessible in a class derived from it due to ambiguity. The warning is enabled by default. Note that the warning for ambiguous virtual bases is enabled by the **-Wextra** option.

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

-Wno-inherited-variadic-ctor

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

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

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

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

-Wsizeof-deallocation (C++ and Objective-C++ only)

Warn about a definition of an unsized deallocation function

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

without a definition of the corresponding sized deallocation function

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

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

-Wsuggest-final-types

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

-Wsuggest-final-methods

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

-Wsuggest-override

Warn about overriding virtual functions that are not marked with the **override** keyword.

-Wno-conversion-null (C++ and Objective-C++ only)

Do not warn for conversions between **NULL** and non-pointer types. **-Wconversion-null** is enabled by default.

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 3, for references.)

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

```
gcc -g -fgnu-runtime -O -c some_class.m
```

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

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

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

-fconstant-string-class=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). On Darwin / macOS platforms, the **-fconstant-cfstrings** option, if also present, overrides the **-fconstant-string-class**

setting and cause `@"..."` literals to be laid out as constant CoreFoundation strings. Note that `-fconstant-cfstrings` is an alias for the target-specific `-mconstant-cfstrings` equivalent.

`-fgnu-runtime`

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

`-fnext-runtime`

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

`-fno-nil-receivers`

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

`-fobjc-abi-version=n`

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

`-fobjc-call-cxx-ctors`

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

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

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

-fobjc-direct-dispatch

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

-fobjc-exceptions

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

-fobjc-gc

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

-fobjc-nilcheck

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

-fobjc-std=objc1

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

-freplace-objc-classes

Emit a special marker instructing `ld(1)` not to statically link in the resulting object file, and allow `dyld(1)` to load it in at run time instead. This is used in conjunction with the Fix-and-Continue debugging mode, where the object file in question may be recompiled and dynamically reloaded in the course of program execution, without the need to restart the program itself. Currently, Fix-and-Continue functionality is only available in conjunction with the NeXT runtime on Mac OS X 10.3 and later.

-fzero-link

When compiling for the NeXT runtime, the compiler ordinarily replaces calls to `objc_getClass(...)` (when the name of the class is known at compile time) with static class references that get initialized at load time, which improves run-time performance. Specifying the **-fzero-link** flag suppresses this behavior and causes calls to `objc_getClass(...)` to be retained. This is useful in Zero-Link debugging mode, since it allows for individual class implementations

to be modified during program execution. The GNU runtime currently always retains calls to `objc_get_class(...)` regardless of command-line options.

-fno-local-ivars

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

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

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

-gen-decls

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

-print-objc-runtime-info

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

3.7 Options Controlling OpenMP and OpenACC

GCC supports OpenMP extensions to the C, C++, and Fortran languages with the `-fopenmp` option. Similarly, OpenACC extensions are supported in all three languages with `-fopenacc`. See Section 6.7 [OpenMP], page 717, and Section 6.8 [OpenACC], page 718, for an overview of these extensions.

-foffload=disable**-foffload=default****-foffload=target-list**

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

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

-foffload-options=options**-foffload-options=target-triplet-list=options**

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

Typical command lines are

```
-foffload-options='-fno-math-errno -ffinite-math-only' \
```

```
-foffload-options=nvptx-none=-latomic
-foffload-options=amdgc-n-amdhsa=-march=gfx906
```

-fopenacc

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

-fopenacc-dim=geom

Specify default compute dimensions for parallel offload regions that do not explicitly specify them. The *geom* value is a triple of ‘:’-separated sizes, in order *gang*, *worker*, and *vector*. A size can be omitted, to use a target-specific default value.

-fopenmp

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

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

-fopenmp-simd

Enable handling of OpenMP’s `simd`, `declare simd`, `declare reduction`, `assume`, `ordered`, `scan` and `loop` directive, and of combined or composite directives with `simd` as constituent with `#pragma omp`, `[[omp::directive(...)]]`, `[[omp::sequence(...)]]` and `[[omp::decl(...)]]` in C/C++ and `!$omp` in Fortran. It additionally enables the conditional compilation sentinel ‘`!$`’ in Fortran. In fixed source form Fortran, the sentinels can also start with ‘`c`’ or ‘`*`’. Other OpenMP directives are ignored. Unless `-fopenmp` is additionally specified, the `loop` region binds to the current task region, independent of the specified `bind` clause.

-fopenmp-target-simd-clone**-fopenmp-target-simd-clone=device-type**

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

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

3.8 Options to Control Diagnostic Messages Formatting

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

`-fmessage-length=n`

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

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

`-fdiagnostics-plain-output`

This option requests that diagnostic output look as plain as possible, which may be useful when running `dejagnu` or other utilities that need to parse diagnostics output and prefer that it remain more stable over time. `-fdiagnostics-plain-output` is currently equivalent to the following options:

```
-fno-diagnostics-show-caret
-fno-diagnostics-show-line-numbers
-fdiagnostics-color=never
-fdiagnostics-urls=never
-fdiagnostics-path-format=separate-events
-fdiagnostics-text-art-charset=none
-fno-diagnostics-show-event-links
-fno-diagnostics-show-nesting
```

In the future, if GCC changes the default appearance of its diagnostics, the corresponding option to disable the new behavior will be added to this list.

`-fdiagnostics-show-location=once`

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

`-fdiagnostics-show-location=every-line`

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

`-fdiagnostics-color[=WHEN]`

`-fno-diagnostics-color`

Use color in diagnostics. *WHEN* is `'never'`, `'always'`, or `'auto'`. The default depends on how the compiler has been configured, it can be any of the above *WHEN* options or also `'never'` if `GCC_COLORS` environment variable isn't present

in the environment, and ‘auto’ otherwise. ‘auto’ makes GCC use color only when the standard error is a terminal, and when not executing in an emacs shell. The forms `-fdiagnostics-color` and `-fno-diagnostics-color` are aliases for `-fdiagnostics-color=always` and `-fdiagnostics-color=never`, respectively.

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

The default `GCC_COLORS` is

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

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

<code>error=</code>	SGR substring for error: markers.
<code>warning=</code>	SGR substring for warning: markers.
<code>note=</code>	SGR substring for note: markers.
<code>path=</code>	SGR substring for colorizing paths of control-flow events as printed via <code>-fdiagnostics-path-format=</code> , such as the identifiers of individual events and lines indicating interprocedural calls and returns.
<code>range1=</code>	SGR substring for first additional range.
<code>range2=</code>	SGR substring for second additional range.
<code>locus=</code>	SGR substring for location information, ‘file:line’ or ‘file:line:column’ etc.
<code>quote=</code>	SGR substring for information printed within quotes.
<code>fname=</code>	SGR substring for names of C++ functions.
<code>targs=</code>	SGR substring for C++ function template parameter bindings.
<code>fixit-insert=</code>	SGR substring for fix-it hints suggesting text to be inserted or replaced.

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

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

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

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

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

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

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

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

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

-fdiagnostics-urls[=*WHEN*]

Use escape sequences to embed URLs in diagnostics. For example, when **-fdiagnostics-show-option** emits text showing the command-line option controlling a diagnostic, embed a URL for documentation of that option.

WHEN is **'never'**, **'always'**, or **'auto'**. **'auto'** makes GCC use URL escape sequences only when the standard error is a terminal, and when not executing in an emacs shell or any graphical terminal which is known to be incompatible with this feature, see below.

The default depends on how the compiler has been configured. It can be any of the above *WHEN* options.

GCC can also be configured (via the **--with-diagnostics-urls=auto-if-env** configure-time option) so that the default is affected by environment variables. Under such a configuration, GCC defaults to using **'auto'** if either **GCC_URLS** or **TERM_URLS** environment variables are present and non-empty in the environment of the compiler, or **'never'** if neither are.

However, even with **-fdiagnostics-urls=always** the behavior is dependent on those environment variables: If **GCC_URLS** is set to empty or **'no'**, do not embed URLs in diagnostics. If set to **'st'**, URLs use ST escape sequences. If set to **'bel'**, the default, URLs use BEL escape sequences. Any other non-empty value enables the feature. If **GCC_URLS** is not set, use **TERM_URLS** as a fallback.

Note: ST is an ANSI escape sequence, string terminator ‘ESC \’, BEL is an ASCII character, CTRL-G that usually sounds like a beep.

At this time GCC tries to detect also a few terminals that are known to not implement the URL feature, and have bugs or at least had bugs in some versions that are still in use, where the URL escapes are likely to misbehave, i.e. print garbage on the screen. That list is currently xfce4-terminal, certain known to be buggy gnome-terminal versions, the linux console, and mingw. This check can be skipped with the `-fdiagnostics-urls=always`.

`-fno-diagnostics-show-option`

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

`-fno-diagnostics-show-caret`

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

`-fno-diagnostics-show-labels`

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

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

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

`-fno-diagnostics-show-event-links`

By default, when printing execution paths (via `-fdiagnostics-path-format=inline-events`), GCC will print lines connecting related events, such as the line connecting events 1 and 2 in:

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

This option suppresses the printing of such connector lines.

-fno-diagnostics-show-cwe

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

-fno-diagnostics-show-rules

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

-fno-diagnostics-show-highlight-colors

GCC can use color for emphasis and contrast when printing diagnostic messages and quoting the user's source.

For example, in

```
demo.c: In function `test_bad_format_string_args':
../src/demo.c:25:18: warning: format '%i' expects argument of type `int', but argument 2 has type `const char*'
25 |   printf("hello %i", msg);
   |                   ^~   ~~~
   |                   |   |
   |                   int const char *
   |                   %s
```

- the **%i** and **int** in the message and the **int** in the quoted source are colored using **highlight-a** (bold green by default), and
- the **const char *** in the message and in the quoted source are both colored using **highlight-b** (bold blue by default).

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

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

-fno-diagnostics-show-line-numbers

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

-fdiagnostics-minimum-margin-width=width

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

-fdiagnostics-show-context[=depth]**-fno-diagnostics-show-context**

With this option, the compiler might print the interesting control flow chain that guards the basic block of the statement which has the warning.

depth is the maximum depth of the control flow chain. Currently, The list of the impacted warning options includes: `-Warray-bounds`, `-Wstringop-overflow`, `-Wstringop-overread`, `-Wstringop-truncation`, and `-Wrestrict`. More warning options might be added to this list in future releases. The forms `-fdiagnostics-show-context` and `-fno-diagnostics-show-context` are aliases for `-fdiagnostics-show-context=1` and `-fdiagnostics-show-context=0`, respectively.

`-fdiagnostics-parseable-fixits`

Emit fix-it hints in a machine-parseable format, suitable for consumption by IDEs. For each fix-it, a line will be printed after the relevant diagnostic, starting with the string “fix-it:”. For example:

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

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

```
00000000011111111122222222222
12345678901234567890123456789
  gtk_widget_showall (dlg);
  ~~~~~
  gtk_widget_show_all
```

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

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

`-fdiagnostics-generate-patch`

Print fix-it hints to stderr in unified diff format, after any diagnostics are printed. For example:

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

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

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

`-fdiagnostics-show-template-tree`

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

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

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

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

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

-fno-elide-type

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

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

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

-fdiagnostics-path-format=KIND

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

KIND is ‘none’, ‘separate-events’, or ‘inline-events’, the default.

‘none’ means to not print diagnostic paths.

‘separate-events’ means to print a separate “note” diagnostic for each event within the diagnostic. For example:

```
test.c:29:5: error: passing NULL as argument 1 to 'PyList_Append' which requires a non-NULL p
test.c:25:10: note: (1) when 'PyList_New' fails, returning NULL
test.c:27:3: note: (2) when 'i < count'
test.c:29:5: note: (3) when calling 'PyList_Append', passing NULL from (1) as argument 1■
```

‘inline-events’ means to print the events “inline” within the source code. This view attempts to consolidate the events into runs of sufficiently-close events, printing them as labelled ranges within the source.

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

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

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

For example:

```
'test': events 1-2
```

```

|
| 133 | {
|     | ^
|     | |
|     | (1) entering 'test'
| 134 | boxed_int *obj = make_boxed_int (i);
|     | ~~~~~
|     | |
|     | (2) calling 'make_boxed_int'
|
+--> 'make_boxed_int': events 3-4
|
| 120 | {
|     | ^
|     | |
|     | (3) entering 'make_boxed_int'
| 121 | boxed_int *result = (boxed_int *)wrapped_malloc (sizeof (boxed_int));
|     | ~~~~~
|     | |
|     | (4) calling 'wrapped_malloc'
|
+--> 'wrapped_malloc': events 5-6
|
| 7 | {
|   | ^
|   | |
|   | (5) entering 'wrapped_malloc'
| 8 | return malloc (size);
|   | ~~~~~
|   | |
|   | (6) calling 'malloc'
|
<-----+
|
'test': event 7
|
| 138 | free_boxed_int (obj);
|     | ~~~~~
|     | |
|     | (7) calling 'free_boxed_int'
|
(etc)

```

-fdiagnostics-show-path-depths

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

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

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

-fno-show-column

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

-fdiagnostics-column-unit=UNIT

Select the units for the column number. This affects traditional diagnostics (in the absence of `-fno-show-column`).

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

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

-fdiagnostics-column-origin=ORIGIN

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

-fdiagnostics-escape-format=FORMAT

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

This option controls how such bytes should be escaped.

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

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

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

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

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

-fdiagnostics-text-art-charset=CHARSET

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

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

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

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

-fno-diagnostics-show-nesting

Some GCC diagnostics have an internal tree-like structure of nested sub-diagnostics, such as for problems when instantiating C++ templates.

By default GCC uses indentation and bullet points in its text output to show the nesting structure of these diagnostics, moves location information to separate lines to make the structure clearer, and eliminates redundant repeated information.

Selecting **-fno-diagnostics-show-nesting** suppresses this indentation, reformatting, and elision, restoring an older ‘look’ for the diagnostics.

-fno-diagnostics-show-nesting-locations

When **fdiagnostics-show-nesting** is enabled, file names and line- and column- numbers are displayed on separate lines from the messages. This location information can be disabled altogether with **-fno-diagnostics-show-nesting-locations**. This option exists for use by GCC developers, for writing DejaGnu test cases.

-fdiagnostics-show-nesting-levels

When **fdiagnostics-show-nesting** is enabled, use **fdiagnostics-show-nesting-levels** to also display numbers showing the depth of the nesting. This option exists for use by GCC developers for debugging nested diagnostics, but may be of use to plugin authors.

-fdiagnostics-format=FORMAT

Select a different format for printing diagnostics. *FORMAT* is ‘text’, ‘sarif-stderr’ or ‘sarif-file’.

Using this option replaces any additional “output sinks” added by **-fdiagnostics-add-output=**, or that set by **-fdiagnostics-set-output=**.

The default is ‘text’.

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

-fdiagnostics-add-output=DIAGNOSTICS-OUTPUT-SPEC

Add an additional “output sink” for emitting diagnostics.

DIAGNOSTICS-OUTPUT-SPEC should specify a scheme, optionally followed by : and one or more *KEY=VALUE* pairs, in this form:

SCHEME

```
SCHEME:KEY=VALUE
SCHEME:KEY=VALUE,KEY2=VALUE2
```

etc.

Schemes, keys, or values with a name prefixed “experimental” may change or be removed without notice. Keys can be per-scheme, or related to GCC as a whole.

SCHEME can be

text Emit diagnostics to stderr using GCC’s classic text output format. Supported keys for the **text** scheme are:

color=[yes|no]

Override colorization settings from **-fdiagnostics-color** for this text output.

show-nesting=[yes|no]

Enable a mode that emphasizes hierarchical relationships within diagnostics messages, as per **-fdiagnostics-show-nesting**. Defaults to **yes**.

show-nesting-locations=[yes|no]

If **show-nesting=yes**, then by default locations are shown; set this key to **no** to disable printing such locations. This exists for use by GCC developers, for writing DejaGnu test cases.

show-nesting-levels=[yes|no]

This is a debugging option for use with **show-nesting=yes**. Set this key to **yes** to print explicit nesting levels in the output. This exists for use by GCC developers.

sarif Emit diagnostics to a file in SARIF format.

Supported keys for the **sarif** scheme are:

file=*FILENAME*

Specify the filename to write the SARIF output to, potentially with a leading absolute or relative path. If not specified, it defaults to **source.sarif**.

serialization=[json]

Specify the serialization format to use when writing out the SARIF. Currently this can only be **json**, but is present as an extension point for experimenting with other serializations.

version=[2.1|2.2-prerelease]

Specify the version of SARIF to use for the output. If not specified, defaults to 2.1. **2.2-prerelease** uses an unofficial draft of the future SARIF 2.2 specification and should only be used for experimentation in this release.

There is also this key intended for use by GCC developers, rather than end-users, and subject to change or removal without notice:

state-graphs=[yes|no]

This is a debugging feature and defaults to **no**. If **state-graphs=yes**, then attempt to capture detailed state information from **-fanalyzer** in the generated SARIF.

experimental-html

Emit diagnostics to a file in HTML format. This scheme is experimental, and may go away in future GCC releases. The keys and details of the output are also subject to change.

Supported keys for the **experimental-html** scheme are:

css=[yes|no]

Add an embedded `<style>` to the generated HTML. Defaults to **yes**.

file=FILENAME

Specify the filename to write the HTML output to, potentially with a leading absolute or relative path. If not specified, it defaults to **source.html**.

javascript=[yes|no]

Add an embedded `<script>` to the generated HTML providing a barebones UI for viewing results. Defaults to **yes**.

There are also these keys intended for use by GCC developers, rather than end-users, and subject to change or removal without notice:

show-state-diagrams=[yes|no]

This is a debugging feature and defaults to **no**. If **show-state-diagrams=yes**, then attempt to use **dot** to generate SVG diagrams in the generated HTML, visualizing the state at each event in a diagnostic path. These are visible by pressing “j” and “k” to single-step forward and backward through events. Enabling this option will slow down HTML generation.

show-graph-dot-src=[yes|no]

This is a debugging feature and defaults to **no**. If **show-graph-dot-src=yes** then if **show-state-diagrams=yes**, the generated state diagrams will also show the **.dot** source input to GraphViz used for the diagram.

show-graph-sarif=[yes|no]

This is a debugging feature and defaults to **no**. If **show-graph-sarif=yes** then if **show-state-diagrams=yes**,

the generated state diagrams will also show a SARIF representation of the state.

As well as scheme-specific keys, the following GCC-related key is usable on sinks of any scheme:

`cfigs=[yes|no]`

If `cfigs=yes` for a sink, then GCC will attempt to send information to that sink about the control flow graphs for the functions it is compiling. Text sinks ignore the information. SARIF sinks will add the graphs within `theRun.graphs`. HTML sinks will generate SVG displaying the graphs. The precise form of the information is subject to change without notice.

For example,

```
-fdiagnostics-add-output=sarif:version=2.1,file=foo.2.1.sarif
-fdiagnostics-add-output=sarif:version=2.2-prerelease,file=foo.2.2.sarif
```

would add a pair of outputs, each writing to a different file, using versions 2.1 and 2.2 of the SARIF standard respectively.

In EBNF:

```
diagnostics-output-specifier = diagnostics-output-name
                               | diagnostics-output-name, ":", key-value-pairs;

diagnostics-output-name = "text" | "sarif" | "experimental-html";

key-value-pairs = key-value-pair
                 | key-value-pair "," key-value-pairs;

key-value-pair = key "=" value;

key = ? string without a '=' ? ;
value = ? string without a ',' ? ;
```

`-fdiagnostics-set-output=DIAGNOSTICS-OUTPUT-SPEC`

This works in a similar way to `-fdiagnostics-add-output=` except that instead of adding an additional “output sink” for diagnostics, it replaces all existing output sinks, such as from `-fdiagnostics-format=`, `-fdiagnostics-add-output=`, or a prior call to `-fdiagnostics-set-output=`.

`-fno-diagnostics-json-formatting`

By default, when JSON is emitted for diagnostics (via `-fdiagnostics-format=sarif-stderr` or `-fdiagnostics-format=sarif-file`), GCC will add newlines and indentation to visually emphasize the hierarchical structure of the JSON.

Use `-fno-diagnostics-json-formatting` to suppress this whitespace. It must be passed before the option it is to affect.

This is intended for compatibility with tools that do not expect the output to contain newlines, such as that emitted by older GCC releases.

3.9 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

--no-warnings

Inhibit all warning messages.

-Werror Turn all warnings into errors.

-Werror= Turn the specified warning into an error. The specifier for a warning is appended; for example **-Werror=switch** turns the warnings controlled by **-Wswitch** into errors. This switch takes a negative form, to be used to negate **-Werror** for specific warnings; for example **-Wno-error=switch** makes **-Wswitch** warnings not be errors, even when **-Werror** is in effect.

The warning message for each controllable warning includes the option that controls the warning. That option can then be used with **-Werror=** and **-Wno-error=** as described above. (Printing of the option in the warning message can be disabled using the **-fno-diagnostics-show-option** flag.)

Note that specifying **-Werror=foo** automatically implies **-Wfoo**. However, **-Wno-error=foo** does not imply anything.

-Wfatal-errors

This option causes the compiler to abort compilation on the first error occurred rather than trying to keep going and printing further error messages.

You can request many specific warnings with options beginning with '**-W**', for example **-Wunused-variable** to request warnings on declarations of variables that are never used. Each of these specific warning options also has a negative form beginning with '**-Wno-**' to turn off warnings; for example, **-Wno-unused-variable**. This manual lists only one of the two forms, whichever is not the default. For further language-specific options also refer to Section 3.5 [C++ Dialect Options], page 52, and Section 3.6 [Objective-C and Objective-C++ Dialect Options], page 82. Additional warnings can be produced by enabling the static analyzer; See Section 3.10 [Static Analyzer Options], page 170.

Some options, such as **-Wall** and **-Wextra**, turn on other options, such as **-Wunused**, which may turn on further options, such as **-Wunused-variable**. The combined effect of positive and negative forms is that more specific options have priority over less specific ones,

independently of their position in the command line. For options of the same specificity, the last one takes effect. Options enabled or disabled via pragmas (see Section 6.5.12 [Diagnostic Pragmas], page 710) take effect as if they appeared at the end of the command line.

When an unrecognized warning option is requested (e.g., `-Wunknown-warning`), GCC gives an error stating that the option is not recognized. However, if the `-Wno-` form is used, the behavior is slightly different: no diagnostic is produced for `-Wno-unknown-warning` unless other diagnostics are being produced. This allows the use of new `-Wno-` options with old compilers, but if something goes wrong, the compiler warns that an unrecognized option is present.

The effectiveness of some warnings depends on optimizations also being enabled. For example, `-Wsuggest-final-types` is more effective with link-time optimization. Some other warnings may not be issued at all unless optimization is enabled. While optimization in general improves the efficacy of warnings about control and data-flow problems, in some cases it may also cause false positives.

`-Wpedantic`
`-pedantic`
`--pedantic`

Issue all the warnings demanded by strict ISO C and ISO C++; diagnose all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++. This follows the version of the ISO C or C++ standard specified by any `-std` option used.

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

`-Wpedantic` does not cause warning messages for use of the alternate keywords whose names begin and end with `'__'`. This alternate format can also be used to disable warnings for non-ISO `'__intN'` types, i.e. `'__intN__'`. Pedantic warnings are also disabled in the expression that follows `__extension__`. However, only system header files should use these escape routes; application programs should avoid them. See Section 6.12.23 [Alternate Keywords], page 792.

Some warnings about non-conforming programs are controlled by options other than `-Wpedantic`; in many cases they are implied by `-Wpedantic` but can be disabled separately by their specific option, e.g. `-Wpedantic -Wno-pointer-sign`.

Where the standard specified with `-std` represents a GNU extended dialect of C, such as `'gnu90'` or `'gnu99'`, there is a corresponding *base standard*, the version of ISO C on which the GNU extended dialect is based. Warnings from `-Wpedantic` are given where they are required by the base standard. (It does not make sense for such warnings to be given only for features not in the specified GNU C dialect, since by definition the GNU dialects of C include all features the compiler supports with the given option, and there would be nothing to warn about.)

-pedantic-errors**--pedantic-errors**

Give an error whenever the *base standard* (see **-Wpedantic**) requires a diagnostic, in some cases where there is undefined behavior at compile-time and in some other cases that do not prevent compilation of programs that are valid according to the standard. This is not equivalent to **-Werror=pedantic**: the latter option is unlikely to be useful, as it only makes errors of the diagnostics that are controlled by **-Wpedantic**, whereas this option also affects required diagnostics that are always enabled or controlled by options other than **-Wpedantic**.

If you want the required diagnostics that are warnings by default to be errors instead, but don't also want to enable the **-Wpedantic** diagnostics, you can specify **-pedantic-errors -Wno-pedantic** (or **-pedantic-errors -Wno-error=pedantic** to enable them but only as warnings).

Some required diagnostics are errors by default, but can be reduced to warnings using **-fpermissive** or their specific warning option, e.g. **-Wno-error=narrowing**.

Some diagnostics for non-ISO practices are controlled by specific warning options other than **-Wpedantic**, but are also made errors by **-pedantic-errors**. For instance:

```
-Wattributes (for standard attributes)
-Wchanges-meaning (C++)
-Wcomma-subscript (C++23 or later)
-Wdeclaration-after-statement (C90 or earlier)
-Welaborated-enum-base (C++11 or later)
-Wimplicit-int (C99 or later)
-Wimplicit-function-declaration (C99 or later)
-Wincompatible-pointer-types
-Wint-conversion
-Wlong-long (C90 or earlier)
-Wmain
-Wnarrowing (C++11 or later)
-Wpointer-arith
-Wpointer-sign
-Wincompatible-pointer-types
-Wregister (C++17 or later)
-Wvla (C90 or earlier)
-Wwrite-strings (C++11 or later)
```

-fpermissive

Downgrade some required diagnostics about nonconformant code from errors to warnings. Thus, using **-fpermissive** allows some nonconforming code to compile. Some C++ diagnostics are controlled only by this flag, but it also downgrades some C and C++ diagnostics that have their own flag:

```
-Wabbreviated-auto-in-template-arg (C++ and Objective-C++ only)
-Wdeclaration-missing-parameter-type (C and Objective-C only)
-Wimplicit-function-declaration (C and Objective-C only)
-Wimplicit-int (C and Objective-C only)
-Wincompatible-pointer-types (C and Objective-C only)
-Wint-conversion (C and Objective-C only)
```

```
-Wnarrowing (C++ and Objective-C++ only)
-Wreturn-mismatch (C and Objective-C only)
-Wtemplate-body (C++ and Objective-C++ only)
```

The `-fpermissive` option is the default for historic C language modes (`-std=c89`, `-std=gnu89`, `-std=c90`, `-std=gnu90`).

`-Wall`

`--all-warnings`

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 52, and Section 3.6 [Objective-C and Objective-C++ Dialect Options], page 82.

`-Wall` turns on the following warning flags:

```
-Waddress
-Waligned-new (C++ and Objective-C++ only)
-Warray-bounds=1 (only with -O2)
-Warray-compare
-Warray-parameter=2
-Wbool-compare
-Wbool-operation
-Wc++11-compat -Wc++14-compat -Wc++17compat -Wc++20compat
-Wcatch-value (C++ and Objective-C++ only)
-Wchar-subscripts
-Wclass-memaccess (C++ and Objective-C++ only)
-Wcomment
-Wdangling-else
-Wdangling-pointer=2
-Wdelete-non-virtual-dtor (C++ and Objective-C++ only)
-Wduplicate-decl-specifier (C and Objective-C only)
-Wenum-compare (in C/ObjC; this is on by default in C++)
-Wenum-int-mismatch (C and Objective-C only)
-Wformat=1
-Wformat-contains-nul
-Wformat-diag
-Wformat-extra-args
-Wformat-overflow=1
-Wformat-truncation=1
-Wformat-zero-length
-Wframe-address
-Wimplicit (C and Objective-C only)
-Wimplicit-function-declaration (C and Objective-C only)
-Wimplicit-int (C and Objective-C only)
-Winfinite-recursion
-Winit-self (C++ and Objective-C++ only)
-Wint-in-bool-context
-Wlogical-not-parentheses
-Wmain (only for C/ObjC and unless -ffreestanding)
-Wmaybe-uninitialized
-Wmemset-elt-size
-Wmemset-transposed-args
-Wmisleading-indentation (only for C/C++)
-Wmismatched-dealloc
-Wmismatched-new-delete (C++ and Objective-C++ only)
-Wmissing-attributes
```

```

-Wmissing-braces (only for C/ObjC)
-Wmultistatement-macros
-Wnarrowing (C++ and Objective-C++ only)
-Wnonnull
-Wnonnull-compare
-Wopenmp-simd (C and C++ only)
-Woverloaded-virtual=1 (C++ and Objective-C++ only)
-Wpacked-not-aligned
-Wparentheses
-Wpessimizing-move (C++ and Objective-C++ only)
-Wpointer-sign (only for C/ObjC)
-Wrange-loop-construct (C++ and Objective-C++ only)
-Wreorder (C++ and Objective-C++ only)
-Wrestrict
-Wreturn-type
-Wself-move (C++ and Objective-C++ only)
-Wsequence-point
-Wsign-compare (C++ and Objective-C++ only)
-Wsizeof-array-div
-Wsizeof-pointer-div
-Wsizeof-pointer-memaccess
-Wstrict-aliasing
-Wswitch
-Wtautological-compare
-Wtrigraphs
-Wuninitialized
-Wunknown-pragmas
-Wunused
-Wunused-but-set-variable
-Wunused-const-variable=1 (only for C/ObjC)
-Wunused-function
-Wunused-label
-Wunused-local-typedefs
-Wunused-value
-Wunused-variable
-Wuse-after-free=2
-Wvla-parameter
-Wvolatile-register-var
-Wzero-length-bounds

```

Note that some warning flags are not implied by `-Wall`. Some of them warn about constructions that users generally do not consider questionable, but which occasionally you might wish to check for; others warn about constructions that are necessary or hard to avoid in some cases, and there is no simple way to modify the code to suppress the warning. Some of them are enabled by `-Wextra` but many of them must be enabled individually.

`-Wextra`

`--extra-warnings`

This enables some extra warning flags that are not enabled by `-Wall`. (This option used to be called `-W`. The older name is still supported, but the newer name is more descriptive.)

```

-Wabsolute-value (only for C/ObjC)
-Walloc-size
-Wcalloc-transposed-args
-Wcast-function-type
-Wclobbered

```

```

-Wdangling-reference (C++ only)
-Wdeprecated-copy (C++ and Objective-C++ only)
-Wempty-body
-Wenum-conversion (only for C/ObjC)
-Wexpansion-to-defined
-Wignored-qualifiers (only for C/C++)
-Wimplicit-fallthrough=3
-Wmaybe-uninitialized
-Wmissing-field-initializers
-Wmissing-parameter-name (C/ObjC only)
-Wmissing-parameter-type (C/ObjC only)
-Wold-style-declaration (C/ObjC only)
-Wmultiple-parameter-fwd-decl-lists (C/ObjC only)
-Woverride-init (C/ObjC only)
-Wredundant-move (C++ and Objective-C++ only)
-Wshift-negative-value (in C++11 to C++17 and in C99 and newer)
-Wsign-compare (C++ and Objective-C++ only)
-Wsized-deallocation (C++ and Objective-C++ only)
-Wstring-compare
-Wtype-limits
-Wuninitialized
-Wunterminated-string-initialization (C/ObjC only)
-Wunused-parameter (only with -Wunused or -Wall)
-Wunused-but-set-parameter (only with -Wunused or -Wall)

```

The option `-Wextra` also prints warning messages for the following cases:

- A pointer is compared against integer zero with `<`, `<=`, `>`, or `>=`.
- (C++ only) An enumerator and a non-enumerator both appear in a conditional expression.
- (C++ only) Ambiguous virtual bases.
- (C++ only) Subscripting an array that has been declared **register**.
- (C++ only) Taking the address of a variable that has been declared **register**.
- (C++ only) A base class is not initialized in the copy constructor of a derived class.

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

Warn about code affected by ABI changes. This includes code that may not be compatible with the vendor-neutral C++ ABI as well as the psABI for the particular target. The latter warnings are also controlled separately by `-Wpsabi`, which is implied by `-Wabi`.

Since G++ now defaults to updating the ABI with each major release, normally `-Wabi` warns only about C++ ABI compatibility problems if there is a check added later in a release series for an ABI issue discovered since the initial release. `-Wabi` warns about more things if an older ABI version is selected (with `-fabi-version=n`).

`-Wabi` can also be used with an explicit version number to warn about C++ ABI compatibility with a particular `-fabi-version` level, e.g. `-Wabi=2` to warn about changes relative to `-fabi-version=2`.

If an explicit version number is provided and `-fabi-compat-version` is not specified, the version number from this option is used for compatibility aliases.

If no explicit version number is provided with this option, but `-fabi-compat-version` is specified, that version number is used for C++ ABI warnings.

Although an effort has been made to warn about all such cases, there are probably some cases that are not warned about, even though G++ is generating incompatible code. There may also be cases where warnings are emitted even though the code that is generated is compatible.

You should rewrite your code to avoid these warnings if you are concerned about the fact that code generated by G++ may not be binary compatible with code generated by other compilers.

Known incompatibilities in `-fabi-version=2` (which was the default from GCC 3.4 to 4.9) include:

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

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

This was fixed in `-fabi-version=3`.

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

The mangling was changed in `-fabi-version=4`.

- `__attribute__((const))` and `noreturn` were mangled as type qualifiers, and `decltype` of a plain declaration was folded away.

These mangling issues were fixed in `-fabi-version=5`.

- Scoped enumerators passed as arguments to a variadic function are promoted like unscoped enumerators, causing `va_arg` to complain. On most targets this does not actually affect the parameter passing ABI, as there is no way to pass an argument smaller than `int`.

Also, the ABI changed the mangling of template argument packs, `const_cast`, `static_cast`, prefix increment/decrement, and a class scope function used as a template argument.

These issues were corrected in `-fabi-version=6`.

- Lambdas in default argument scope were mangled incorrectly, and the ABI changed the mangling of `nullptr_t`.

These issues were corrected in `-fabi-version=7`.

- When mangling a function type with function-cv-qualifiers, the un-qualified function type was incorrectly treated as a substitution candidate.

This was fixed in `-fabi-version=8`, the default for GCC 5.1.

- `decltype(nullptr)` incorrectly had an alignment of 1, leading to unaligned accesses. Note that this did not affect the ABI of a function with a `nullptr_t` parameter, as parameters have a minimum alignment.

This was fixed in `-fabi-version=9`, the default for GCC 5.2.

- Target-specific attributes that affect the identity of a type, such as `ia32` calling conventions on a function type (`stdcall`, `regparm`, etc.), did not

affect the mangled name, leading to name collisions when function pointers were used as template arguments.

This was fixed in `-fabi-version=10`, the default for GCC 6.1.

`-Wpsabi` (C, Objective-C, C++ and Objective-C++ only)

`-Wpsabi` enables warnings about processor-specific ABI changes, such as changes in alignment requirements or how function arguments are passed. On several targets, including AArch64, ARM, x86, MIPS, RS6000/PowerPC, and S/390, these details have changed between different versions of GCC and/or different versions of the C or C++ language standards in ways that affect binary compatibility of compiled code. With `-Wpsabi`, GCC can detect potentially incompatible usages and warn you about them.

`-Wpsabi` is enabled by default, and is also implied by `-Wabi`.

`-Wno-changes-meaning` (C++ and Objective-C++ only)

C++ requires that unqualified uses of a name within a class have the same meaning in the complete scope of the class, so declaring the name after using it is ill-formed:

```
struct A;
struct B1 { A a; typedef A A; }; // warning, 'A' changes meaning
struct B2 { A a; struct A { }; }; // error, 'A' changes meaning
```

By default, the B1 case is only a warning because the two declarations have the same type, while the B2 case is an error. Both diagnostics can be disabled with `-Wno-changes-meaning`. Alternately, the error case can be reduced to a warning with `-Wno-error=changes-meaning` or `-fpermissive`.

Both diagnostics are also suppressed by `-fms-extensions`.

`-Wchar-subscripts`

Warn if an array subscript has type `char`. This is a common cause of error, as programmers often forget that this type is signed on some machines. See Section 4.4 [Characters implementation], page 564, and [char type signedness], page 51.

This warning is enabled by `-Wall`. When enabled, the warning is given regardless of whether `char` is unsigned by default on the target, and it is also not affected by the `-fsigned-char` or `-funsigned-char` options.

`-Wno-coverage-mismatch`

Warn if feedback profiles do not match when using the `-fprofile-use` option. If a source file is changed between compiling with `-fprofile-generate` and with `-fprofile-use`, the files with the profile feedback can fail to match the source file and GCC cannot use the profile feedback information. By default, this warning is enabled and is treated as an error. `-Wno-coverage-mismatch` can be used to disable the warning or `-Wno-error=coverage-mismatch` can be used to disable the error. Disabling the error for this warning can result in poorly optimized code and is useful only in the case of very minor changes such as bug fixes to an existing code-base. Completely disabling the warning is not recommended.

-Wno-coverage-too-many-conditions

Warn if `-fcondition-coverage` is used and an expression have too many terms and GCC gives up coverage. Coverage is given up when there are more terms in the conditional than there are bits in a `gcov_type_unsigned`. This warning is enabled by default.

-Wno-coverage-too-many-paths

Warn if `-fpath-coverage` is used and a function has too many paths and GCC gives up coverage. Giving up is controlled by `-fpath-coverage-limit`. This warning is enabled by default.

-Wno-coverage-invalid-line-number

Warn in case a function ends earlier than it begins due to an invalid `linenum` macros. The warning is emitted only with `--coverage` enabled.

By default, this warning is enabled and is treated as an error. `-Wno-coverage-invalid-line-number` can be used to disable the warning or `-Wno-error=coverage-invalid-line-number` can be used to disable the error.

-Wno-cpp (C, Objective-C, C++, Objective-C++ and Fortran only)

Suppress warning messages emitted by `#warning` directives.

-Wdouble-promotion (C, C++, Objective-C and Objective-C++ only)

Give a warning when a value of type `float` is implicitly promoted to `double`. CPUs with a 32-bit “single-precision” floating-point unit implement `float` in hardware, but emulate `double` in software. On such a machine, doing computations using `double` values is much more expensive because of the overhead required for software emulation.

It is easy to accidentally do computations with `double` because floating-point literals are implicitly of type `double`. For example, in:

```
float area(float radius)
{
    return 3.14159 * radius * radius;
}
```

the compiler performs the entire computation with `double` because the floating-point literal is a `double`.

-Wduplicate-decl-specifier (C and Objective-C only)

Warn if a declaration has a duplicate `const`, `volatile`, `restrict` or `_Atomic` specifier. This warning is enabled by `-Wall`.

-Wformat**-Wformat=n**

Check calls to `printf` and `scanf`, etc., to make sure that the arguments supplied have types appropriate to the format string specified, and that the conversions specified in the format string make sense. This includes standard functions, and others specified by format attributes (see Section 6.4.1 [Common Attributes], page 595), in the `printf`, `scanf`, `strftime` and `strfmon` (an X/Open extension, not in the C standard) families (or other target-specific families). Which functions are checked without format attributes having been specified depends

on the standard version selected, and such checks of functions without the attribute specified are disabled by `-ffreestanding` or `-fno-builtin`.

The formats are checked against the format features supported by GNU libc version 2.2. These include all ISO C90 and C99 features, as well as features from the Single Unix Specification and some BSD and GNU extensions. Other library implementations may not support all these features; GCC does not support warning about features that go beyond a particular library's limitations. However, if `-Wpedantic` is used with `-Wformat`, warnings are given about format features not in the selected standard version (but not for `strfmon` formats, since those are not in any version of the C standard). See Section 3.4 [Options Controlling C Dialect], page 45.

`-Wformat=1`

`-Wformat` Option `-Wformat` is equivalent to `-Wformat=1`, and `-Wno-format` is equivalent to `-Wformat=0`. Since `-Wformat` also checks for null format arguments for several functions, `-Wformat` also implies `-Wnonnull`. Some aspects of this level of format checking can be disabled by the options: `-Wno-format-contains-nul`, `-Wno-format-diag`, `-Wno-format-extra-args`, and `-Wno-format-zero-length`. `-Wformat` is enabled by `-Wall`.

`-Wformat=2`

Enable `-Wformat` plus additional format checks. Currently equivalent to `-Wformat -Wformat-nonliteral -Wformat-security -Wformat-y2k`.

`-Wno-format-contains-nul`

If `-Wformat` is specified, do not warn about format strings that contain NUL bytes.

`-Wno-format-diag`

If `-Wformat` is specified, do not warn about format strings that are unsuitable for GCC diagnostics.

`-Wno-format-extra-args`

If `-Wformat` is specified, do not warn about excess arguments to a `printf` or `scanf` format function. The C standard specifies that such arguments are ignored.

Where the unused arguments lie between used arguments that are specified with '\$' operand number specifications, normally warnings are still given, since the implementation could not know what type to pass to `va_arg` to skip the unused arguments. However, in the case of `scanf` formats, this option suppresses the warning if the unused arguments are all pointers, since the Single Unix Specification says that such unused arguments are allowed.

`-Wformat-overflow`

`-Wformat-overflow=level`

Warn about calls to formatted input/output functions such as `sprintf` and `vsprintf` that might overflow the destination buffer. When the exact number of bytes written by a format directive cannot be determined at compile-time

it is estimated based on heuristics that depend on the *level* argument and on optimization. While enabling optimization will in most cases improve the accuracy of the warning, it may also result in false positives.

-Wformat-overflow

-Wformat-overflow=1

Level 1 of **-Wformat-overflow** enabled by **-Wformat** employs a conservative approach that warns only about calls that most likely overflow the buffer. At this level, numeric arguments to format directives with unknown values are assumed to have the value of one, and strings of unknown length to be empty. Numeric arguments that are known to be bounded to a subrange of their type, or string arguments whose output is bounded either by their directive's precision or by a finite set of string literals, are assumed to take on the value within the range that results in the most bytes on output. For example, the call to `sprintf` below is diagnosed because even with both *a* and *b* equal to zero, the terminating NUL character (`'\0'`) appended by the function to the destination buffer will be written past its end. Increasing the size of the buffer by a single byte is sufficient to avoid the warning, though it may not be sufficient to avoid the overflow.

```
void f (int a, int b)
{
    char buf [13];
    sprintf (buf, "a = %i, b = %i\n", a, b);
}
```

-Wformat-overflow=2

Level 2 warns also about calls that might overflow the destination buffer given an argument of sufficient length or magnitude. At level 2, unknown numeric arguments are assumed to have the minimum representable value for signed types with a precision greater than 1, and the maximum representable value otherwise. Unknown string arguments whose length cannot be assumed to be bounded either by the directive's precision, or by a finite set of string literals they may evaluate to, or the character array they may point to, are assumed to be 1 character long.

At level 2, the call in the example above is again diagnosed, but this time because with *a* equal to a 32-bit `INT_MIN` the first `%i` directive will write some of its digits beyond the end of the destination buffer. To make the call safe regardless of the values of the two variables, the size of the destination buffer must be increased to at least 34 bytes. GCC includes the minimum size of the buffer in an informational note following the warning.

An alternative to increasing the size of the destination buffer is to constrain the range of formatted values. The maximum length of string arguments can be bounded by specifying the precision in the format directive. When numeric arguments of format directives can

be assumed to be bounded by less than the precision of their type, choosing an appropriate length modifier to the format specifier will reduce the required buffer size. For example, if *a* and *b* in the example above can be assumed to be within the precision of the `short int` type then using either the `%hi` format directive or casting the argument to `short` reduces the maximum required size of the buffer to 24 bytes.

```
void f (int a, int b)
{
    char buf [23];
    sprintf (buf, "a = %hi, b = %i\n", a, (short)b);
}
```

`-Wno-format-zero-length`

If `-Wformat` is specified, do not warn about zero-length formats. The C standard specifies that zero-length formats are allowed.

`-Wformat-nonliteral`

If `-Wformat` is specified, also warn if the format string is not a string literal and so cannot be checked, unless the format function takes its format arguments as a `va_list`.

`-Wformat-security`

If `-Wformat` is specified, also warn about uses of format functions that represent possible security problems. At present, this warns about calls to `printf` and `scanf` functions where the format string is not a string literal and there are no format arguments, as in `printf (foo);`. This may be a security hole if the format string came from untrusted input and contains `'%n'`. (This is currently a subset of what `-Wformat-nonliteral` warns about, but in future warnings may be added to `-Wformat-security` that are not included in `-Wformat-nonliteral`.)

`-Wformat-signedness`

If `-Wformat` is specified, also warn if the format string requires an unsigned argument and the argument is signed and vice versa.

`-Wformat-truncation`

`-Wformat-truncation=level`

Warn about calls to formatted input/output functions such as `snprintf` and `vsnprintf` that might result in output truncation. When the exact number of bytes written by a format directive cannot be determined at compile-time it is estimated based on heuristics that depend on the *level* argument and on optimization. While enabling optimization will in most cases improve the accuracy of the warning, it may also result in false positives. Except as noted otherwise, the option uses the same logic `-Wformat-overflow`.

`-Wformat-truncation`

`-Wformat-truncation=1`

Level 1 of `-Wformat-truncation` enabled by `-Wformat` employs a conservative approach that warns only about calls to bounded functions whose return value is unused and that will most likely result in output truncation.

-Wformat-truncation=2

Level 2 warns also about calls to bounded functions whose return value is used and that might result in truncation given an argument of sufficient length or magnitude.

-Wformat-y2k

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

-Wnonnull

Warn about passing a null pointer for arguments marked as requiring a non-null value by the **nonnull** function attribute.

-Wnonnull is included in **-Wall** and **-Wformat**. It can be disabled with the **-Wno-nonnull** option.

-Wnonnull-compare

Warn when comparing an argument marked with the **nonnull** function attribute against null inside the function.

-Wnonnull-compare is included in **-Wall**. It can be disabled with the **-Wno-nonnull-compare** option.

-Wnull-dereference

Warn if the compiler detects paths that trigger erroneous or undefined behavior due to dereferencing a null pointer. This option is only active when **-fdelete-null-pointer-checks** is active, which is enabled by optimizations in most targets. The precision of the warnings depends on the optimization options used.

-Wno-musttail-local-addr

Do not warn about passing a pointer (or in C++, a reference) to a local variable or label to argument of a **musttail** call. Those variables go out of scope before the tail call instruction.

-Wmaybe-musttail-local-addr

Warn when address of a local variable can escape to a **musttail** call, unless it goes out of scope already before the **musttail** call.

```
int foo (int *);

int
bar (int *x)
{
    if (x[0] == 1)
    {
        int a = 42;
        foo (&a);
        /* Without the musttail attribute this call would not
           be tail called, because address of the a variable escapes
           and the second foo call could dereference it. With the attribute
           the local variables are assumed to go out of scope immediately
           before the tail call instruction and the compiler warns about
           this. */
        [[gnu::musttail]] return foo (nullptr);
    }
}
```

```

else
{
    {
        int a = 42;
        foo (&a);
    }
    /* The a variable isn't already in scope, so even when it
       escaped, even without musttail attribute it would be
       undefined behavior to dereference it and the compiler could
       turn this into a tail call. No warning is diagnosed here. */
    [[gnu::musttail]] return foo (nullptr);
}
}

```

This warning is enabled by `-Wextra`.

`-Wnrvo` (C++ and Objective-C++ only)

Warn if the compiler does not elide the copy from a local variable to the return value of a function in a context where it is allowed by [class.copy.elision]. This elision is commonly known as the Named Return Value Optimization. For instance, in the example below the compiler cannot elide copies from both `v1` and `v2`, so it elides neither.

```

std::vector<int> f()
{
    std::vector<int> v1, v2;
    // ...
    if (cond) return v1;
    else return v2; // warning: not eliding copy
}

```

`-Winfinite-recursion`

Warn about infinitely recursive calls. The warning is effective at all optimization levels but requires optimization in order to detect infinite recursion in calls between two or more functions. `-Winfinite-recursion` is included in `-Wall`.

Compare with `-Wanalyzer-infinite-recursion` which provides a similar diagnostic, but is implemented in a different way (as part of `-fanalyzer`).

`-Winit-self` (C, C++, Objective-C and Objective-C++ only)

Warn about uninitialized variables that are initialized with themselves. Note this option can only be used with the `-Wuninitialized` option.

For example, GCC warns about `i` being uninitialized in the following snippet only when `-Winit-self` has been specified:

```

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

```

This warning is enabled by `-Wall` in C++.

`-Wno-implicit-int` (C and Objective-C only)

This option controls warnings when a declaration does not specify a type. This warning is enabled by default, as an error, in C99 and later dialects of C, and also by `-Wall`. The error can be downgraded to a warning using

`-fpermissive` (along with certain other errors), or for this error alone, with `-Wno-error=implicit-int`.

This warning is upgraded to an error by `-pedantic-errors`.

`-Wno-implicit-function-declaration` (C and Objective-C only)

This option controls warnings when a function is used before being declared. This warning is enabled by default, as an error, in C99 and later dialects of C, and also by `-Wall`. The error can be downgraded to a warning using `-fpermissive` (along with certain other errors), or for this error alone, with `-Wno-error=implicit-function-declaration`.

This warning is upgraded to an error by `-pedantic-errors`.

`-Wimplicit` (C and Objective-C only)

Same as `-Wimplicit-int` and `-Wimplicit-function-declaration`. This warning is enabled by `-Wall`.

`-Whardened`

Warn when `-fhardened` did not enable an option from its set (for which see `-fhardened`). For instance, using `-fhardened` and `-fstack-protector` at the same time on the command line causes `-Whardened` to warn because `-fstack-protector-strong` will not be enabled by `-fhardened`.

This warning is enabled by default and has effect only when `-fhardened` is enabled.

`-Wimplicit-fallthrough`

`-Wimplicit-fallthrough` is the same as `-Wimplicit-fallthrough=3` and `-Wno-implicit-fallthrough` is the same as `-Wimplicit-fallthrough=0`.

`-Wimplicit-fallthrough=n`

Warn when a switch case falls through. For example:

```
switch (cond)
{
  case 1:
    a = 1;
    break;
  case 2:
    a = 2;
  case 3:
    a = 3;
    break;
}
```

This warning does not warn when the last statement of a case cannot fall through, e.g. when there is a return statement or a call to function declared with the `noreturn` attribute. `-Wimplicit-fallthrough=` also takes into account control flow statements, such as ifs, and only warns when appropriate. E.g.

```

switch (cond)
{
  case 1:
    if (i > 3) {
      bar (5);
      break;
    } else if (i < 1) {
      bar (0);
    } else
      return;
  default:
    ...
}

```

Since there are occasions where a switch case fall through is desirable, GCC provides an attribute, `__attribute__((fallthrough))`, that is to be used along with a null statement to suppress this warning that would normally occur:

```

switch (cond)
{
  case 1:
    bar (0);
    __attribute__((fallthrough));
  default:
    ...
}

```

C++17 and C23 provide a standard way to suppress the `-Wimplicit-fallthrough` warning using `[[fallthrough]]`; instead of the GNU attribute. In C++11 or C++14 users can use `[[gnu::fallthrough]]`; which is a GNU extension. Instead of these attributes, it is also possible to add a fallthrough comment to silence the warning. The whole body of the C or C++ style comment should match the given regular expressions listed below. The option argument *n* specifies what kind of comments are accepted:

- `-Wimplicit-fallthrough=0` disables the warning altogether.
- `-Wimplicit-fallthrough=1` matches `.*` regular expression, any comment is used as fallthrough comment.
- `-Wimplicit-fallthrough=2` case insensitively matches `.*falls?[\t-]*thr(ough|u).*` regular expression.
- `-Wimplicit-fallthrough=3` case sensitively matches one of the following regular expressions:
 - `-fallthrough`
 - `@fallthrough@`
 - `lint -fallthrough[\t]*`
 - `[\t.!?]*(ELSE,? |INTENTIONAL(LY)?)? FALL(S | |-)?THR(OUGH|U)[\t.!?]*(-[^\n\r]*)?`
 - `[\t.!?]*(Else,? |Intentional(ly)?)? Fall((s | |-)[Tt]|t)hr(ough|u)[\t.!?]*(-[^\n\r]*)?`
 - `[\t.!?]*([Ee]lse,? |[Ii]ntentional(ly)?)? fall(s | |-)?thr(ough|u)[\t.!?]*(-[^\n\r]*)?`

- `-Wimplicit-fallthrough=4` case sensitively matches one of the following regular expressions:
 - `-fallthrough`
 - `@fallthrough@`
 - `lint -fallthrough[\t]*`
 - `[\t]*FALLTHR(OUGH|U)[\t]*`
- `-Wimplicit-fallthrough=5` doesn't recognize any comments as fallthrough comments, only attributes disable the warning.

The comment needs to be followed after optional whitespace and other comments by `case` or `default` keywords or by a user label that precedes some `case` or `default` label.

```
switch (cond)
{
  case 1:
    bar (0);
    /* FALLTHRU */
  default:
    ...
}
```

The `-Wimplicit-fallthrough=3` warning is enabled by `-Wextra`.

-Wno-if-not-aligned (C, C++, Objective-C and Objective-C++ only)

Control if warnings triggered by the `warn_if_not_aligned` attribute should be issued. These warnings are enabled by default.

-Wignored-qualifiers (C and C++ only)

Warn if the return type of a function has a type qualifier such as `const`. For ISO C such a type qualifier has no effect, since the value returned by a function is not an lvalue. For C++, the warning is only emitted for scalar types or `void`. ISO C prohibits qualified `void` return types on function definitions, so such return types always receive a warning even without this option.

This warning is also enabled by `-Wextra`.

-Wno-ignored-attributes (C and C++ only)

This option controls warnings when an attribute is ignored. This is different from the `-Wattributes` option in that it warns whenever the compiler decides to drop an attribute, not that the attribute is either unknown, used in a wrong place, etc. This warning is enabled by default.

-Wmain

Warn if the type of `main` is suspicious. `main` should be a function with external linkage, returning `int`, taking either zero arguments, two, or three arguments of appropriate types. This warning is enabled by default in C++ and is enabled by either `-Wall` or `-Wpedantic`.

This warning is upgraded to an error by `-pedantic-errors`.

-Wmisleading-indentation (C and C++ only)

Warn when the indentation of the code does not reflect the block structure. Specifically, a warning is issued for `if`, `else`, `while`, and `for` clauses with a

guarded statement that does not use braces, followed by an unguarded statement with the same indentation.

In the following example, the call to “bar” is misleadingly indented as if it were guarded by the “if” conditional.

```
if (some_condition ())
    foo ();
    bar (); /* Gotcha: this is not guarded by the "if". */
```

In the case of mixed tabs and spaces, the warning uses the `-ftabstop=` option to determine if the statements line up (defaulting to 8).

The warning is not issued for code involving multiline preprocessor logic such as the following example.

```
if (flagA)
    foo (0);
#ifdef SOME_CONDITION_THAT_DOES_NOT_HOLD
    if (flagB)
#endif
    foo (1);
```

The warning is not issued after a `#line` directive, since this typically indicates autogenerated code, and no assumptions can be made about the layout of the file that the directive references.

This warning is enabled by `-Wall` in C and C++.

-Wmissing-attributes

Warn when a declaration of a function is missing one or more attributes that a related function is declared with and whose absence may adversely affect the correctness or efficiency of generated code. For example, the warning is issued for declarations of aliases that use attributes to specify less restrictive requirements than those of their targets. This typically represents a potential optimization opportunity. By contrast, the `-Wattribute-alias=2` option controls warnings issued when the alias is more restrictive than the target, which could lead to incorrect code generation. Attributes considered include `alloc_align`, `alloc_size`, `cold`, `const`, `hot`, `leaf`, `malloc`, `nonnull`, `noreturn`, `nothrow`, `pure`, `returns_nonnull`, and `returns_twice`.

In C++, the warning is issued when an explicit specialization of a primary template declared with attribute `alloc_align`, `alloc_size`, `assume_aligned`, `format`, `format_arg`, `malloc`, or `nonnull` is declared without it. Attributes `deprecated`, `error`, and `warning` suppress the warning. (see Section 6.4.1 [Common Attributes], page 595).

You can use the `copy` attribute to apply the same set of attributes to a declaration as that on another declaration without explicitly enumerating the attributes. This attribute can be applied to declarations of functions, variables, or types. Section 6.4.1 [Common Attributes], page 595.

`-Wmissing-attributes` is enabled by `-Wall`.

For example, since the declaration of the primary function template below makes use of both attribute `malloc` and `alloc_size` the declaration of the explicit specialization of the template is diagnosed because it is missing one of the attributes.

```

template <class T>
T* __attribute__((malloc, alloc_size (1)))
allocate (size_t);

template <>
void* __attribute__((malloc)) // missing alloc_size
allocate<void> (size_t);

```

-Wmissing-braces

Warn if an aggregate or union initializer is not fully bracketed. In the following example, the initializer for `a` is not fully bracketed, but that for `b` is fully bracketed.

```

int a[2][2] = { 0, 1, 2, 3 };
int b[2][2] = { { 0, 1 }, { 2, 3 } };

```

This warning is enabled by `-Wall`.

-Wmissing-include-dirs (C, C++, Objective-C, Objective-C++ and Fortran only)

Warn if a user-supplied include directory does not exist. This option is disabled by default for C, C++, Objective-C and Objective-C++. For Fortran, it is partially enabled by default by warning for `-I` and `-J`, only.

-Wno-missing-profile

This option controls warnings if feedback profiles are missing when using the `-fprofile-use` option. This option diagnoses those cases where a new function or a new file is added between compiling with `-fprofile-generate` and with `-fprofile-use`, without regenerating the profiles. In these cases, the profile feedback data files do not contain any profile feedback information for the newly added function or file respectively. Also, in the case when profile count data (`.gcda`) files are removed, GCC cannot use any profile feedback information. In all these cases, warnings are issued to inform you that a profile generation step is due. Ignoring the warning can result in poorly optimized code. `-Wno-missing-profile` can be used to disable the warning, but this is not recommended and should be done only when non-existent profile data is justified.

-Wmismatched-dealloc

Warn for calls to deallocation functions with pointer arguments returned from allocation functions for which the former isn't a suitable deallocator. A pair of functions can be associated as matching allocators and deallocators by use of attribute `malloc`. Unless disabled by the `-fno-builtin` option the standard functions `calloc`, `malloc`, `realloc`, and `free`, as well as the corresponding forms of C++ operator `new` and operator `delete` are implicitly associated as matching allocators and deallocators. In the following example `mydealloc` is the deallocator for pointers returned from `myalloc`.

```

void mydealloc (void*);

__attribute__((malloc (mydealloc, 1))) void*
myalloc (size_t);

void f (void)
{
    void *p = myalloc (32);
    // ...use p...
}

```

```

    free (p);    // warning: not a matching deallocator for myalloc
    mydealloc (p); // ok
}

```

In C++, the related option `-Wmismatched-new-delete` diagnoses mismatches involving either operator `new` or operator `delete`.

Option `-Wmismatched-dealloc` is included in `-Wall`.

`-Wmultistatement-macros`

Warn about unsafe multiple statement macros that appear to be guarded by a clause such as `if`, `else`, `for`, `switch`, or `while`, in which only the first statement is actually guarded after the macro is expanded.

For example:

```

#define DOIT x++; y++
if (c)
    DOIT;

```

will increment `y` unconditionally, not just when `c` holds. The can usually be fixed by wrapping the macro in a do-while loop:

```

#define DOIT do { x++; y++; } while (0)
if (c)
    DOIT;

```

This warning is enabled by `-Wall` in C and C++.

`-Wparentheses`

Warn if parentheses are omitted in certain contexts, such as when there is an assignment in a context where a truth value is expected, or when operators are nested whose precedence people often get confused about.

Also warn if a comparison like `x<=y<=z` appears; this is equivalent to `(x<=y ? 1 : 0) <= z`, which is a different interpretation from that of ordinary mathematical notation.

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

For C++ this also warns for some cases of unnecessary parentheses in declarations, which can indicate an attempt at a function call instead of a declaration:

```

{
    // Declares a local variable called mymutex.
    std::unique_lock<std::mutex> (mymutex);
    // User meant std::unique_lock<std::mutex> lock (mymutex);
}

```

This warning is enabled by `-Wall`.

`-Wno-self-move` (C++ and Objective-C++ only)

This warning warns when a value is moved to itself with `std::move`. Such a `std::move` typically has no effect.

```

struct T {
    ...
};
void fn()

```

```

{
    T t;
    ...
    t = std::move (t);
}

```

This warning is enabled by `-Wall`.

`-Wsequence-point`

Warn about code that may have undefined semantics because of violations of sequence point rules in the C and C++ standards.

The C and C++ standards define the order in which expressions in a C/C++ program are evaluated in terms of *sequence points*, which represent a partial ordering between the execution of parts of the program: those executed before the sequence point, and those executed after it. These occur after the evaluation of a full expression (one which is not part of a larger expression), after the evaluation of the first operand of a `&&`, `||`, `? :` or `,` (comma) operator, before a function is called (but after the evaluation of its arguments and the expression denoting the called function), and in certain other places. Other than as expressed by the sequence point rules, the order of evaluation of subexpressions of an expression is not specified. All these rules describe only a partial order rather than a total order, since, for example, if two functions are called within one expression with no sequence point between them, the order in which the functions are called is not specified. However, the standards committee have ruled that function calls do not overlap.

It is not specified when between sequence points modifications to the values of objects take effect. Programs whose behavior depends on this have undefined behavior; the C and C++ standards specify that “Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.”. If a program breaks these rules, the results on any particular implementation are entirely unpredictable.

Examples of code with undefined behavior are `a = a++;`, `a[n] = b[n++]` and `a[i++] = i;`. Some more complicated cases are not diagnosed by this option, and it may give an occasional false positive result, but in general it has been found fairly effective at detecting this sort of problem in programs.

The C++17 standard will define the order of evaluation of operands in more cases: in particular it requires that the right-hand side of an assignment be evaluated before the left-hand side, so the above examples are no longer undefined. But this option will still warn about them, to help people avoid writing code that is undefined in C and earlier revisions of C++.

The standard is worded confusingly, therefore there is some debate over the precise meaning of the sequence point rules in subtle cases. Links to discussions of the problem, including proposed formal definitions, may be found on the GCC readings page, at <https://gcc.gnu.org/readings.html>.

This warning is enabled by `-Wall` for C and C++.

-Wno-return-local-addr

Do not warn about returning a pointer (or in C++, a reference) to a variable that goes out of scope after the function returns.

-Wreturn-mismatch

Warn about return statements without an expressions in functions which do not return `void`. Also warn about a `return` statement with an expression in a function whose return type is `void`, unless the expression type is also `void`. As a GNU extension, the latter case is accepted without a warning unless `-Wpedantic` is used.

Attempting to use the return value of a non-`void` function other than `main` that flows off the end by reaching the closing curly brace that terminates the function is undefined.

This warning is specific to C and enabled by default. In C99 and later language dialects, it is treated as an error. It can be downgraded to a warning using `-fpermissive` (along with other warnings), or for just this warning, with `-Wno-error=return-mismatch`.

-Wreturn-type

Warn whenever a function is defined with a return type that defaults to `int` (unless `-Wimplicit-int` is active, which takes precedence). Also warn if execution may reach the end of the function body, or if the function does not contain any return statement at all.

Attempting to use the return value of a non-`void` function other than `main` that flows off the end by reaching the closing curly brace that terminates the function is undefined.

Unlike in C, in C++, flowing off the end of a non-`void` function other than `main` results in undefined behavior even when the value of the function is not used.

This warning is enabled by default in C++ and by `-Wall` otherwise.

-Wno-shift-count-negative

Controls warnings if a shift count is negative. This warning is enabled by default.

-Wno-shift-count-overflow

Controls warnings if a shift count is greater than or equal to the bit width of the type. This warning is enabled by default.

-Wshift-negative-value

Warn if left shifting a negative value. This warning is enabled by `-Wextra` in C99 (and newer) and C++11 to C++17 modes.

-Wno-shift-overflow**-Wshift-overflow=n**

These options control warnings about left shift overflows.

-Wshift-overflow=1

This is the warning level of `-Wshift-overflow` and is enabled by default in C99 and C++11 modes (and newer). This warning level does not warn about left-shifting 1 into the sign bit. (However, in

C, such an overflow is still rejected in contexts where an integer constant expression is required.) No warning is emitted in C++20 mode (and newer), as signed left shifts always wrap.

-Wshift-overflow=2

This warning level also warns about left-shifting 1 into the sign bit, unless C++14 mode (or newer) is active.

-Wswitch Warn whenever a **switch** statement has an index of enumerated type and lacks a **case** for one or more of the named codes of that enumeration. (The presence of a **default** label prevents this warning.) **case** labels that do not correspond to enumerators also provoke warnings when this option is used, unless the enumeration is marked with the **flag_enum** attribute. This warning is enabled by **-Wall**.

-Wswitch-default

Warn whenever a **switch** statement does not have a **default** case.

-Wswitch-enum

Warn whenever a **switch** statement has an index of enumerated type and lacks a **case** for one or more of the named codes of that enumeration. **case** labels that do not correspond to enumerators also provoke warnings when this option is used, unless the enumeration is marked with the **flag_enum** attribute. The only difference between **-Wswitch** and this option is that this option gives a warning about an omitted enumeration code even if there is a **default** label.

-Wno-switch-bool

Do not warn when a **switch** statement has an index of boolean type and the case values are outside the range of a boolean type. It is possible to suppress this warning by casting the controlling expression to a type other than **bool**. For example:

```
switch ((int) (a == 4))
{
    ...
}
```

This warning is enabled by default for C and C++ programs.

-Wno-switch-outside-range

This option controls warnings when a **switch** case has a value that is outside of its respective type range. This warning is enabled by default for C and C++ programs.

-Wno-switch-unreachable

Do not warn when a **switch** statement contains statements between the controlling expression and the first case label, which will never be executed. For example:

```
switch (cond)
{
    i = 15;
    ...
    case 5:
    ...
}
```

`-Wswitch-unreachable` does not warn if the statement between the controlling expression and the first case label is just a declaration:

```
switch (cond)
{
    int i;
    ...
    case 5:
        i = 5;
    ...
}
```

This warning is enabled by default for C and C++ programs.

`-Wsync-nand` (C and C++ only)

Warn when `__sync_fetch_and_nand` and `__sync_nand_and_fetch` built-in functions are used. These functions changed semantics in GCC 4.4.

`-Wtrivial-auto-var-init`

Warn when `-ftrivial-auto-var-init` cannot initialize the automatic variable. A common situation is an automatic variable that is declared between the controlling expression and the first case label of a `switch` statement.

`-Wunused-but-set-parameter`

`-Wunused-but-set-parameter` is the same as `-Wunused-but-set-parameter=3` and `-Wno-unused-but-set-parameter` is the same as `-Wunused-but-set-parameter=0`.

`-Wunused-but-set-parameter=n`

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 6.4.1 [Common Attributes], page 595).

`-Wunused-but-set-parameter=0` disables the warning. With `-Wunused-but-set-parameter=1` all uses except initialization and left hand side of assignment which is not further used disable the warning. With `-Wunused-but-set-parameter=2` additionally uses of parameter in `++` and `--` operators don't count as uses. And finally with `-Wunused-but-set-parameter=3` additionally uses in `parm @= rhs` outside of `rhs` don't count as uses. See `-Wunused-but-set-variable=n` option for examples.

This `-Wunused-but-set-parameter=3` warning is also enabled by `-Wunused` together with `-Wextra`.

`-Wunused-but-set-variable`

`-Wunused-but-set-variable` is the same as `-Wunused-but-set-variable=3` and `-Wno-unused-but-set-variable` is the same as `-Wunused-but-set-variable=0`.

`-Wunused-but-set-variable=n`

Warn whenever a local variable is assigned to, but otherwise unused (aside from its declaration). This `-Wunused-but-set-variable=3` warning is enabled by `-Wall`.

To suppress this warning use the `unused` attribute (see Section 6.4.1 [Common Attributes], page 595).

`-Wunused-but-set-variable=0` disables the warning. With `-Wunused-but-set-variable=1` all uses except initialization and left hand side of assignment which is not further used disable the warning. With `-Wunused-but-set-variable=2` additionally uses of variable in `++` and `--` operators don't count as uses. And finally with `-Wunused-but-set-variable=3` additionally uses in `parm @= rhs` outside of `rhs` don't count as uses.

This `-Wunused-but-set-variable=3` warning is also enabled by `-Wunused`, which is enabled by `-Wall`.

```
void foo (void)
{
    int a = 1; // -Wunused-variable warning
    int b = 0; // Warning for n >= 1
    b = 1; b = 2;
    int c = 0; // Warning for n >= 2
    ++c; c--; --c; c++;
    int d = 0; // Warning for n >= 3
    d += 4;
    int e = 0; // No warning, cast to void
    (void) e;
    int f = 0; // No warning, f used
    int g = f = 5;
    (void) g;
    int h = 0; // No warning, preincrement result used
    int i = ++h;
    (void) i;
    int j = 0; // No warning, postdecrement result used
    int k = j--;
    (void) k;
    int l = 0; // No warning, l used
    int m = l |= 2;
    (void) m;
}
```

`-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 6.4.1 [Common Attributes], page 595).

`-Wunused-local-typedefs` (C, Objective-C, C++ and Objective-C++ only)

Warn when a typedef locally defined in a function is not used. This warning is enabled by `-Wall`.

`-Wunused-parameter`

Warn whenever a function parameter is unused aside from its declaration. This option is not enabled by `-Wunused` unless `-Wextra` is also specified.

To suppress this warning use the `unused` attribute (see Section 6.4.1 [Common Attributes], page 595).

-Wno-unused-result

Do not warn if a caller of a function marked with attribute `warn_unused_result` (see Section 6.4.1 [Common Attributes], page 595) does not use its return value. The default is `-Wunused-result`.

-Wunused-variable

Warn whenever a local or static variable is unused aside from its declaration. This option implies `-Wunused-const-variable=1` for C, but not for C++. This warning is enabled by `-Wall`.

To suppress this warning use the `unused` attribute (see Section 6.4.1 [Common Attributes], page 595).

-Wunused-const-variable

-Wunused-const-variable=n

Warn whenever a constant static variable is unused aside from its declaration.

To suppress this warning use the `unused` attribute (see Section 6.4.1 [Common Attributes], page 595).

-Wunused-const-variable=1

Warn about unused static const variables defined in the main compilation unit, but not about static const variables declared in any header included.

`-Wunused-const-variable=1` is enabled by either `-Wunused-variable` or `-Wunused` for C, but not for C++. In C, this declares variable storage, but in C++, this is not an error since const variables take the place of `#defines`.

-Wunused-const-variable=2

This warning level also warns for unused constant static variables in headers (excluding system headers). It is equivalent to the short form `-Wunused-const-variable`. This level must be explicitly requested in both C and C++ because it might be hard to clean up all headers included.

-Wunused-value

Warn whenever a statement computes a result that is explicitly not used. To suppress this warning cast the unused expression to `void`. This includes an expression-statement or the left-hand side of a comma expression that contains no side effects. For example, an expression such as `x[i,j]` causes a warning, while `x[(void)i,j]` does not.

This warning is enabled by `-Wall`.

-Wunused All the above `-Wunused` options combined, except those documented as needing to be specified explicitly.

In order to get a warning about an unused function parameter, you must either specify `-Wextra -Wunused` (note that `-Wall` implies `-Wunused`), or separately specify `-Wunused-parameter` and/or `-Wunused-but-set-parameter`.

`-Wunused` enables only `-Wunused-const-variable=1` rather than `-Wunused-const-variable`, and only for C, not C++.

`-Wuse-after-free` (C, Objective-C, C++ and Objective-C++ only)

`-Wuse-after-free=n`

Warn about uses of pointers to dynamically allocated objects that have been rendered indeterminate by a call to a deallocation function. The warning is enabled at all optimization levels but may yield different results with optimization than without.

`-Wuse-after-free=1`

At level 1 the warning attempts to diagnose only unconditional uses of pointers made indeterminate by a deallocation call or a successful call to `realloc`, regardless of whether or not the call resulted in an actual reallocation of memory. This includes double-`free` calls as well as uses in arithmetic and relational expressions. Although undefined, uses of indeterminate pointers in equality (or inequality) expressions are not diagnosed at this level.

`-Wuse-after-free=2`

At level 2, in addition to unconditional uses, the warning also diagnoses conditional uses of pointers made indeterminate by a deallocation call. As at level 2, uses in equality (or inequality) expressions are not diagnosed. For example, the second call to `free` in the following function is diagnosed at this level:

```
struct A { int refcount; void *data; };

void release (struct A *p)
{
    int refcount = --p->refcount;
    free (p);
    if (refcount == 0)
        free (p->data);    // warning: p may be used after free
}
```

`-Wuse-after-free=3`

At level 3, the warning also diagnoses uses of indeterminate pointers in equality expressions. All uses of indeterminate pointers are undefined but equality tests sometimes appear after calls to `realloc` as an attempt to determine whether the call resulted in relocating the object to a different address. They are diagnosed at a separate level to aid gradually transitioning legacy code to safe alternatives. For example, the equality test in the function below is diagnosed at this level:

```
void adjust_pointers (int**, int);

void grow (int **p, int n)
{
    int **q = (int**)realloc (p, n * 2);
    if (q == p)
        return;
    adjust_pointers ((int**)q, n);
}
```

```
    }
```

To avoid the warning at this level, store offsets into allocated memory instead of pointers. This approach obviates needing to adjust the stored pointers after reallocation.

`-Wuse-after-free=2` is included in `-Wall`.

`-Wuseless-cast` (C, Objective-C, C++ and Objective-C++ only)

Warn when an expression is cast to its own type. This warning does not occur when a class object is converted to a non-reference type as that is a way to create a temporary:

```
struct S { };
void g (S&&);
void f (S&& arg)
{
    g (S(arg)); // make arg prvalue so that it can bind to S&&
}
```

`-Wuninitialized`

Warn if an object with automatic or allocated storage duration is used without having been initialized. In C++, also warn if a non-static reference or non-static `const` member appears in a class without constructors.

In addition, passing a pointer (or in C++, a reference) to an uninitialized object to a `const`-qualified argument of a built-in function known to read the object is also diagnosed by this warning. (`-Wmaybe-uninitialized` is issued for ordinary functions.)

If you want to warn about code that uses the uninitialized value of the variable in its own initializer, use the `-Winit-self` option.

These warnings occur for individual uninitialized elements of structure, union or array variables as well as for variables that are uninitialized as a whole. They do not occur for variables or elements declared `volatile`. Because these warnings depend on optimization, the exact variables or elements for which there are warnings depend on the precise optimization options and version of GCC used.

Note that there may be no warning about a variable that is used only to compute a value that itself is never used, because such computations may be deleted by data flow analysis before the warnings are printed.

In C++, this warning also warns about using uninitialized objects in member-initializer-lists. For example, GCC warns about `b` being uninitialized in the following snippet:

```
struct A {
    int a;
    int b;
    A() : a(b) { }
};
```

`-Wno-invalid-memory-model`

This option controls warnings for invocations of Section 7.9.1 [`__atomic` Builtins], page 822, Section 7.9.2 [`__sync` Builtins], page 827, and the C11 atomic generic functions with a memory consistency argument that is either

invalid for the operation or outside the range of values of the `memory_order` enumeration. For example, since the `__atomic_store` and `__atomic_store_n` built-ins are only defined for the relaxed, release, and sequentially consistent memory orders the following code is diagnosed:

```
void store (int *i)
{
    __atomic_store_n (i, 0, memory_order_consume);
}
```

`-Winvalid-memory-model` is enabled by default.

`-Wmaybe-uninitialized`

For an object with automatic or allocated storage duration, if there exists a path from the function entry to a use of the object that is initialized, but there exist some other paths for which the object is not initialized, the compiler emits a warning if it cannot prove the uninitialized paths are not executed at run time.

In addition, passing a pointer (or in C++, a reference) to an uninitialized object to a `const`-qualified function argument is also diagnosed by this warning. (`-Wuninitialized` is issued for built-in functions known to read the object.) Annotating the function with attribute `access (none)` indicates that the argument isn't used to access the object and avoids the warning (see Section 6.4.1 [Common Attributes], page 595).

These warnings are only possible in optimizing compilation, because otherwise GCC does not keep track of the state of variables. On the other hand, `-Wmaybe-uninitialized` is known not to warn in many situations (false negatives) due to optimizations taking advantage of undefinedness of uninitialized uses like constant propagation.

These warnings are made optional because GCC may not be able to determine when the code is correct in spite of appearing to have an error. Here is one example of how this can happen:

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

If the value of `y` is always 1, 2 or 3, then `x` is always initialized, but GCC doesn't know this. To suppress the warning, you need to provide a default case with `assert(0)` or similar code.

This option also warns when a non-volatile automatic variable might be changed by a call to `longjmp`. The compiler sees only the calls to `setjmp`. It cannot know where `longjmp` will be called; in fact, a signal handler could call it at any point in the code. As a result, you may get a warning even when there is in fact

no problem because `longjmp` cannot in fact be called at the place that would cause a problem.

Some spurious warnings can be avoided if you declare all the functions you use that never return as `noreturn`. See Section 6.4.1 [Common Attributes], page 595.

This warning is enabled by `-Wall` or `-Wextra`.

`-Wunknown-pragmas`

Warn when a `#pragma` directive is encountered that is not understood by GCC. If this command-line option is used, warnings are even issued for unknown pragmas in system header files. This is not the case if the warnings are only enabled by the `-Wall` command-line option.

`-Wno-pragmas`

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

`-Wno-pragma-once-outside-header`

Do not warn when `#pragma once` is used in a file that is not a header file, such as a main file.

`-Wno-prio-ctor-dtor`

Do not warn if a priority from 0 to 100 is used for constructor or destructor. The use of constructor and destructor attributes allow you to assign a priority to the constructor/destructor to control its order of execution before `main` is called or after it returns. The priority values must be greater than 100 as the compiler reserves priority values between 0–100 for the implementation.

`-Wstrict-aliasing`

This option is only active when `-fstrict-aliasing` is active. It warns about code that might break the strict aliasing rules that the compiler is using for optimization. The warning does not catch all cases, but does attempt to catch the more common pitfalls. It is included in `-Wall`. It is equivalent to `-Wstrict-aliasing=3`.

`-Wstrict-aliasing=n`

This option is only active when `-fstrict-aliasing` is active. It warns about code that might break the strict aliasing rules that the compiler is using for optimization. Higher levels correspond to higher accuracy (fewer false positives). Higher levels also correspond to more effort, similar to the way `-O` works. `-Wstrict-aliasing` is equivalent to `-Wstrict-aliasing=3`.

Level 1: Most aggressive, quick, least accurate. Possibly useful when higher levels do not warn but `-fstrict-aliasing` still breaks the code, as it has very few false negatives. However, it has many false positives. Warns for all pointer conversions between possibly incompatible types, even if never dereferenced. Runs in the front end only.

Level 2: Aggressive, quick, not too precise. May still have many false positives (not as many as level 1 though), and few false negatives (but possibly more than level 1). Unlike level 1, it only warns when an address is taken. Warns about incomplete types. Runs in the front end only.

Level 3 (default for `-Wstrict-aliasing`): Should have very few false positives and few false negatives. Slightly slower than levels 1 or 2 when optimization is enabled. Takes care of the common pun+dereference pattern in the front end: `*(int*)&some_float`. If optimization is enabled, it also runs in the back end, where it deals with multiple statement cases using flow-sensitive points-to information. Only warns when the converted pointer is dereferenced. Does not warn about incomplete types.

`-Wstrict-overflow`

`-Wstrict-overflow=n`

Does nothing. Preserved for backward compatibility.

`-Wstrict-overflow=1`

Does nothing. Preserved for backward compatibility.

`-Wstrict-overflow=2`

Does nothing. Preserved for backward compatibility.

`-Wstrict-overflow=3`

Does nothing. Preserved for backward compatibility.

`-Wstrict-overflow=4`

Does nothing. Preserved for backward compatibility.

`-Wstrict-overflow=5`

Does nothing. Preserved for backward compatibility.

`-Wstring-compare`

Warn for calls to `strcmp` and `strncmp` whose result is determined to be either zero or non-zero in tests for such equality owing to the length of one argument being greater than the size of the array the other argument is stored in (or the bound in the case of `strncmp`). Such calls could be mistakes. For example, the call to `strcmp` below is diagnosed because its result is necessarily non-zero irrespective of the contents of the array `a`.

```
extern char a[4];
void f (char *d)
{
    strcpy (d, "string");
    ...
    if (0 == strcmp (a, d))    // cannot be true
        puts ("a and d are the same");
}
```

`-Wstring-compare` is enabled by `-Wextra`.

`-Wno-stringop-overflow`

`-Wstringop-overflow`

`-Wstringop-overflow=type`

Warn for code that can be statically determined to cause buffer overflows or memory overruns, such as calls to `memcpy` and `strcpy` that overflow the destination buffer. The optional argument is one greater than the type of Object Size Checking to perform to determine the size of the destination. See Section 7.10 [Object Size Checking], page 830. The argument is meaningful only for string

functions that operate on character arrays; raw memory functions like `memcpy` always use type-zero Object Size Checking.

The option also warns for calls that specify a size in excess of the largest possible object or at most `SIZE_MAX / 2` bytes.

The option produces the best results with optimization enabled but can detect a small subset of simple buffer overflows even without optimization in calls to the GCC built-in functions like `__builtin_memcpy` that correspond to the standard functions. In any case, the option warns about just a subset of buffer overflows detected by the corresponding overflow checking built-ins, such as `__builtin___memcpy_chk`, which can perform run-time checking if the access cannot be identified as safe at compile time.

For example, the option issues a warning for the `strcpy` call below because it copies at least 5 characters (the string "blue" including the terminating NUL) into the buffer of size 4.

```
enum Color { blue, purple, yellow };
const char* f (enum Color clr)
{
    static char buf [4];
    const char *str;
    switch (clr)
    {
        case blue: str = "blue"; break;
        case purple: str = "purple"; break;
        case yellow: str = "yellow"; break;
    }

    return strcpy (buf, str);    // warning here
}
```

The effect of this option is not limited to string or memory manipulation functions. In this example, a warning is diagnosed because a 1-element array is passed to a function requiring at least a 4-element array argument:

```
void f (int[static 4]);

void g (void)
{
    int *p = (int *) malloc (1 * sizeof(int));
    f (p);    // warning here
}
```

Option `-Wstringop-overflow=2` is enabled by default.

`-Wstringop-overflow`

`-Wstringop-overflow=1`

The `-Wstringop-overflow=1` option uses type-zero Object Size Checking to determine the sizes of destination objects. At this setting the option does not warn for writes past the end of subobjects of larger objects accessed by pointers unless the size of the largest surrounding object is known. When the destination may be one of several objects it is assumed to be the largest one of them. On Linux systems, when optimization is enabled at this setting

the option warns for the same code as when the `_FORTIFY_SOURCE` macro is defined to a non-zero value.

`-Wstringop-overflow=2`

The `-Wstringop-overflow=2` option uses type-one Object Size Checking to determine the sizes of destination objects. At this setting the option warns about overflows when writing to members of the largest complete objects whose exact size is known. However, it does not warn for excessive writes to the same members of unknown objects referenced by pointers since they may point to arrays containing unknown numbers of elements. This is the default setting of the option.

`-Wstringop-overflow=3`

The `-Wstringop-overflow=3` option uses type-two Object Size Checking to determine the sizes of destination objects. At this setting the option warns about overflowing the smallest object or data member. This is the most restrictive setting of the option that may result in warnings for safe code.

`-Wstringop-overflow=4`

The `-Wstringop-overflow=4` option uses type-three Object Size Checking to determine the sizes of destination objects. At this setting the option warns about overflowing any data members, and when the destination is one of several objects it uses the size of the largest of them to decide whether to issue a warning. Similarly to `-Wstringop-overflow=3` this setting of the option may result in warnings for benign code.

`-Wno-stringop-overread`

Warn for calls to string manipulation functions such as `memchr`, or `strcpy` that are determined to read past the end of the source sequence.

Option `-Wstringop-overread` is enabled by default.

`-Wno-stringop-truncation`

Do not warn for calls to bounded string manipulation functions such as `strncat`, `strncpy`, and `stpncpy` that may either truncate the copied string or leave the destination unchanged.

In the following example, the call to `strncat` specifies a bound that is less than the length of the source string. As a result, the copy of the source will be truncated and so the call is diagnosed. To avoid the warning use `bufsize - strlen(buf) - 1` as the bound.

```
void append (char *buf, size_t bufsize)
{
    strncat (buf, ".txt", 3);
}
```

As another example, the following call to `strncpy` results in copying to `d` just the characters preceding the terminating NUL, without appending the NUL to the end. Assuming the result of `strncpy` is necessarily a NUL-terminated

string is a common mistake, and so the call is diagnosed. To avoid the warning when the result is not expected to be NUL-terminated, call `memcpy` instead.

```
void copy (char *d, const char *s)
{
    strncpy (d, s, strlen (s));
}
```

In the following example, the call to `strncpy` specifies the size of the destination buffer as the bound. If the length of the source string is equal to or greater than this size the result of the copy will not be NUL-terminated. Therefore, the call is also diagnosed. To avoid the warning, specify `sizeof buf - 1` as the bound and set the last element of the buffer to NUL.

```
void copy (const char *s)
{
    char buf[80];
    strncpy (buf, s, sizeof buf);
    ...
}
```

In situations where a character array is intended to store a sequence of bytes with no terminating NUL such an array may be annotated with attribute `nonstring` to avoid this warning. Such arrays, however, are not suitable arguments to functions that expect NUL-terminated strings. To help detect accidental misuses of such arrays GCC issues warnings unless it can prove that the use is safe. See Section 6.4.1 [Common Attributes], page 595.

`-Wstrict-flex-arrays` (C and C++ only)

Warn about improper usages of flexible array members according to the *level* of the `strict_flex_array (level)` attribute attached to the trailing array field of a structure if it's available, otherwise according to the *level* of the option `-fstrict-flex-arrays=level`. See Section 6.4.1 [Common Attributes], page 595, for more information about the attribute, and Section 3.4 [C Dialect Options], page 45, for more information about the option. `-Wstrict-flex-arrays` is effective only when *level* is greater than 0.

When *level*=1, warnings are issued for a trailing array reference of a structure that have 2 or more elements if the trailing array is referenced as a flexible array member.

When *level*=2, in addition to *level*=1, additional warnings are issued for a trailing one-element array reference of a structure if the array is referenced as a flexible array member.

When *level*=3, in addition to *level*=2, additional warnings are issued for a trailing zero-length array reference of a structure if the array is referenced as a flexible array member.

This option is more effective when `-ftree-vrp` is active (the default for `-O2` and above) but some warnings may be diagnosed even without optimization.

`-Wsuggest-attribute=attribute-name`

Warn for cases where adding an attribute may be beneficial. The *attribute-names* currently supported are listed below.

`-Wsuggest-attribute=pure`
`-Wsuggest-attribute=const`
`-Wsuggest-attribute=noreturn`
`-Wmissing-noreturn`
`-Wsuggest-attribute=malloc`
`-Wsuggest-attribute=returns_nonnull`

Warn about functions that might be candidates for attributes `pure`, `const`, `noreturn`, `malloc` or `returns_nonnull`. The compiler only warns for functions visible in other compilation units or (in the case of `pure` and `const`) if it cannot prove that the function returns normally. A function returns normally if it doesn't contain an infinite loop or return abnormally by throwing, calling `abort` or trapping. This analysis requires option `-fipa-pure-const`, which is enabled by default at `-O` and higher. Higher optimization levels improve the accuracy of the analysis.

`-Wsuggest-attribute=format`
`-Wmissing-format-attribute`

Warn about function pointers that might be candidates for `format` attributes. Note these are only possible candidates, not absolute ones. GCC guesses that function pointers with `format` attributes that are used in assignment, initialization, parameter passing or return statements should have a corresponding `format` attribute in the resulting type. I.e. the left-hand side of the assignment or initialization, the type of the parameter variable, or the return type of the containing function respectively should also have a `format` attribute to avoid the warning.

GCC also warns about function definitions that might be candidates for `format` attributes. Again, these are only possible candidates. GCC guesses that `format` attributes might be appropriate for any function that calls a function like `vprintf` or `vscanf`, but this might not always be the case, and some functions for which `format` attributes are appropriate may not be detected.

`-Wsuggest-attribute=cold`

Warn about functions that might be candidates for `cold` attribute. This is based on static detection and generally only warns about functions which always leads to a call to another `cold` function such as wrappers of C++ `throw` or fatal error reporting functions leading to `abort`.

`-Walloc-size`

Warn about calls to allocation functions decorated with attribute `alloc_size` that specify insufficient size for the target type of the pointer the result is assigned to, including those to the built-in forms of the functions `aligned_alloc`, `alloca`, `calloc`, `malloc`, and `realloc`.

-Walloc-zero

Warn about calls to allocation functions decorated with attribute `alloc_size` that specify zero bytes, including those to the built-in forms of the functions `aligned_alloc`, `alloca`, `calloc`, `malloc`, and `realloc`. Because the behavior of these functions when called with a zero size differs among implementations (and in the case of `realloc` has been deprecated) relying on it may result in subtle portability bugs and should be avoided.

-Wcalloc-transposed-args

Warn about calls to allocation functions decorated with attribute `alloc_size` with two arguments, which use `sizeof` operator as the earlier size argument and don't use it as the later size argument. This is a coding style warning. The first argument to `calloc` is documented to be number of elements in array, while the second argument is size of each element, so `calloc (n, sizeof (int))` is preferred over `calloc (sizeof (int), n)`. If `sizeof` in the earlier argument and not the latter is intentional, the warning can be suppressed by using `calloc (sizeof (struct S) + 0, n)` or `calloc (1 * sizeof (struct S), 4)` or using `sizeof` in the later argument as well.

-Walloc-size-larger-than=byte-size

Warn about calls to functions decorated with attribute `alloc_size` that attempt to allocate objects larger than the specified number of bytes, or where the result of the size computation in an integer type with infinite precision would exceed the value of 'PTRDIFF_MAX' on the target. `-Walloc-size-larger-than='PTRDIFF_MAX'` is enabled by default. Warnings controlled by the option can be disabled either by specifying `byte-size` of 'SIZE_MAX' or more or by `-Wno-alloc-size-larger-than`. See Section 6.4.1 [Common Attributes], page 595.

-Wno-alloc-size-larger-than

Disable `-Walloc-size-larger-than=` warnings. The option is equivalent to `-Walloc-size-larger-than='SIZE_MAX'` or larger.

-Walloca Warn on all uses of `alloca` in the source.**-Wauto-profile**

Output warnings about auto-profile inconsistencies.

-Wcannot-profile

Warn when profiling instrumentation was requested, but could not be applied to a certain function.

-Walloca-larger-than=byte-size

Warns on calls to `alloca` with an integer argument whose value is either zero, or that is not bounded by a controlling predicate that limits its value to at most `byte-size`. It also warns for calls to `alloca` where the bound value is unknown. Arguments of non-integer types are considered unbounded even if they appear to be constrained to the expected range.

For example, a bounded case of `alloca` could be:

```
void func (size_t n)
{
    void *p;
```

```

    if (n <= 1000)
        p = alloca (n);
    else
        p = malloc (n);
    f (p);
}

```

In the above example, passing `-Walloca-larger-than=1000` would not issue a warning because the call to `alloca` is known to be at most 1000 bytes. However, if `-Walloca-larger-than=500` were passed, the compiler would emit a warning.

Unbounded uses, on the other hand, are uses of `alloca` with no controlling predicate constraining its integer argument. For example:

```

void func ()
{
    void *p = alloca (n);
    f (p);
}

```

If `-Walloca-larger-than=500` were passed, the above would trigger a warning, but this time because of the lack of bounds checking.

Note, that even seemingly correct code involving signed integers could cause a warning:

```

void func (signed int n)
{
    if (n < 500)
    {
        p = alloca (n);
        f (p);
    }
}

```

In the above example, `n` could be negative, causing a larger than expected argument to be implicitly cast into the `alloca` call.

This option also warns when `alloca` is used in a loop.

`-Walloca-larger-than='PTRDIFF_MAX'` is enabled by default but is usually only effective when `-ftree-vrp` is active (default for `-O2` and above).

See also `-Wvla-larger-than='byte-size'`.

`-Wno-alloca-larger-than`

Disable `-Walloca-larger-than=` warnings. The option is equivalent to `-Walloca-larger-than='SIZE_MAX'` or larger.

`-Warith-conversion`

Do warn about implicit conversions from arithmetic operations even when conversion of the operands to the same type cannot change their values. This affects warnings from `-Wconversion`, `-Wfloat-conversion`, and `-Wsign-conversion`.

```

void f (char c, int i)
{
    c = c + i; // warns with -Wconversion
    c = c + 1; // only warns with -Warith-conversion
}

```

-Warray-bounds**-Warray-bounds=n**

Warn about out of bounds subscripts or offsets into arrays. This warning is enabled by **-Wall**. It is more effective when **-ftree-vrp** is active (the default for **-O2** and above) but a subset of instances are issued even without optimization.

By default, the trailing array of a structure will be treated as a flexible array member by **-Warray-bounds** or **-Warray-bounds=n** if it is declared as either a flexible array member per C99 standard onwards (`[]`), a GCC zero-length array extension (`[0]`), or an one-element array (`[1]`). As a result, out of bounds subscripts or offsets into zero-length arrays or one-element arrays are not warned by default.

You can add the option **-fstrict-flex-arrays** or **-fstrict-flex-arrays=level** to control how this option treat trailing array of a structure as a flexible array member:

when *level*≤1, no change to the default behavior.

when *level*=2, additional warnings will be issued for out of bounds subscripts or offsets into one-element arrays;

when *level*=3, in addition to *level*=2, additional warnings will be issued for out of bounds subscripts or offsets into zero-length arrays.

-Warray-bounds=1

This is the default warning level of **-Warray-bounds** and is enabled by **-Wall**; higher levels are not, and must be explicitly requested.

-Warray-bounds=2

This warning level also warns about the intermediate results of pointer arithmetic that may yield out of bounds values. This warning level may give a larger number of false positives and is deactivated by default.

-Wunterminated-string-initialization (C and Objective-C only)

Warn about character arrays initialized as unterminated character sequences with a string literal, unless the declaration being initialized has the **nonstring** attribute. For example:

```
char arr[3] = "foo"; /* Warning. */
char arr2[3] __attribute__((nonstring)) = "bar"; /* No warning. */
```

This warning is enabled by **-Wextra**. If **-Wc++-compat** is enabled, the warning has slightly different wording and warns even if the declaration being initialized has the **nonstring** warning, as in C++ such initializations are an error.

-Warray-compare

Warn about equality and relational comparisons between two operands of array type. This comparison was deprecated in C++20. For example:

```
int arr1[5];
int arr2[5];
bool same = arr1 == arr2;
```

-Warray-compare is enabled by **-Wall**.

-Warray-parameter**-Warray-parameter=n**

Warn about redeclarations of functions involving parameters of array or pointer types of inconsistent kinds or forms, and enable the detection of out-of-bounds accesses to such parameters by warnings such as **-Warray-bounds**.

If the first function declaration uses the array form for a parameter declaration, the bound specified in the array is assumed to be the minimum number of elements expected to be provided in calls to the function and the maximum number of elements accessed by it. Failing to provide arguments of sufficient size or accessing more than the maximum number of elements may be diagnosed by warnings such as **-Warray-bounds** or **-Wstringop-overflow**. At level 1, the warning diagnoses inconsistencies involving array parameters declared using the **T[static N]** form.

For example, the warning triggers for the second declaration of **f** because the first one with the keyword **static** specifies that the array argument must have at least four elements, while the second allows an array of any size to be passed to **f**.

```
void f (int[static 4]);
void f (int[]);          // warning (inconsistent array form)

void g (void)
{
    int *p = (int *)malloc (1 * sizeof (int));
    f (p);                // warning (array too small)
    ...
}
```

At level 2 the warning also triggers for redeclarations involving any other inconsistency in array or pointer argument forms denoting array sizes. Pointers and arrays of unspecified bound are considered equivalent and do not trigger a warning.

```
void g (int*);
void g (int[]);          // no warning
void g (int[8]);         // warning (inconsistent array bound)
```

-Warray-parameter=2 is included in **-Wall**. The **-Wvla-parameter** option triggers warnings for similar inconsistencies involving Variable Length Array arguments.

The short form of the option **-Warray-parameter** is equivalent to **-Warray-parameter=2**. The negative form **-Wno-array-parameter** is equivalent to **-Warray-parameter=0**.

-Wattribute-alias=n**-Wno-attribute-alias**

Warn about declarations using the **alias** and similar attributes whose target is incompatible with the type of the alias. See Section 6.4.1 [Common Attributes], page 595.

-Wattribute-alias=1

The default warning level of the **-Wattribute-alias** option diagnoses incompatibilities between the type of the alias declaration and

that of its target. Such incompatibilities are typically indicative of bugs.

`-Wattribute-alias=2`

At this level `-Wattribute-alias` also diagnoses cases where the attributes of the alias declaration are more restrictive than the attributes applied to its target. These mismatches can potentially result in incorrect code generation. In other cases they may be benign and could be resolved simply by adding the missing attribute to the target. For comparison, see the `-Wmissing-attributes` option, which controls diagnostics when the alias declaration is less restrictive than the target, rather than more restrictive.

Attributes considered include `alloc_align`, `alloc_size`, `cold`, `const`, `hot`, `leaf`, `malloc`, `nonnull`, `noreturn`, `nothrow`, `pure`, `returns_nonnull`, and `returns_twice`.

`-Wattribute-alias` is equivalent to `-Wattribute-alias=1`. This is the default. You can disable these warnings with either `-Wno-attribute-alias` or `-Wattribute-alias=0`.

`-Wbidi-chars=[none|unpaired|any|ucn]`

Warn about possibly misleading UTF-8 bidirectional control characters in comments, string literals, character constants, and identifiers. Such characters can change left-to-right writing direction into right-to-left (and vice versa), which can cause confusion between the logical order and visual order. This may be dangerous; for instance, it may seem that a piece of code is not commented out, whereas it in fact is.

There are three levels of warning supported by GCC. The default is `-Wbidi-chars=unpaired`, which warns about improperly terminated bidi contexts. `-Wbidi-chars=none` turns the warning off. `-Wbidi-chars=any` warns about any use of bidirectional control characters.

By default, this warning does not warn about UCNs. It is, however, possible to turn on such checking by using `-Wbidi-chars=unpaired,ucn` or `-Wbidi-chars=any,ucn`. Using `-Wbidi-chars=ucn` is valid, and is equivalent to `-Wbidi-chars=unpaired,ucn`, if no previous `-Wbidi-chars=any` was specified.

`-Wbool-compare`

Warn about boolean expression compared with an integer value different from `true/false`. For instance, the following comparison is always false:

```
int n = 5;
...
if ((n > 1) == 2) { ... }
```

This warning is enabled by `-Wall`.

`-Wbool-operation`

Warn about suspicious operations on expressions of a boolean type. For instance, bitwise negation of a boolean is very likely a bug in the program. For C, this warning also warns about incrementing or decrementing a boolean,

which rarely makes sense. (In C++, decrementing a boolean is always invalid. Incrementing a boolean is invalid in C++17, and deprecated otherwise.)

This warning is enabled by `-Wall`.

`-Wduplicated-branches`

Warn when an if-else has identical branches. This warning detects cases like

```
if (p != NULL)
    return 0;
else
    return 0;
```

It doesn't warn when both branches contain just a null statement. This warning also warn for conditional operators:

```
int i = x ? *p : *p;
```

`-Wduplicated-cond`

Warn about duplicated conditions in an if-else-if chain. For instance, warn for the following code:

```
if (p->q != NULL) { ... }
else if (p->q != NULL) { ... }
```

`-Wframe-address`

Warn when the `'__builtin_frame_address'` or `'__builtin_return_address'` is called with an argument greater than 0. Such calls may return indeterminate values or crash the program. The warning is included in `-Wall`.

`-Wno-discarded-qualifiers` (C and Objective-C only)

Do not warn if type qualifiers on pointers are being discarded. Typically, the compiler warns if a `const char *` variable is passed to a function that takes a `char *` parameter. This option can be used to suppress such a warning.

`-Wno-discarded-array-qualifiers` (C and Objective-C only)

Do not warn if type qualifiers on arrays which are pointer targets are being discarded. Typically, the compiler warns if a `const int (*)[]` variable is passed to a function that takes a `int (*)[]` parameter. This option can be used to suppress such a warning.

`-Wno-incompatible-pointer-types` (C and Objective-C only)

Do not warn when there is a conversion between pointers that have incompatible types. This warning is for cases not covered by `-Wno-pointer-sign`, which warns for pointer argument passing or assignment with different signedness.

By default, in C99 and later dialects of C, GCC treats this issue as an error. The error can be downgraded to a warning using `-fpermissive` (along with certain other errors), or for this error alone, with `-Wno-error=incompatible-pointer-types`.

This warning is upgraded to an error by `-pedantic-errors`.

`-Wno-int-conversion` (C and Objective-C only)

Do not warn about incompatible integer to pointer and pointer to integer conversions. This warning is about implicit conversions; for explicit conversions the warnings `-Wno-int-to-pointer-cast` and `-Wno-pointer-to-int-cast` may be used.

By default, in C99 and later dialects of C, GCC treats this issue as an error. The error can be downgraded to a warning using `-fpermissive` (along with certain other errors), or for this error alone, with `-Wno-error=int-conversion`.

This warning is upgraded to an error by `-pedantic-errors`.

`-Wzero-as-null-pointer-constant`

Warn when a literal `'0'` is used as null pointer constant.

`-Wzero-length-bounds`

Warn about accesses to elements of zero-length array members that might overlap other members of the same object. Declaring interior zero-length arrays is discouraged because accesses to them are undefined. See Section 6.2.2 [Zero Length], page 582.

For example, the first two stores in function `bad` are diagnosed because the array elements overlap the subsequent members `b` and `c`. The third store is diagnosed by `-Warray-bounds` because it is beyond the bounds of the enclosing object.

```
struct X { int a[0]; int b, c; };
struct X x;

void bad (void)
{
    x.a[0] = 0;    // -Wzero-length-bounds
    x.a[1] = 1;    // -Wzero-length-bounds
    x.a[2] = 2;    // -Warray-bounds
}
```

Option `-Wzero-length-bounds` is enabled by `-Warray-bounds`.

`-Wno-div-by-zero`

Do not warn about compile-time integer division by zero. Floating-point division by zero is not warned about, as it can be a legitimate way of obtaining infinities and NaNs.

`-Wsystem-headers`

Print warning messages for constructs found in system header files. Warnings from system headers are normally suppressed, on the assumption that they usually do not indicate real problems and would only make the compiler output harder to read. Using this command-line option tells GCC to emit warnings from system headers as if they occurred in user code. However, note that using `-Wall` in conjunction with this option does *not* warn about unknown pragmas in system headers—for that, `-Wunknown-pragmas` must also be used.

`-Wtautological-compare`

Warn if a self-comparison always evaluates to true or false. This warning detects various mistakes such as:

```
int i = 1;
...
if (i > i) { ... }
```

This warning also warns about bitwise comparisons that always evaluate to true or false, for instance:

```
if ((a & 16) == 10) { ... }
```

will always be false.

This warning is enabled by `-Wall`.

`-Wtrailing-whitespace`

`-Wtrailing-whitespace=kind`

Warn about trailing whitespace at the end of lines, including inside of comments, but excluding trailing whitespace in raw string literals. `-Wtrailing-whitespace` is equivalent to `-Wtrailing-whitespace=blanks` and warns just about trailing space and horizontal tab characters. `-Wtrailing-whitespace=any` warns about those or trailing form feed or vertical tab characters. `-Wno-trailing-whitespace` or `-Wtrailing-whitespace=none` disables the warning, which is the default. This is a coding style warning.

`-Wleading-whitespace=kind`

Warn about style issues in leading whitespace, but not about the amount of indentation. Some projects use coding styles where only spaces are used for indentation, others use only tabs, others use zero or more tabs (for multiples of `-ftabstop=n`) followed by zero or fewer than n spaces. No warning is emitted on lines which contain solely whitespace (although `-Wtrailing-whitespace=` warning might be emitted), no warnings are emitted inside of raw string literals. Warnings are also emitted for leading whitespace inside of multi-line comments. `-Wleading-whitespace=spaces` warns about leading whitespace other than spaces for projects which want to indent just by spaces. `-Wleading-whitespace=tabs` warns about leading whitespace other than horizontal tabs for projects which want to indent just by horizontal tabs. `-Wleading-whitespace=blanks` warns about leading whitespace other than spaces and horizontal tabs, or about horizontal tab after a space in the leading whitespace, or about n or more consecutive spaces in leading whitespace (where n is argument of `-ftabstop=n`, 8 by default). `-Wleading-whitespace=none` disables the warning, which is the default. This is a coding style warning.

`-Wtrampolines`

Warn about trampolines generated for pointers to nested functions. A trampoline is a small piece of data or code that is created at run time on the stack when the address of a nested function is taken, and is used to call the nested function indirectly. For some targets, it is made up of data only and thus requires no special treatment. But, for most targets, it is made up of code and thus requires the stack to be made executable in order for the program to work properly.

`-Wfloat-equal`

Warn if floating-point values are used in equality comparisons.

The idea behind this is that sometimes it is convenient (for the programmer) to consider floating-point values as approximations to infinitely precise real numbers. If you are doing this, then you need to compute (by analyzing the code, or in some other way) the maximum or likely maximum error that the computation introduces, and allow for it when performing comparisons (and when producing output, but that's a different problem). In particular, instead

of testing for equality, you should check to see whether the two values have ranges that overlap; and this is done with the relational operators, so equality comparisons are probably mistaken.

-Wtraditional (C and Objective-C only)

Warn about certain constructs that behave differently in traditional and ISO C. Also warn about ISO C constructs that have no traditional C equivalent, and/or problematic constructs that should be avoided.

- Macro parameters that appear within string literals in the macro body. In traditional C macro replacement takes place within string literals, but in ISO C it does not.
- In traditional C, some preprocessor directives did not exist. Traditional preprocessors only considered a line to be a directive if the '#' appeared in column 1 on the line. Therefore **-Wtraditional** warns about directives that traditional C understands but ignores because the '#' does not appear as the first character on the line. It also suggests you hide directives like **#pragma** not understood by traditional C by indenting them. Some traditional implementations do not recognize **#elif**, so this option suggests avoiding it altogether.
- A function-like macro that appears without arguments.
- The unary plus operator.
- The 'U' integer constant suffix, or the 'F' or 'L' floating-point constant suffixes. (Traditional C does support the 'L' suffix on integer constants.) Note, these suffixes appear in macros defined in the system headers of most modern systems, e.g. the '_MIN'/'_MAX' macros in `<limits.h>`. Use of these macros in user code might normally lead to spurious warnings, however GCC's integrated preprocessor has enough context to avoid warning in these cases.
- A function declared external in one block and then used after the end of the block.
- A **switch** statement has an operand of type **long**.
- A non-**static** function declaration follows a **static** one. This construct is not accepted by some traditional C compilers.
- The ISO type of an integer constant has a different width or signedness from its traditional type. This warning is only issued if the base of the constant is ten. I.e. hexadecimal or octal values, which typically represent bit patterns, are not warned about.
- Usage of ISO string concatenation is detected.
- Initialization of automatic aggregates.
- Identifier conflicts with labels. Traditional C lacks a separate namespace for labels.
- Initialization of unions. If the initializer is zero, the warning is omitted. This is done under the assumption that the zero initializer in user code appears conditioned on e.g. `__STDC__` to avoid missing initializer warnings and relies on default initialization to zero in the traditional C case.

- Conversions by prototypes between fixed/floating-point values and vice versa. The absence of these prototypes when compiling with traditional C causes serious problems. This is a subset of the possible conversion warnings; for the full set use `-Wtraditional-conversion`.
- Use of ISO C style function definitions. This warning intentionally is *not* issued for prototype declarations or variadic functions because these ISO C features appear in your code when using libiberty's traditional C compatibility macros, `PARAMS` and `VPARAMS`. This warning is also bypassed for nested functions because that feature is already a GCC extension and thus not relevant to traditional C compatibility.

-Wtraditional-conversion (C and Objective-C only)

Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype. This includes conversions of fixed point to floating and vice versa, and conversions changing the width or signedness of a fixed-point argument except when the same as the default promotion.

-Wdeclaration-after-statement (C and Objective-C only)

Warn when a declaration is found after a statement in a block. This construct, known from C++, was introduced with ISO C99 and is by default allowed in GCC. It is not supported by ISO C90. See Section 6.12.15 [Mixed Labels and Declarations], page 791.

This warning is upgraded to an error by `-pedantic-errors`.

-Wshadow Warn whenever a local variable or type declaration shadows another variable, parameter, type, class member (in C++), or instance variable (in Objective-C) or whenever a built-in function is shadowed. Note that in C++, the compiler warns if a local variable shadows an explicit typedef, but not if it shadows a struct/class/enum. If this warning is enabled, it includes also all instances of local shadowing. This means that `-Wno-shadow=local` and `-Wno-shadow=compatible-local` are ignored when `-Wshadow` is used. Same as `-Wshadow=global`.

-Wno-shadow-ivar (Objective-C only)

Do not warn whenever a local variable shadows an instance variable in an Objective-C method.

-Wshadow=global

Warn for any shadowing. Same as `-Wshadow`.

-Wshadow=local

Warn when a local variable shadows another local variable or parameter.

-Wshadow=compatible-local

Warn when a local variable shadows another local variable or parameter whose type is compatible with that of the shadowing variable. In C++, type compatibility here means the type of the shadowing variable can be converted to that of the shadowed variable. The creation of this flag (in addition to `-Wshadow=local`) is based on the idea that when a local variable shadows another one of incom-

patible type, it is most likely intentional, not a bug or typo, as shown in the following example:

```
for (SomeIterator i = SomeObj.begin(); i != SomeObj.end(); ++i)
{
    for (int i = 0; i < N; ++i)
    {
        ...
    }
    ...
}
```

Since the two variable `i` in the example above have incompatible types, enabling only `-Wshadow=compatible-local` does not emit a warning. Because their types are incompatible, if a programmer accidentally uses one in place of the other, type checking is expected to catch that and emit an error or warning. Use of this flag instead of `-Wshadow=local` can possibly reduce the number of warnings triggered by intentional shadowing. Note that this also means that shadowing `const char *i` by `char *i` does not emit a warning.

This warning is also enabled by `-Wshadow=local`.

`-Wlarger-than=byte-size`

Warn whenever an object is defined whose size exceeds *byte-size*. `-Wlarger-than='PTRDIFF_MAX'` is enabled by default. Warnings controlled by the option can be disabled either by specifying *byte-size* of `'SIZE_MAX'` or more or by `-Wno-larger-than`.

Also warn for calls to bounded functions such as `memchr` or `strnlen` that specify a bound greater than the largest possible object, which is `'PTRDIFF_MAX'` bytes by default. These warnings can only be disabled by `-Wno-larger-than`.

`-Wno-larger-than`

Disable `-Wlarger-than=` warnings. The option is equivalent to `-Wlarger-than='SIZE_MAX'` or `larger`.

`-Wframe-larger-than=byte-size`

Warn if the size of a function frame exceeds *byte-size*. The computation done to determine the stack frame size is approximate and not conservative. The actual requirements may be somewhat greater than *byte-size* even if you do not get a warning. In addition, any space allocated via `alloca`, variable-length arrays, or related constructs is not included by the compiler when determining whether or not to issue a warning. `-Wframe-larger-than='PTRDIFF_MAX'` is enabled by default. Warnings controlled by the option can be disabled either by specifying *byte-size* of `'SIZE_MAX'` or more or by `-Wno-frame-larger-than`.

`-Wno-frame-larger-than`

Disable `-Wframe-larger-than=` warnings. The option is equivalent to `-Wframe-larger-than='SIZE_MAX'` or `larger`.

`-Wfree-nonheap-object`

Warn when attempting to deallocate an object that was either not allocated on the heap, or by using a pointer that was not returned from a prior call to the corresponding allocation function. For example, because the call to `stpcpy`

returns a pointer to the terminating nul character and not to the beginning of the object, the call to `free` below is diagnosed.

```
void f (char *p)
{
    p = strcpy (p, "abc");
    // ...
    free (p);    // warning
}
```

`-Wfree-nonheap-object` is included in `-Wall`.

`-Wstack-usage=byte-size`

Warn if the stack usage of a function might exceed *byte-size*. The computation done to determine the stack usage is conservative. Any space allocated via `alloca`, variable-length arrays, or related constructs is included by the compiler when determining whether or not to issue a warning.

The message is in keeping with the output of `-fstack-usage`.

- If the stack usage is fully static but exceeds the specified amount, it's:

```
warning: stack usage is 1120 bytes
```

- If the stack usage is (partly) dynamic but bounded, it's:

```
warning: stack usage might be 1648 bytes
```

- If the stack usage is (partly) dynamic and not bounded, it's:

```
warning: stack usage might be unbounded
```

`-Wstack-usage='PTRDIFF_MAX'` is enabled by default. Warnings controlled by the option can be disabled either by specifying *byte-size* of `'SIZE_MAX'` or more or by `-Wno-stack-usage`.

`-Wno-stack-usage`

Disable `-Wstack-usage=` warnings. The option is equivalent to `-Wstack-usage='SIZE_MAX'` or larger.

`-Wunsafe-loop-optimizations`

Warn if the loop cannot be optimized because the compiler cannot assume anything on the bounds of the loop indices. With `-funsafe-loop-optimizations` warn if the compiler makes such assumptions.

`-Wno-pedantic-ms-format` (MinGW targets only)

When used in combination with `-Wformat` and `-pedantic` without GNU extensions, this option disables the warnings about non-ISO `printf` / `scanf` format width specifiers `I32`, `I64`, and `I` used on Windows targets, which depend on the MS runtime.

`-Wpointer-arith`

Warn about anything that depends on the “size of” a function type or of `void`. GNU C assigns these types a size of 1, for convenience in calculations with `void *` pointers and pointers to functions. In C++, warn also when an arithmetic operation involves `NULL`. This warning is also enabled by `-Wpedantic`.

This warning is upgraded to an error by `-pedantic-errors`.

-Wno-pointer-compare

Do not warn if a pointer is compared with a zero character constant. This usually means that the pointer was meant to be dereferenced. For example:

```
const char *p = foo ();
if (p == '\0')
    return 42;
```

Note that the code above is invalid in C++11.

This warning is enabled by default.

-Wno-tsan

Disable warnings about unsupported features in ThreadSanitizer.

ThreadSanitizer does not support `std::atomic_thread_fence` and can report false positives.

-Wtype-limits

Warn if a comparison is always true or always false due to the limited range of the data type, but do not warn for constant expressions. For example, warn if an unsigned variable is compared against zero with `<` or `>=`. This warning is also enabled by `-Wextra`.

-Wabsolute-value (C and Objective-C only)

Warn for calls to standard functions that compute the absolute value of an argument when a more appropriate standard function is available. For example, calling `abs(3.14)` triggers the warning because the appropriate function to call to compute the absolute value of a double argument is `fabs`. The option also triggers warnings when the argument in a call to such a function has an unsigned type. This warning can be suppressed with an explicit type cast and it is also enabled by `-Wextra`.

-Wcomment**-Wcomments**

Warn whenever a comment-start sequence `/*` appears in a `/*` comment, or whenever a backslash-newline appears in a `/**` comment. This warning is enabled by `-Wall`.

-Wtrigraphs

Warn if any trigraphs are encountered that might change the meaning of the program. Trigraphs within comments are not warned about, except those that would form escaped newlines.

This option is implied by `-Wall`. If `-Wall` is not given, this option is still enabled unless trigraphs are enabled. To get trigraph conversion without warnings, but get the other `-Wall` warnings, use `'-trigraphs -Wall -Wno-trigraphs'`.

-Wundef

Warn if an undefined identifier is evaluated in an `#if` directive. Such identifiers are replaced with zero.

-Wexpansion-to-defined

Warn whenever `defined` is encountered in the expansion of a macro (including the case where the macro is expanded by an `#if` directive). Such usage is not portable. This warning is also enabled by `-Wpedantic` and `-Wextra`.

-Wunused-macros

Warn about macros defined in the main file that are unused. A macro is *used* if it is expanded or tested for existence at least once. The preprocessor also warns if the macro has not been used at the time it is redefined or undefined. Built-in macros, macros defined on the command line, and macros defined in include files are not warned about.

Note: If a macro is actually used, but only used in skipped conditional blocks, then the preprocessor reports it as unused. To avoid the warning in such a case, you might improve the scope of the macro's definition by, for example, moving it into the first skipped block. Alternatively, you could provide a dummy use with something like:

```
#if defined the_macro_causing_the_warning
#endif
```

-Wno-endif-labels

Do not warn whenever an **#else** or an **#endif** are followed by text. This sometimes happens in older programs with code of the form

```
#if F00
...
#else F00
...
#endif F00
```

The second and third **F00** should be in comments. This warning is on by default.

-Wbad-function-cast (C and Objective-C only)

Warn when a function call is cast to a non-matching type. For example, warn if a call to a function returning an integer type is cast to a pointer type.

-Wc90-c99-compat (C and Objective-C only)

Warn about features not present in ISO C90, but present in ISO C99. For instance, warn about use of variable length arrays, **long long** type, **bool** type, compound literals, designated initializers, and so on. This option is independent of the standards mode. Warnings are disabled in the expression that follows **__extension__**.

-Wc99-c11-compat (C and Objective-C only)

Warn about features not present in ISO C99, but present in ISO C11. For instance, warn about use of anonymous structures and unions, **_Atomic** type qualifier, **_Thread_local** storage-class specifier, **_Alignas** specifier, **alignof** operator, **_Generic** keyword, and so on. This option is independent of the standards mode. Warnings are disabled in the expression that follows **__extension__**.

-Wc11-c23-compat (C and Objective-C only)**-Wc11-c2x-compat** (C and Objective-C only)

Warn about features not present in ISO C11, but present in ISO C23. For instance, warn about omitting the string in **_Static_assert**, use of **'[[]]'** syntax for attributes, use of decimal floating-point types, and so on. This option is independent of the standards mode. Warnings are disabled in the expression that follows **__extension__**. The name **-Wc11-c2x-compat** is deprecated.

When not compiling in C23 mode, these warnings are upgraded to errors by **-pedantic-errors**.

`-Wc23-c2y-compat` (C and Objective-C only)

`-Wc23-c2y-compat` (C and Objective-C only)

Warn about features not present in ISO C23, but present in ISO C2Y. For instance, warn about `_Generic` selecting with a type name instead of an expression. This option is independent of the standards mode. Warnings are disabled in the expression that follows `__extension__`.

When not compiling in C2Y mode, these warnings are upgraded to errors by `-pedantic-errors`.

`-Wc++-compat` (C and Objective-C only)

Warn about ISO C constructs that are outside of the common subset of ISO C and ISO C++, e.g. request for implicit conversion from `void *` to a pointer to non-void type.

`-Wc++11-compat` (C++ and Objective-C++ only)

Warn about C++ constructs whose meaning differs between ISO C++ 1998 and ISO C++ 2011, e.g., identifiers in ISO C++ 1998 that are keywords in ISO C++ 2011. This warning turns on `-Wnarrowing` and is enabled by `-Wall`.

`-Wc++14-compat` (C++ and Objective-C++ only)

Warn about C++ constructs whose meaning differs between ISO C++ 2011 and ISO C++ 2014. This warning is enabled by `-Wall`.

`-Wc++17-compat` (C++ and Objective-C++ only)

Warn about C++ constructs whose meaning differs between ISO C++ 2014 and ISO C++ 2017. This warning is enabled by `-Wall`.

`-Wc++20-compat` (C++ and Objective-C++ only)

Warn about C++ constructs whose meaning differs between ISO C++ 2017 and ISO C++ 2020. This warning is enabled by `-Wall`.

`-Wc++26-compat` (C++ and Objective-C++ only)

Warn about C++ constructs whose meaning differs between ISO C++ 2023 and upcoming ISO C++ 2026. This warning is enabled by `-Wall`.

`-Wno-c++11-extensions` (C++ and Objective-C++ only)

Do not warn about C++11 constructs in code being compiled using an older C++ standard. Even without this option, some C++11 constructs will only be diagnosed if `-Wpedantic` is used.

`-Wno-c++14-extensions` (C++ and Objective-C++ only)

Do not warn about C++14 constructs in code being compiled using an older C++ standard. Even without this option, some C++14 constructs will only be diagnosed if `-Wpedantic` is used.

`-Wno-c++17-extensions` (C++ and Objective-C++ only)

Do not warn about C++17 constructs in code being compiled using an older C++ standard. Even without this option, some C++17 constructs will only be diagnosed if `-Wpedantic` is used.

-Wno-c++20-extensions (C++ and Objective-C++ only)

Do not warn about C++20 constructs in code being compiled using an older C++ standard. Even without this option, some C++20 constructs will only be diagnosed if **-Wpedantic** is used.

-Wno-c++23-extensions (C++ and Objective-C++ only)

Do not warn about C++23 constructs in code being compiled using an older C++ standard. Even without this option, some C++23 constructs will only be diagnosed if **-Wpedantic** is used.

-Wno-c++26-extensions (C++ and Objective-C++ only)

Do not warn about C++26 constructs in code being compiled using an older C++ standard. Even without this option, some C++26 constructs will only be diagnosed if **-Wpedantic** is used.

-Wcast-qual

Warn whenever a pointer is cast so as to remove a type qualifier from the target type. For example, warn if a **const char *** is cast to an ordinary **char ***.

Also warn when making a cast that introduces a type qualifier in an unsafe way. For example, casting **char **** to **const char **** is unsafe, as in this example:

```
/* p is char ** value. */
const char **q = (const char **) p;
/* Assignment of readonly string to const char * is OK. */
*q = "string";
/* Now char** pointer points to read-only memory. */
**p = 'b';
```

-Wcast-align

Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a **char *** is cast to an **int *** on machines where integers can only be accessed at two- or four-byte boundaries.

Warn whenever a pointer is cast such that the required alignment of the target is increased. For example, warn if a **char *** is cast to an **int *** regardless of the target machine.

-Wcast-function-type

Warn when a function pointer is cast to an incompatible function pointer. In a cast involving function types with a variable argument list only the types of initial arguments that are provided are considered. Any parameter of pointer-type matches any other pointer-type. Any benign differences in integral types are ignored, like **int** vs. **long** on ILP32 targets. Likewise type qualifiers are ignored. The function type **void (*)** (**void**) is special and matches everything, which can be used to suppress this warning. In a cast involving pointer to member types this warning warns whenever the type cast is changing the pointer to member type. This warning is enabled by **-Wextra**.

-Wcast-user-defined

Warn when a cast to reference type does not involve a user-defined conversion that the programmer might expect to be called.

```
struct A { operator const int&(); } a;
auto r = (int&)a; // warning
```

This warning is enabled by default.

-Wwrite-strings

When compiling C, give string constants the type `const char[length]` so that copying the address of one into a non-`const char *` pointer produces a warning. These warnings help you find at compile time code that can try to write into a string constant, but only if you have been very careful about using `const` in declarations and prototypes. Otherwise, it is just a nuisance. This is why we did not make `-Wall` request these warnings.

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

This warning is upgraded to an error by `-pedantic-errors` in C++11 mode or later.

-Wclobbered

Warn for variables that might be changed by `longjmp` or `vfork`. This warning is also enabled by `-Wextra`.

-Wno-complain-wrong-lang

By default, language front ends complain when a command-line option is valid, but not applicable to that front end. This may be disabled with `-Wno-complain-wrong-lang`, which is mostly useful when invoking a single compiler driver for multiple source files written in different languages, for example:

```
$ g++ -fno-rtti a.cc b.f90
```

The driver `g++` invokes the C++ front end to compile `a.cc` and the Fortran front end to compile `b.f90`. The latter front end diagnoses ‘f951: Warning: command-line option ‘-fno-rtti’ is valid for C++/D/ObjC++ but not for Fortran’, which may be disabled with `-Wno-complain-wrong-lang`.

This option can also be used to disable warnings like ‘cc1plus: note: CTF debug info requested, but not supported for ‘GNU C++20’ frontend’ produced by `-gctf` or `-gsctf` for unsupported languages.

-Wcompare-distinct-pointer-types (C and Objective-C only)

Warn if pointers of distinct types are compared without a cast. This warning is enabled by default.

-Wconversion

Warn for implicit conversions that may alter a value. This includes conversions between real and integer, like `abs(x)` when `x` is `double`; conversions between signed and unsigned, like `unsigned ui = -1`; and conversions to smaller types, like `sqrtf(M_PI)`. Do not warn for explicit casts like `abs((int)x)` and `ui = (unsigned)-1`, or if the value is not changed by the conversion like in `abs(2.0)`. Warnings about conversions between signed and unsigned integers can be disabled by using `-Wno-sign-conversion`.

For C++, also warn for confusing overload resolution for user-defined conversions; and conversions that never use a type conversion operator: conversions to `void`, the same type, a base class or a reference to them. Warnings about

conversions between signed and unsigned integers are disabled by default in C++ unless `-Wsign-conversion` is explicitly enabled.

Warnings about conversion from arithmetic on a small type back to that type are only given with `-Warith-conversion`.

`-Wdangling-else`

Warn about constructions where there may be confusion to which `if` statement an `else` branch belongs. Here is an example of such a case:

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

In C/C++, every `else` branch belongs to the innermost possible `if` statement, which in this example is `if (b)`. This is often not what the programmer expected, as illustrated in the above example by indentation the programmer chose. When there is the potential for this confusion, GCC issues a warning when this flag is specified. To eliminate the warning, add explicit braces around the innermost `if` statement so there is no way the `else` can belong to the enclosing `if`. The resulting code looks like this:

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

This warning is enabled by `-Wparentheses`.

`-Wdangling-pointer`

`-Wdangling-pointer=n`

Warn about uses of pointers (or C++ references) to objects with automatic storage duration after their lifetime has ended. This includes local variables declared in nested blocks, compound literals and other unnamed temporary objects. In addition, warn about storing the address of such objects in escaped pointers. The warning is enabled at all optimization levels but may yield different results with optimization than without.

`-Wdangling-pointer=1`

At level 1, the warning diagnoses only unconditional uses of dangling pointers.

`-Wdangling-pointer=2`

At level 2, in addition to unconditional uses the warning also diagnoses conditional uses of dangling pointers.

The short form `-Wdangling-pointer` is equivalent to `-Wdangling-pointer=2`, while `-Wno-dangling-pointer` and `-Wdangling-pointer=0` have the same effect of disabling the warnings. `-Wdangling-pointer=2` is included in `-Wall`.

This example triggers the warning at level 1; the address of the unnamed temporary is unconditionally referenced outside of its scope.

```
char f (char c1, char c2, char c3)
{
    char *p;
    {
        p = (char[]) { c1, c2, c3 };
    }
    // warning: using dangling pointer 'p' to an unnamed temporary
    return *p;
}
```

In the following function the store of the address of the local variable `x` in the escaped pointer `*p` triggers the warning at level 1.

```
void g (int **p)
{
    int x = 7;
    // warning: storing the address of local variable 'x' in '*p'
    *p = &x;
}
```

In this example, the array `a` is out of scope when the pointer `s` is used. Since the code that sets `s` is conditional, the warning triggers at level 2.

```
extern void frob (const char *);
void h (char *s)
{
    if (!s)
    {
        char a[12] = "tmpname";
        s = a;
    }
    // warning: dangling pointer 's' to 'a' may be used
    frob (s);
}
```

`-Wdate-time`

Warn when macros `__TIME__`, `__DATE__` or `__TIMESTAMP__` are encountered as they might prevent bitwise-identical reproducible compilations.

`-Wempty-body`

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

`-Wno-endif-labels`

Do not warn about stray tokens after `#else` and `#endif`.

`-Wenum-compare`

Warn about a comparison between values of different enumerated types. In C++, enumerated type mismatches in conditional expressions are also diagnosed and the warning is enabled by default. In C, this warning is enabled by `-Wall`.

`-Wenum-conversion`

Warn when a value of enumerated type is implicitly converted to a different enumerated type. This warning is enabled by `-Wextra` in C.

-Wenum-int-mismatch (C and Objective-C only)

Warn about mismatches between an enumerated type and an integer type in declarations. For example:

```
enum E { l = -1, z = 0, g = 1 };
int foo(void);
enum E foo(void);
```

In C, an enumerated type is compatible with `char`, a signed integer type, or an unsigned integer type. However, since the choice of the underlying type of an enumerated type is implementation-defined, such mismatches may cause portability issues. In C++, such mismatches are an error. In C, this warning is enabled by `-Wall` and `-Wc++-compat`.

-Wjump-misses-init (C, Objective-C only)

Warn if a `goto` statement or a `switch` statement jumps forward across the initialization of a variable, or jumps backward to a label after the variable has been initialized. This only warns about variables that are initialized when they are declared. This warning is only supported for C and Objective-C; in C++ this sort of branch is an error in any case.

`-Wjump-misses-init` is included in `-Wc++-compat`. It can be disabled with the `-Wno-jump-misses-init` option.

-Wsign-compare

Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned. In C++, this warning is also enabled by `-Wall`. In C, it is also enabled by `-Wextra`.

-Wsign-conversion

Warn for implicit conversions that may change the sign of an integer value, like assigning a signed integer expression to an unsigned integer variable. An explicit cast silences the warning. In C, this option is enabled also by `-Wconversion`.

-Wflex-array-member-not-at-end (C and C++ only)

Warn when a structure containing a C99 flexible array member as the last field is not at the end of another structure. This warning warns e.g. about

```
struct flex { int length; char data[]; };
struct mid_flex { int m; struct flex flex_data; int n; };
```

-Wfloat-conversion

Warn for implicit conversions that reduce the precision of a real value. This includes conversions from real to integer, and from higher precision real to lower precision real values. This option is also enabled by `-Wconversion`.

-Wno-scalar-storage-order

Do not warn on suspicious constructs involving reverse scalar storage order.

-Wsizeof-array-div

Warn about divisions of two `sizeof` operators when the first one is applied to an array and the divisor does not equal the size of the array element. In such a case, the computation will not yield the number of elements in the array, which is likely what the user intended. This warning warns e.g. about

```
int fn ()
```

```

{
    int arr[10];
    return sizeof (arr) / sizeof (short);
}

```

This warning is enabled by `-Wall`.

`-Wsizeof-pointer-div`

Warn for suspicious divisions of two `sizeof` expressions that divide the pointer size by the element size, which is the usual way to compute the array size but won't work out correctly with pointers. This warning warns e.g. about `sizeof (ptr) / sizeof (ptr[0])` if `ptr` is not an array, but a pointer. This warning is enabled by `-Wall`.

`-Wsizeof-pointer-memaccess`

Warn for suspicious length parameters to certain string and memory built-in functions if the argument uses `sizeof`. This warning triggers for example for `memset (ptr, 0, sizeof (ptr))`; if `ptr` is not an array, but a pointer, and suggests a possible fix, or about `memcpy (&foo, ptr, sizeof (&foo))`; `-Wsizeof-pointer-memaccess` also warns about calls to bounded string copy functions like `strncat` or `strncpy` that specify as the bound a `sizeof` expression of the source array. For example, in the following function the call to `strncat` specifies the size of the source string as the bound. That is almost certainly a mistake and so the call is diagnosed.

```

void make_file (const char *name)
{
    char path[PATH_MAX];
    strncpy (path, name, sizeof path - 1);
    strncat (path, ".text", sizeof ".text");
    ...
}

```

The `-Wsizeof-pointer-memaccess` option is enabled by `-Wall`.

`-Wno-sizeof-array-argument`

Do not warn when the `sizeof` operator is applied to a parameter that is declared as an array in a function definition. This warning is enabled by default for C and C++ programs.

`-Wmemset-elt-size`

Warn for suspicious calls to the `memset` built-in function, if the first argument references an array, and the third argument is a number equal to the number of elements, but not equal to the size of the array in memory. This indicates that the user has omitted a multiplication by the element size. This warning is enabled by `-Wall`.

`-Wmemset-transposed-args`

Warn for suspicious calls to the `memset` built-in function where the second argument is not zero and the third argument is zero. For example, the call `memset (buf, sizeof buf, 0)` is diagnosed because `memset (buf, 0, sizeof buf)` was meant instead. The diagnostic is only emitted if the third argument is a literal zero. Otherwise, if it is an expression that is folded to zero, or a cast of zero to some type, it is far less likely that the arguments have been mistakenly transposed and no warning is emitted. This warning is enabled by `-Wall`.

-Waddress

Warn about suspicious uses of address expressions. These include comparing the address of a function or a declared object to the null pointer constant such as in

```
void f (void);
void g (void)
{
    if (!f)    // warning: expression evaluates to false
        abort ();
}
```

comparisons of a pointer to a string literal, such as in

```
void f (const char *x)
{
    if (x == "abc")    // warning: expression evaluates to false
        puts ("equal");
}
```

and tests of the results of pointer addition or subtraction for equality to null, such as in

```
void f (const int *p, int i)
{
    return p + i == NULL;
}
```

Such uses typically indicate a programmer error: the address of most functions and objects necessarily evaluates to true (the exception are weak symbols), so their use in a conditional might indicate missing parentheses in a function call or a missing dereference in an array expression. The subset of the warning for object pointers can be suppressed by casting the pointer operand to an integer type such as `intptr_t` or `uintptr_t`. Comparisons against string literals result in unspecified behavior and are not portable, and suggest the intent was to call `strcmp`. The warning is suppressed if the suspicious expression is the result of macro expansion. **-Waddress** is enabled by **-Wall**.

-Wno-address-of-packed-member

Do not warn when the address of packed member of struct or union is taken, which usually results in an unaligned pointer value. This is enabled by default.

-Wlogical-op

Warn about suspicious uses of logical operators in expressions. This includes using logical operators in contexts where a bitwise operator is likely to be expected. Also warns when the operands of a logical operator are the same:

```
extern int a;
if (a < 0 && a < 0) { ... }
```

-Wconstant-logical-operand

Warn about another case of suspicious uses of logical operators in expressions, when neither operand of a logical operator is boolean and one of the operands is (or folds into) a constant other than 0 or 1, or enumerator with value 1 if the enumeral type contains enumerators with values other than 0 or 1. In such case the warning will suggest using corresponding bitwise operator.

```
extern int a;
if (a && 64) { ... }
```

If the warning is a false positive, one can clarify the code by using e.g. `a && (64 != 0)` instead.

-Wlogical-not-parentheses

Warn about logical not used on the left hand side operand of a comparison. This option does not warn if the right operand is considered to be a boolean expression. Its purpose is to detect suspicious code like the following:

```
int a;
...
if (!a > 1) { ... }
```

It is possible to suppress the warning by wrapping the LHS into parentheses:

```
if ((!a) > 1) { ... }
```

This warning is enabled by `-Wall`.

-Waggregate-return

Warn if any functions that return structures or unions are defined or called. (In languages where you can return an array, this also elicits a warning.)

-Wno-aggressive-loop-optimizations

Do not warn if the compiler detects undefined behavior in a loop with a constant number of iterations. `-Waggressive-loop-optimizations` is enabled by default.

-Wno-attributes

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

Warnings about ill-formed uses of standard attributes are upgraded to errors by `-pedantic-errors`.

Additionally, using `-Wno-attributes=`, it is possible to suppress warnings about unknown scoped attributes (in C++11 and C23). For example, `-Wno-attributes=vendor::attr` disables warning about the following declaration:

```
[[vendor::attr]] void f();
```

It is also possible to disable warning about all attributes in a namespace using `-Wno-attributes=vendor::` which prevents warning about both of these declarations:

```
[[vendor::safe]] void f();
[[vendor::unsafe]] void f2();
```

Note that `-Wno-attributes=` does not imply `-Wno-attributes`.

-Wno-builtin-declaration-mismatch

Warn if a built-in function is declared with an incompatible signature or as a non-function, or when a built-in function declared with a type that does not include a prototype is called with arguments whose promoted types do not match those expected by the function. When `-Wextra` is specified, also warn when a built-in function that takes arguments is declared without a prototype. The `-Wbuiltin-declaration-mismatch` warning is enabled by default. To

avoid the warning include the appropriate header to bring the prototypes of built-in functions into scope.

For example, the call to `memset` below is diagnosed by the warning because the function expects a value of type `size_t` as its argument but the type of `32` is `int`. With `-Wextra`, the declaration of the function is diagnosed as well.

```
extern void* memset ();
void f (void *d)
{
    memset (d, '\0', 32);
}
```

`-Wno-builtin-macro-redefined`

Do not warn if certain built-in macros are redefined. This suppresses warnings for redefinition of `__TIMESTAMP__`, `__TIME__`, `__DATE__`, `__FILE__`, and `__BASE_FILE__`.

`-Wkeyword-macro`

Warn if a keyword is defined as a macro or undefined. For C++ identifiers with special meaning or standard attribute identifiers are diagnosed as well. This warning is enabled by default for C++26 if `-Wpedantic` and emits a `pedwarn` in that case.

`-Wfree-labels` (C and Objective-C only)

Warn if a label is applied to a non-statement, or occurs at the end of a compound statement. Such labels are allowed by C23 and later dialects of C, and are available as a GCC extension in all other dialects.

This warning is also enabled by `-Wc11-c23-compat`. It is turned into an error if building for a C version before C23 by `-pedantic-errors`.

`-Wheader-guard`

Warn if a valid preprocessor header multiple inclusion guard has a `#define` directive right after `#ifndef` or `#if !defined` directive for the multiple inclusion guard, which defines a different macro from the guard macro with a similar name, the actual multiple inclusion guard macro isn't defined at the corresponding `#ifndef` directive at the end of the header, and the `#define` directive defines an object-like macro with empty definition. In such case, it often is just a misspelled guard name, either in the `#ifndef` or `#if !defined` directive or in the subsequent `#define` directive. This warning is enabled by `-Wall`.

`-Wstrict-prototypes` (C and Objective-C only)

Warn if a function is declared or defined without specifying the argument types. (An old-style function definition is permitted without a warning if preceded by a declaration that specifies the argument types.)

`-Wold-style-declaration` (C and Objective-C only)

Warn for obsolescent usages, according to the C Standard, in a declaration. For example, warn if storage-class specifiers like `static` are not the first things in a declaration. This warning is also enabled by `-Wextra`.

`-Wold-style-definition` (C and Objective-C only)

Warn if an old-style function definition is used. A warning is given even if there is a previous prototype. A definition using `'()`' is not considered an old-style

definition in C23 mode, because it is equivalent to `'(void)'` in that case, but is considered an old-style definition for older standards.

-Wmultiple-parameter-fwd-decl-lists (C and Objective-C only)

Warn if more than one list of forward declarations of parameters appears in a function prototype. This warning is also enabled by **-Wextra**.

-Wdeprecated-non-prototype (C and Objective-C only)

Warn if a function declared with an empty parameter list `'()'` is called with one or more arguments, or if a function definition with one or more parameters is encountered after such a declaration. Both cases are errors in C23 and later dialects of C.

This warning is also enabled by **-Wc11-c23-compat**.

-Wmissing-parameter-name (C and Objective-C only)

Warn if a function definition omits a parameter name, specifying only its type. This can be used to document that a parameter is unused in the definition. It is part of C23 and later dialects of C, and available as a GCC extension in all other dialects.

This warning is also enabled by **-Wc11-c23-compat**. It is turned into an error if building for a C version before C23 by **-pedantic-errors**.

-Wmissing-parameter-type (C and Objective-C only)

A function parameter is declared without a type specifier in K&R-style functions:

```
void foo(bar) { }
```

This warning is also enabled by **-Wextra**.

-Wno-declaration-missing-parameter-type (C and Objective-C only)

Do not warn if a function declaration contains a parameter name without a type. Such function declarations do not provide a function prototype and prevent most type checking in function calls.

This warning is enabled by default. In C99 and later dialects of C, it is treated as an error. The error can be downgraded to a warning using **-fpermissive** (along with certain other errors), or for this error alone, with **-Wno-error=declaration-missing-parameter-type**.

This warning is upgraded to an error by **-pedantic-errors**.

-Wmissing-prototypes (C and Objective-C only)

Warn if a global function is defined without a previous prototype declaration. This warning is issued even if the definition itself provides a prototype. Use this option to detect global functions that do not have a matching prototype declaration in a header file. This option is not valid for C++ because all function declarations provide prototypes and a non-matching declaration declares an overload rather than conflict with an earlier declaration. Use **-Wmissing-declarations** to detect missing declarations in C++.

-Wmissing-variable-declarations (C and Objective-C only)

Warn if a global variable is defined without a previous declaration. Use this option to detect global variables that do not have a matching extern declaration in a header file.

-Wmissing-declarations

Warn if a global function is defined without a previous declaration. Do so even if the definition itself provides a prototype. Use this option to detect global functions that are not declared in header files. In C, no warnings are issued for functions with previous non-prototype declarations; use **-Wmissing-prototypes** to detect missing prototypes. In C++, no warnings are issued for function templates, or for inline functions, or for functions in anonymous namespaces.

-Wmissing-field-initializers

Warn if a structure's initializer has some fields missing. For example, the following code causes such a warning, because `x.h` is implicitly zero:

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

In C this option does not warn about designated initializers, so the following modification does not trigger a warning:

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

In C this option does not warn about the universal zero initializer `{ 0 }`:

```
struct s { int f, g, h; };
struct s x = { 0 };
```

Likewise, in C++ this option does not warn about the empty `{ }` initializer, for example:

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

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

-Wno-missing-requires

By default, the compiler warns about a concept-id appearing as a C++20 simple-requirement:

```
bool satisfied = requires { C<T> };
```

Here `'satisfied'` will be true if `'C<T>'` is a valid expression, which it is for all `T`. Presumably the user meant to write

```
bool satisfied = requires { requires C<T> };
```

so `'satisfied'` is only true if concept `'C'` is satisfied for type `'T'`.

This warning can be disabled with **-Wno-missing-requires**.

-Wno-missing-template-keyword

The member access tokens `., ->` and `::` must be followed by the **template** keyword if the parent object is dependent and the member being named is a template.

```
template <class X>
void DoStuff (X x)
{
    x.template DoSomeOtherStuff<X>(); // Good.
    x.DoMoreStuff<X>(); // Warning, x is dependent.
}
```

In rare cases it is possible to get false positives. To silence this, wrap the expression in parentheses. For example, the following is treated as a template, even where `m` and `N` are integers:

```
void NotATemplate (my_class t)
{
    int N = 5;

    bool test = t.m < N > (0); // Treated as a template.
    test = (t.m < N) > (0); // Same meaning, but not treated as a template.
}
```

This warning can be disabled with `-Wno-missing-template-keyword`.

`-Wno-multichar`

Do not warn if a multicharacter constant (`'FOOF'`) is used. Usually they indicate a typo in the user's code, as they have implementation-defined values, and should not be used in portable code.

`-Wnormalized=[none|id|nfc|nfkc]`

In ISO C and ISO C++, two identifiers are different if they are different sequences of characters. However, sometimes when characters outside the basic ASCII character set are used, you can have two different character sequences that look the same. To avoid confusion, the ISO 10646 standard sets out some *normalization rules* which when applied ensure that two sequences that look the same are turned into the same sequence. GCC can warn you if you are using identifiers that have not been normalized; this option controls that warning.

There are four levels of warning supported by GCC. The default is `-Wnormalized=nfc`, which warns about any identifier that is not in the ISO 10646 “C” normalized form, *NFC*. *NFC* is the recommended form for most uses. It is equivalent to `-Wnormalized`.

Unfortunately, there are some characters allowed in identifiers by ISO C and ISO C++ that, when turned into *NFC*, are not allowed in identifiers. That is, there's no way to use these symbols in portable ISO C or C++ and have all your identifiers in *NFC*. `-Wnormalized=id` suppresses the warning for these characters. It is hoped that future versions of the standards involved will correct this, which is why this option is not the default.

You can switch the warning off for all characters by writing `-Wnormalized=none` or `-Wno-normalized`. You should only do this if you are using some other normalization scheme (like “D”), because otherwise you can easily create bugs that are literally impossible to see.

Some characters in ISO 10646 have distinct meanings but look identical in some fonts or display methodologies, especially once formatting has been applied. For instance `\u207F`, “SUPERSCRIPT LATIN SMALL LETTER N”, displays just like a regular `n` that has been placed in a superscript. ISO 10646 defines the *NFKC* normalization scheme to convert all these into a standard form as well, and GCC warns if your code is not in *NFKC* if you use `-Wnormalized=nfkc`. This warning is comparable to warning about every identifier that contains the letter `O` because it might be confused with the digit `0`, and so is not the default,

but may be useful as a local coding convention if the programming environment cannot be fixed to display these characters distinctly.

-Wno-attribute-warning

Do not warn about usage of functions declared with `warning` attribute (see Section 6.4.1 [Common Attributes], page 595). By default, this warning is enabled. `-Wno-attribute-warning` can be used to disable the warning or `-Wno-error=attribute-warning` can be used to disable the error when compiled with `-Werror` flag.

-Wno-deprecated

Do not warn about usage of deprecated features. See Section 8.10 [Deprecated Features], page 1045.

In C++, explicitly specifying `-Wdeprecated` also enables warnings about some features that are deprecated in later language standards, specifically `-Wcomma-subscript`, `-Wvolatile`, `-Wdeprecated-enum-float-conversion`, `-Wdeprecated-enum-enum-conversion`, `-Wdeprecated-literal-operator`, and `-Wdeprecated-variadic-comma-omission`.

-Wno-deprecated-declarations

Do not warn about uses of functions, variables, or types marked as deprecated by using the `deprecated` attribute. See Section 6.4 [Attributes], page 593.

-Wno-deprecated-openmp

Do not warn about deprecated OpenMP code.

-Wno-overflow

Do not warn about compile-time overflow in constant expressions.

-Wno-odr Warn about One Definition Rule violations during link-time optimization. Enabled by default.

-Wopenacc-parallelism

Warn about potentially suboptimal choices related to OpenACC parallelism.

-Wno-openmp

Warn about suspicious OpenMP code.

-Wopenmp-simd

Warn if the vectorizer cost model overrides the OpenMP `simd` directive set by user. The `-fsimd-cost-model=unlimited` option can be used to relax the cost model.

-Woverride-init (C and Objective-C only)

Warn if an initialized field without side effects is overridden when using designated initializers (see Section 6.2.11 [Designated Initializers], page 588).

This warning is included in `-Wextra`. To get other `-Wextra` warnings without this one, use `-Wextra -Wno-override-init`.

-Wno-override-init-side-effects (C and Objective-C only)

Do not warn if an initialized field with side effects is overridden when using designated initializers (see Section 6.2.11 [Designated Initializers], page 588). This warning is enabled by default.

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

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

-Wnpacked-bitfield-compat

The 4.1, 4.2 and 4.3 series of GCC ignore the `packed` attribute on bit-fields of type `char`. This was fixed in GCC 4.4 but the change can lead to differences in the structure layout. GCC informs you when the offset of such a field has changed in GCC 4.4. For example there is no longer a 4-bit padding between field `a` and `b` in this structure:

```
struct foo
{
    char a:4;
    char b:8;
} __attribute__((packed));
```

This warning is enabled by default. Use `-Wno-packed-bitfield-compat` to disable this warning.

-Wpacked-not-aligned (C, C++, Objective-C and Objective-C++ only)

Warn if a structure field with explicitly specified alignment in a packed struct or union is misaligned. For example, a warning will be issued on `struct S`, like, `warning: alignment 1 of 'struct S' is less than 8`, in this code:

```
struct __attribute__((aligned (8))) S8 { char a[8]; };
struct __attribute__((packed)) S {
    struct S8 s8;
};
```

This warning is enabled by `-Wall`.

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

-Wredundant-decls

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

-Wrestrict

Warn when an object referenced by a `restrict`-qualified parameter (or, in C++, a `__restrict`-qualified parameter) is aliased by another argument, or when copies between such objects overlap. For example, the call to the `strcpy` function below attempts to truncate the string by replacing its initial characters

with the last four. However, because the call writes the terminating NUL into `a[4]`, the copies overlap and the call is diagnosed.

```
void foo (void)
{
    char a[] = "abcd1234";
    strcpy (a, a + 4);
    ...
}
```

The `-Wrestrict` option detects some instances of simple overlap even without optimization but works best at `-O2` and above. It is included in `-Wall`.

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

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

-Winline Warn if a function that is declared as inline cannot be inlined. Even with this option, the compiler does not warn about failures to inline functions declared in system headers.

The compiler uses a variety of heuristics to determine whether or not to inline a function. For example, the compiler takes into account the size of the function being inlined and the amount of inlining that has already been done in the current function. Therefore, seemingly insignificant changes in the source program can cause the warnings produced by `-Winline` to appear or disappear.

-Winterference-size

-Wno-interference-size

Warn about questionable uses of the C++17 `constexpr` variables `std::hardware_destructive_interference_size` and `std::hardware_constructive_interference_size`.

These variables are intended to be used for controlling class layout, to avoid false sharing in concurrent code:

```
struct independent_fields {
    alignas(std::hardware_destructive_interference_size)
    std::atomic<int> one;
    alignas(std::hardware_destructive_interference_size)
    std::atomic<int> two;
};
```

Here ‘one’ and ‘two’ are intended to be far enough apart that stores to one won’t require accesses to the other to reload the cache line.

By default, these variables are given values based on the current `-mtune` option, typically to the L1 cache line size for the particular target CPU, sometimes to a range if tuning for a generic target. So all translation units that depend on ABI compatibility for the use of these variables must be compiled with the same `-mtune` (or `-mcpu`).

If ABI stability is important, such as if the use is in a header for a library, you should probably not use the hardware interference size variables at all.

These warnings are enabled by default. If you are confident that your use of these variables does not affect ABI outside a single build of your project, you can turn off the warning with `-Wno-interference-size`.

-Wint-in-bool-context

Warn for suspicious use of integer values where boolean values are expected, such as conditional expressions (`?:`) using non-boolean integer constants in boolean context, like `if (a <= b ? 2 : 3)`. Or left shifting of signed integers in boolean context, like `for (a = 0; 1 << a; a++);`. Likewise for all kinds of multiplications regardless of the data type. This warning is enabled by `-Wall`.

-Wno-int-to-pointer-cast

Suppress warnings from casts to pointer type of an integer of a different size. In C++, casting to a pointer type of smaller size is an error. `Wint-to-pointer-cast` is enabled by default.

-Wno-pointer-to-int-cast (C and Objective-C only)

Suppress warnings from casts from a pointer to an integer type of a different size.

-Winvalid-pch

Warn if a precompiled header (see Section 3.22 [Precompiled Headers], page 555) is found in the search path but cannot be used.

-Winvalid-utf8

Warn if an invalid UTF-8 character is found. This warning is on by default for C++23 if `-finput-charset=UTF-8` is used and turned into error with `-pedantic-errors`.

-Wno-unicode

Don't diagnose invalid forms of delimited or named escape sequences which are treated as separate tokens. `Wunicode` is enabled by default.

-Wlong-long

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

This warning is upgraded to an error by `-pedantic-errors`.

-Wvariadic-macros

Warn if variadic macros are used in ISO C90 mode, or if the GNU alternate syntax is used in ISO C99 mode. This is enabled by either `-Wpedantic` or `-Wtraditional`. To inhibit the warning messages, use `-Wno-variadic-macros`.

-Wno-varargs

Do not warn upon questionable usage of the macros used to handle variable arguments like `va_start`. These warnings are enabled by default.

-Wvector-operation-performance

Warn if vector operation is not implemented via SIMD capabilities of the architecture. Mainly useful for the performance tuning. Vector operation can be implemented **piecewise**, which means that the scalar operation is performed on every vector element; **in parallel**, which means that the vector operation is implemented using scalars of wider type, which normally is more performance efficient; and **as a single scalar**, which means that vector fits into a scalar type.

-Wvla Warn if a variable-length array is used in the code. **-Wno-vla** prevents the **-Wpedantic** warning of the variable-length array.

This warning is upgraded to an error by **-pedantic-errors**.

-Wvla-larger-than=byte-size

If this option is used, the compiler warns for declarations of variable-length arrays whose size is either unbounded, or bounded by an argument that allows the array size to exceed *byte-size* bytes. This is similar to how **-Walloca-larger-than=byte-size** works, but with variable-length arrays.

Note that GCC may optimize small variable-length arrays of a known value into plain arrays, so this warning may not get triggered for such arrays.

-Wvla-larger-than=PTRDIFF_MAX is enabled by default but is typically only effective when **-ftree-*vrp*** is active (default for **-O2** and above).

See also **-Walloca-larger-than=byte-size**.

-Wno-vla-larger-than

Disable **-Wvla-larger-than=** warnings. The option is equivalent to **-Wvla-larger-than=SIZE_MAX** or larger.

-Wvla-parameter

Warn about redeclarations of functions involving arguments of Variable Length Array types of inconsistent kinds or forms, and enable the detection of out-of-bounds accesses to such parameters by warnings such as **-Warray-bounds**.

If the first function declaration uses the VLA form the bound specified in the array is assumed to be the minimum number of elements expected to be provided in calls to the function and the maximum number of elements accessed by it. Failing to provide arguments of sufficient size or accessing more than the maximum number of elements may be diagnosed.

For example, the warning triggers for the following redeclarations because the first one allows an array of any size to be passed to **f** while the second one specifies that the array argument must have at least **n** elements. In addition, calling **f** with the associated VLA bound parameter in excess of the actual VLA bound triggers a warning as well.

```
void f (int n, int[n]);
// warning: argument 2 previously declared as a VLA
void f (int, int[]);

void g (int n)
{
    if (n > 4)
        return;
    int a[n];
    // warning: access to a by f may be out of bounds
    f (sizeof a, a);
    ...
}
```

-Wvla-parameter is included in **-Wall**. The **-Warray-parameter** option triggers warnings for similar problems involving ordinary array arguments.

-Wvolatile-register-var

Warn if a register variable is declared volatile. The volatile modifier does not inhibit all optimizations that may eliminate reads and/or writes to register variables. This warning is enabled by **-Wall**.

-Wno-xor-used-as-pow (C, C++, Objective-C and Objective-C++ only)

Disable warnings about uses of `^`, the exclusive or operator, where it appears the code meant exponentiation. Specifically, the warning occurs when the left-hand side is the decimal constant 2 or 10 and the right-hand side is also a decimal constant.

In C and C++, `^` means exclusive or, whereas in some other languages (e.g. TeX and some versions of BASIC) it means exponentiation.

This warning can be silenced by converting one of the operands to hexadecimal as well as by compiling with **-Wno-xor-used-as-pow**.

-Wdisabled-optimization

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

-Wpointer-sign (C and Objective-C only)

Warn for pointer argument passing or assignment with different signedness. This option is only supported for C and Objective-C. It is implied by **-Wall** and by **-Wpedantic**, which can be disabled with **-Wno-pointer-sign**.

This warning is upgraded to an error by **-pedantic-errors**.

-Wstack-protector

This option is only active when **-fstack-protector** is active. It warns about functions that are not protected against stack smashing.

-Woverlength-strings

Warn about string constants that are longer than the “minimum maximum” length specified in the C standard. Modern compilers generally allow string constants that are much longer than the standard's minimum limit, but very portable programs should avoid using longer strings.

The limit applies *after* string constant concatenation, and does not count the trailing NUL. In C90, the limit was 509 characters; in C99, it was raised to 4095. C++98 does not specify a normative minimum maximum, so we do not diagnose overlength strings in C++.

This option is implied by **-Wpedantic**, and can be disabled with **-Wno-overlength-strings**.

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

Issue a warning for any floating constant that does not have a suffix. When used together with **-Wsystem-headers** it warns about such constants in system header files. This can be useful when preparing code to use with the `FLOAT_CONST_DECIMAL64` pragma from the decimal floating-point extension to C99.

-Wno-lto-type-mismatch

During the link-time optimization, do not warn about type mismatches in global declarations from different compilation units. Requires `-flto` to be enabled. Enabled by default.

-Wno-designated-init (C and Objective-C only)

Suppress warnings when a positional initializer is used to initialize a structure that has been marked with the `designated_init` attribute.

-Wzero-init-padding-bits=value

Warn about automatic variable initializers that might not zero initialize padding bits.

Certain languages guarantee zero initialization of padding bits in certain cases. The `-fzero-init-padding-bits=unions` and `-fzero-init-padding-bits=all` options provide additional guarantees that padding bits will be zero initialized in other circumstances.

This warning is mainly intended to find source code that might need to be modified or else recompiled with `-fzero-init-padding-bits=unions` or `-fzero-init-padding-bits=all` in order to retain the behavior that it had when compiled by versions of GCC older than 15.1. It can also be used to find code that might result in uninitialized struct padding unless GCC is invoked with `-fzero-init-padding-bits=all` (regardless of the fact that older versions of GCC would not have guaranteed initialization of struct padding either).

It does not follow that all source code which causes warnings about uninitialized padding bits has undefined behavior: usually, the values of padding bits have no effect on execution of a program.

The two options interact as follows:

	f:standard	f:unions	f:all
W:standard	X	X	X
W:unions	U	X	X
W:all	A	S	X

Letter Meaning

X	No warnings about padding bits.
U	Warnings about padding bits of unions.
S	Warnings about padding bits of structs.
A	Warnings about padding bits of structs and unions.

Examples of union initialization for ISO C23 with `-fzero-init-padding-bits=standard` and `-Wzero-init-padding-bits=unions`:

```
long long q;
unsigned char r;
union U { unsigned char a; long long b; };
struct F { long long a; union U b; };

// Padding bits are explicitly initialized
union U h = {};

// Padding bits might not be initialized
union U i = { r };
```

```

// Padding bits might not be initialized
union U j = { .a = r };

// Union is expected to have no padding bits
union U k = { .b = q };

// Padding bits (of union) are explicitly initialized
struct F l = { q, {}, .b.a = r };

// Padding bits (of union) might not be initialized
struct F m = { q, {}, .b = { .a = r } };

```

In the declaration of `m`, an empty initializer that would otherwise guarantee zero initialization of the padding bits of the union (i.e., storage allocated for member `b` that is not part of member `a`) is overridden by a subsequently listed initializer. This is an anti-pattern.

Whether or not a given initializer causes a warning to be produced can be predicted for simple cases but in general it is target-dependent because the layout of storage is target-dependent. The level of optimization, size, and initial value of the object being initialized also affect whether warnings are produced.

3.10 Options That Control Static Analysis

`-fanalyzer`

This option enables an static analysis of program flow which looks for “interesting” interprocedural paths through the code, and issues warnings for problems found on them.

This analysis is much more expensive than other GCC warnings.

In technical terms, it performs coverage-guided symbolic execution of the code being compiled. It is neither sound nor complete: it can have false positives and false negatives. It is a bug-finding tool, rather than a tool for proving program correctness.

The analyzer is only suitable for use on C code in this release.

Enabling this option effectively enables the following warnings:

```

-Wanalyzer-allocation-size
-Wanalyzer-deref-before-check
-Wanalyzer-div-by-zero
-Wanalyzer-double-fclose
-Wanalyzer-double-free
-Wanalyzer-exposure-through-output-file
-Wanalyzer-exposure-through-uninit-copy
-Wanalyzer-fd-access-mode-mismatch
-Wanalyzer-fd-double-close
-Wanalyzer-fd-leak
-Wanalyzer-fd-phase-mismatch
-Wanalyzer-fd-type-mismatch
-Wanalyzer-fd-use-after-close
-Wanalyzer-fd-use-without-check
-Wanalyzer-file-leak
-Wanalyzer-free-of-non-heap

```

```

-Wanalyzer-imprecise-fp-arithmetic
-Wanalyzer-infinite-loop
-Wanalyzer-infinite-recursion
-Wanalyzer-jump-through-null
-Wanalyzer-malloc-leak
-Wanalyzer-mismatching-deallocation
-Wanalyzer-mkostemp-redundant-flags
-Wanalyzer-mktemp-missing-placeholder
-Wanalyzer-mktemp-of-string-literal
-Wanalyzer-null-argument
-Wanalyzer-null-dereference
-Wanalyzer-out-of-bounds
-Wanalyzer-overlapping-buffers
-Wanalyzer-possible-null-argument
-Wanalyzer-possible-null-dereference
-Wanalyzer-putenv-of-auto-var
-Wanalyzer-shift-count-negative
-Wanalyzer-shift-count-overflow
-Wanalyzer-stale-setjmp-buffer
-Wanalyzer-tainted-allocation-size
-Wanalyzer-tainted-array-index
-Wanalyzer-tainted-assertion
-Wanalyzer-tainted-divisor
-Wanalyzer-tainted-offset
-Wanalyzer-tainted-size
-Wanalyzer-throw-of-unexpected-type
-Wanalyzer-undefined-behavior-ptrdiff
-Wanalyzer-undefined-behavior-strtok
-Wanalyzer-unsafe-call-within-signal-handler
-Wanalyzer-use-after-free
-Wanalyzer-use-of-pointer-in-stale-stack-frame
-Wanalyzer-use-of-uninitialized-value
-Wanalyzer-va-arg-type-mismatch
-Wanalyzer-va-list-exhausted
-Wanalyzer-va-list-leak
-Wanalyzer-va-list-use-after-va-end
-Wanalyzer-write-to-const
-Wanalyzer-write-to-string-literal

```

This option is only available if GCC was configured with analyzer support enabled.

-Wanalyzer-symbol-too-complex

If **-fanalyzer** is enabled, the analyzer uses various heuristics to attempt to track the state of memory, but these can be defeated by sufficiently complicated code.

By default, the analysis silently stops tracking values of expressions if they exceed an internal limit, and falls back to an imprecise representation for such expressions. The **-Wanalyzer-symbol-too-complex** option warns if this occurs.

-Wanalyzer-too-complex

If **-fanalyzer** is enabled, the analyzer uses various heuristics to attempt to explore the control flow and data flow in the program, but these can be defeated by sufficiently complicated code.

By default, the analysis silently stops if the code is too complicated for the analyzer to fully explore and it reaches an internal limit. The `-Wanalyzer-too-complex` option warns if this occurs.

-Wno-analyzer-allocation-size

This warning requires `-fanalyzer`, which enables it; to disable it, use `-Wno-analyzer-allocation-size`.

This diagnostic warns for paths through the code in which a pointer to a buffer is assigned to point at a buffer with a size that is not a multiple of `sizeof(*pointer)`.

See CWE-131: Incorrect Calculation of Buffer Size (<https://cwe.mitre.org/data/definitions/131.html>).

-Wno-analyzer-div-by-zero

This warning requires `-fanalyzer`, which enables it; to disable it, use `-Wno-analyzer-div-by-zero`.

This diagnostic warns for paths through the code which attempt integer division by zero. It is analogous to `-Wdiv-by-zero`, but implemented in a different way.

See CWE-369: Divide By Zero (<https://cwe.mitre.org/data/definitions/369.html>).

-Wno-analyzer-deref-before-check

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-deref-before-check` to disable it.

This diagnostic warns for paths through the code in which a pointer is checked for NULL *after* it has already been dereferenced, suggesting that the pointer could have been NULL. Such cases suggest that the check for NULL is either redundant, or that it needs to be moved to before the pointer is dereferenced.

This diagnostic also considers values passed to a function argument marked with `__attribute__((nonnull))` as requiring a non-NULL value, and thus will complain if such values are checked for NULL after returning from such a function call.

This diagnostic is unlikely to be reported when any level of optimization is enabled, as GCC's optimization logic will typically consider such checks for NULL as being redundant, and optimize them away before the analyzer "sees" them. Hence optimization should be disabled when attempting to trigger this diagnostic.

-Wno-analyzer-double-fclose

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-double-fclose` to disable it.

This diagnostic warns for paths through the code in which a `FILE *` can have `fclose` called on it more than once.

See CWE-1341: Multiple Releases of Same Resource or Handle (<https://cwe.mitre.org/data/definitions/1341.html>).

-Wno-analyzer-double-free

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-double-free` to disable it.

This diagnostic warns for paths through the code in which a pointer can have a deallocator called on it more than once, either `free`, or a deallocator referenced by attribute `malloc`.

See CWE-415: Double Free (<https://cwe.mitre.org/data/definitions/415.html>).

`-Wno-analyzer-exposure-through-output-file`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-exposure-through-output-file` to disable it.

This diagnostic warns for paths through the code in which a security-sensitive value is written to an output file (such as writing a password to a log file).

See CWE-532: Information Exposure Through Log Files (<https://cwe.mitre.org/data/definitions/532.html>).

`-Wanalyzer-exposure-through-uninit-copy`

This warning requires both `-fanalyzer` and the use of a plugin to specify a function that copies across a “trust boundary”. Use `-Wno-analyzer-exposure-through-uninit-copy` to disable it.

This diagnostic warns for “infoleaks” - paths through the code in which uninitialized values are copied across a security boundary (such as code within an OS kernel that copies a partially-initialized struct on the stack to user space).

See CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (<https://cwe.mitre.org/data/definitions/200.html>).

`-Wno-analyzer-fd-access-mode-mismatch`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-fd-access-mode-mismatch` to disable it.

This diagnostic warns for paths through code in which a `read` on a write-only file descriptor is attempted, or vice versa.

This diagnostic also warns for code paths in a which a function with attribute `fd_arg_read (N)` is called with a file descriptor opened with `O_WRONLY` at referenced argument `N` or a function with attribute `fd_arg_write (N)` is called with a file descriptor opened with `O_RDONLY` at referenced argument `N`.

`-Wno-analyzer-fd-double-close`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-fd-double-close` to disable it.

This diagnostic warns for paths through code in which a file descriptor can be closed more than once.

See CWE-1341: Multiple Releases of Same Resource or Handle (<https://cwe.mitre.org/data/definitions/1341.html>).

`-Wno-analyzer-fd-leak`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-fd-leak` to disable it.

This diagnostic warns for paths through code in which an open file descriptor is leaked.

See CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime (<https://cwe.mitre.org/data/definitions/775.html>).

`-Wno-analyzer-fd-phase-mismatch`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-fd-phase-mismatch` to disable it.

This diagnostic warns for paths through code in which an operation is attempted in the wrong phase of a file descriptor's lifetime. For example, it will warn on attempts to call `accept` on a stream socket that has not yet had `listen` successfully called on it.

See CWE-666: Operation on Resource in Wrong Phase of Lifetime (<https://cwe.mitre.org/data/definitions/666.html>).

`-Wno-analyzer-fd-type-mismatch`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-fd-type-mismatch` to disable it.

This diagnostic warns for paths through code in which an operation is attempted on the wrong type of file descriptor. For example, it will warn on attempts to use socket operations on a file descriptor obtained via `open`, or when attempting to use a stream socket operation on a datagram socket.

`-Wno-analyzer-fd-use-after-close`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-fd-use-after-close` to disable it.

This diagnostic warns for paths through code in which a read or write is called on a closed file descriptor.

This diagnostic also warns for paths through code in which a function with attribute `fd_arg (N)` or `fd_arg_read (N)` or `fd_arg_write (N)` is called with a closed file descriptor at referenced argument `N`.

`-Wno-analyzer-fd-use-without-check`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-fd-use-without-check` to disable it.

This diagnostic warns for paths through code in which a file descriptor is used without being checked for validity.

This diagnostic also warns for paths through code in which a function with attribute `fd_arg (N)` or `fd_arg_read (N)` or `fd_arg_write (N)` is called with a file descriptor, at referenced argument `N`, without being checked for validity.

`-Wno-analyzer-file-leak`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-file-leak` to disable it.

This diagnostic warns for paths through the code in which a `<stdio.h> FILE *` stream object is leaked.

See CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime (<https://cwe.mitre.org/data/definitions/775.html>).

-Wno-analyzer-free-of-non-heap

This warning requires **-fanalyzer**, which enables it; use **-Wno-analyzer-free-of-non-heap** to disable it.

This diagnostic warns for paths through the code in which **free** is called on a non-heap pointer (e.g. an on-stack buffer, or a global).

See CWE-590: Free of Memory not on the Heap (<https://cwe.mitre.org/data/definitions/590.html>).

-Wno-analyzer-imprecise-fp-arithmetic

This warning requires **-fanalyzer**, which enables it; use **-Wno-analyzer-imprecise-fp-arithmetic** to disable it.

This diagnostic warns for paths through the code in which floating-point arithmetic is used in locations where precise computation is needed. This diagnostic only warns on use of floating-point operands inside the calculation of an allocation size at the moment.

-Wno-analyzer-infinite-loop

This warning requires **-fanalyzer**, which enables it; use **-Wno-analyzer-infinite-loop** to disable it.

This diagnostics warns for paths through the code which appear to lead to an infinite loop.

Specifically, the analyzer will issue this warning when it "sees" a loop in which:

- no externally-visible work could be being done within the loop
- there is no way to escape from the loop
- the analyzer is sufficiently confident about the program state throughout the loop to know that the above are true

One way for this warning to be emitted is when there is an execution path through a loop for which taking the path on one iteration implies that the same path will be taken on all subsequent iterations.

For example, consider:

```
while (1)
{
    char opcode = *cpu_state.pc;
    switch (opcode)
    {
        case OP_CODE_F00:
            handle_opcode_foo (&cpu_state);
            break;
        case OP_CODE_BAR:
            handle_opcode_bar (&cpu_state);
            break;
    }
}
```

The analyzer will complain for the above case because if **opcode** ever matches none of the cases, the **switch** will follow the implicit **default** case, making the body of the loop be a “no-op” with **cpu_state.pc** unchanged, and thus using the same value of **opcode** on all subsequent iterations, leading to an infinite loop.

See CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop') (<https://cwe.mitre.org/data/definitions/835.html>).

-Wno-analyzer-infinite-recursion

This warning requires **-fanalyzer**, which enables it; use **-Wno-analyzer-infinite-recursion** to disable it.

This diagnostics warns for paths through the code which appear to lead to infinite recursion.

Specifically, when the analyzer "sees" a recursive call, it will compare the state of memory at the entry to the new frame with that at the entry to the previous frame of that function on the stack. The warning is issued if nothing in memory appears to be changing; any changes observed to parameters or globals are assumed to lead to termination of the recursion and thus suppress the warning.

This diagnostic is likely to miss cases of infinite recursion that are converted to iteration by the optimizer before the analyzer "sees" them. Hence optimization should be disabled when attempting to trigger this diagnostic.

Compare with **-Winfinite-recursion**, which provides a similar diagnostic, but is implemented in a different way.

See CWE-674: Uncontrolled Recursion (<https://cwe.mitre.org/data/definitions/674.html>).

-Wno-analyzer-jump-through-null

This warning requires **-fanalyzer**, which enables it; use **-Wno-analyzer-jump-through-null** to disable it.

This diagnostic warns for paths through the code in which a NULL function pointer is called.

-Wno-analyzer-malloc-leak

This warning requires **-fanalyzer**, which enables it; use **-Wno-analyzer-malloc-leak** to disable it.

This diagnostic warns for paths through the code in which a pointer allocated via an allocator is leaked: either **malloc**, or a function marked with attribute **malloc**.

See CWE-401: Missing Release of Memory after Effective Lifetime (<https://cwe.mitre.org/data/definitions/401.html>).

-Wno-analyzer-mismatching-deallocation

This warning requires **-fanalyzer**, which enables it; use **-Wno-analyzer-mismatching-deallocation** to disable it.

This diagnostic warns for paths through the code in which the wrong deallocation function is called on a pointer value, based on which function was used to allocate the pointer value. The diagnostic will warn about mismatches between **free**, scalar **delete** and vector **delete[]**, and those marked as allocator/deallocator pairs using attribute **malloc**.

See CWE-762: Mismatched Memory Management Routines (<https://cwe.mitre.org/data/definitions/762.html>).

-Wno-analyzer-mkostemp-redundant-flags

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-mkostemp-redundant-flags` to disable it.

This diagnostic warns for paths through the code in which `mkostemp` or `mkostemps` is called with flags that include `O_RDWR`, `O_CREAT`, or `O_EXCL`. These flags are already implied by the function and specifying them is unnecessary, and produces errors on some systems.

-Wno-analyzer-mktemp-missing-placeholder

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-mktemp-missing-placeholder` to disable it.

This diagnostic warns for paths through the code in which a function in the `mktemp` family (`mkstemp`, `mkostemp`, `mkstemps`, `mkostemps`, `mkdtemp`, `mktemp`) is called with a template string that does not contain the placeholder `'XXXXXX'` at the expected position.

-Wno-analyzer-mktemp-of-string-literal

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-mktemp-of-string-literal` to disable it.

This diagnostic warns for paths through the code in which a function in the `mktemp` family (`mkstemp`, `mkostemp`, `mkstemps`, `mkostemps`, `mkdtemp`, `mktemp`) is called on a string literal. Since these functions modify their argument in place, passing a string literal leads to undefined behavior.

See STR30-C. Do not attempt to modify string literals (<https://wiki.sei.cmu.edu/confluence/x/VtYxBQ>).

-Wno-analyzer-out-of-bounds

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-out-of-bounds` to disable it.

This diagnostic warns for paths through the code in which a buffer is definitely read or written out-of-bounds. The diagnostic applies for cases where the analyzer is able to determine a constant offset and for accesses past the end of a buffer, also a constant capacity. Further, the diagnostic does limited checking for accesses past the end when the offset as well as the capacity is symbolic.

See CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer (<https://cwe.mitre.org/data/definitions/119.html>).

For cases where the analyzer is able, it will emit a text art diagram visualizing the spatial relationship between the memory region that the analyzer predicts would be accessed, versus the range of memory that is valid to access: whether they overlap, are touching, are close or far apart; which one is before or after in memory, the relative sizes involved, the direction of the access (read vs write), and, in some cases, the values of data involved. This diagram can be suppressed using `-fdiagnostics-text-art-charset=none`.

-Wno-analyzer-overlapping-buffers

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-overlapping-buffers` to disable it.

This diagnostic warns for paths through the code in which overlapping buffers are passed to an API for which the behavior on such buffers is undefined.

Specifically, the diagnostic occurs on calls to the following functions

- `memcpy`
- `strcat`
- `strcpy`

for cases where the buffers are known to overlap.

`-Wno-analyzer-possible-null-argument`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-possible-null-argument` to disable it.

This diagnostic warns for paths through the code in which a possibly-NULL value is passed to a function argument marked with `__attribute__((nonnull))` as requiring a non-NULL value.

See CWE-690: Unchecked Return Value to NULL Pointer Dereference (<https://cwe.mitre.org/data/definitions/690.html>).

`-Wno-analyzer-possible-null-dereference`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-possible-null-dereference` to disable it.

This diagnostic warns for paths through the code in which a possibly-NULL value is dereferenced.

See CWE-690: Unchecked Return Value to NULL Pointer Dereference (<https://cwe.mitre.org/data/definitions/690.html>).

`-Wno-analyzer-null-argument`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-null-argument` to disable it.

This diagnostic warns for paths through the code in which a value known to be NULL is passed to a function argument marked with `__attribute__((nonnull))` as requiring a non-NULL value.

See CWE-476: NULL Pointer Dereference (<https://cwe.mitre.org/data/definitions/476.html>).

`-Wno-analyzer-null-dereference`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-null-dereference` to disable it.

This diagnostic warns for paths through the code in which a value known to be NULL is dereferenced.

See CWE-476: NULL Pointer Dereference (<https://cwe.mitre.org/data/definitions/476.html>).

`-Wno-analyzer-putenv-of-auto-var`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-putenv-of-auto-var` to disable it.

This diagnostic warns for paths through the code in which a call to `putenv` is passed a pointer to an automatic variable or an on-stack buffer.

See POS34-C. Do not call `putenv()` with a pointer to an automatic variable as the argument (<https://wiki.sei.cmu.edu/confluence/x/6NYxBQ>).

-Wno-analyzer-shift-count-negative

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-shift-count-negative` to disable it.

This diagnostic warns for paths through the code in which a shift is attempted with a negative count. It is analogous to the `-Wshift-count-negative` diagnostic implemented in the C/C++ front ends, but is implemented based on analyzing interprocedural paths, rather than merely parsing the syntax tree. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

-Wno-analyzer-shift-count-overflow

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-shift-count-overflow` to disable it.

This diagnostic warns for paths through the code in which a shift is attempted with a count greater than or equal to the precision of the operand's type. It is analogous to the `-Wshift-count-overflow` diagnostic implemented in the C/C++ front ends, but is implemented based on analyzing interprocedural paths, rather than merely parsing the syntax tree. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

-Wno-analyzer-stale-setjmp-buffer

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-stale-setjmp-buffer` to disable it.

This diagnostic warns for paths through the code in which `longjmp` is called to rewind to a `jmp_buf` relating to a `setjmp` call in a function that has returned. When `setjmp` is called on a `jmp_buf` to record a rewind location, it records the stack frame. The stack frame becomes invalid when the function containing the `setjmp` call returns. Attempting to rewind to it via `longjmp` would reference a stack frame that no longer exists, and likely lead to a crash (or worse).

-Wno-analyzer-tainted-allocation-size

This warning requires `-fanalyzer` which enables it; use `-Wno-analyzer-tainted-allocation-size` to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the size of an allocation without being sanitized, so that an attacker could inject an excessively large allocation and potentially cause a denial of service attack.

See CWE-789: Memory Allocation with Excessive Size Value (<https://cwe.mitre.org/data/definitions/789.html>).

-Wno-analyzer-tainted-assertion

This warning requires `-fanalyzer` which enables it; use `-Wno-analyzer-tainted-assertion` to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as part of a condition without being first

sanitized, and that condition guards a call to a function marked with attribute `noreturn` (such as the function `__builtin_unreachable`). Such functions typically indicate abnormal termination of the program, such as for assertion failure handlers. For example:

```
assert (some_tainted_value < SOME_LIMIT);
```

In such cases:

- when assertion-checking is enabled: an attacker could trigger a denial of service by injecting an assertion failure
- when assertion-checking is disabled, such as by defining `NDEBUG`, an attacker could inject data that subverts the process, since it presumably violates a precondition that is being assumed by the code.

Note that when assertion-checking is disabled, the assertions are typically removed by the preprocessor before the analyzer has a chance to "see" them, so this diagnostic can only generate warnings on builds in which assertion-checking is enabled.

For the purpose of this warning, any function marked with attribute `noreturn` is considered as a possible assertion failure handler, including `__builtin_unreachable`. Note that these functions are sometimes removed by the optimizer before the analyzer "sees" them. Hence optimization should be disabled when attempting to trigger this diagnostic.

See CWE-617: Reachable Assertion (<https://cwe.mitre.org/data/definitions/617.html>).

The warning can also report problematic constructions such as

```
switch (some_tainted_value) {
case 0:
    /* [...etc; various valid cases omitted...] */
    break;

default:
    __builtin_unreachable (); /* BUG: attacker can trigger this */
}
```

despite the above not being an assertion failure, strictly speaking.

-Wno-analyzer-tainted-array-index

This warning requires `-fanalyzer` which enables it; use `-Wno-analyzer-tainted-array-index` to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the index of an array access without being sanitized, so that an attacker could inject an out-of-bounds access.

See CWE-129: Improper Validation of Array Index (<https://cwe.mitre.org/data/definitions/129.html>).

-Wno-analyzer-tainted-divisor

This warning requires `-fanalyzer` which enables it; use `-Wno-analyzer-tainted-divisor` to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the divisor in a division or modulus

operation without being sanitized, so that an attacker could inject a division-by-zero.

See CWE-369: Divide By Zero (<https://cwe.mitre.org/data/definitions/369.html>).

`-Wno-analyzer-tainted-offset`

This warning requires `-fanalyzer` which enables it; use `-Wno-analyzer-tainted-offset` to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as a pointer offset without being sanitized, so that an attacker could inject an out-of-bounds access.

See CWE-823: Use of Out-of-range Pointer Offset (<https://cwe.mitre.org/data/definitions/823.html>).

`-Wno-analyzer-tainted-size`

This warning requires `-fanalyzer` which enables it; use `-Wno-analyzer-tainted-size` to disable it.

This diagnostic warns for paths through the code in which a value that could be under an attacker's control is used as the size of an operation such as `memset` without being sanitized, so that an attacker could inject an out-of-bounds access.

See CWE-129: Improper Validation of Array Index (<https://cwe.mitre.org/data/definitions/129.html>).

`-Wno-analyzer-throw-of-unexpected-type`

This warning requires `-fanalyzer` which enables it; use `-Wno-analyzer-throw-of-unexpected-type` to disable it. Dynamic exception specifications are only available in C++14 and earlier.

This diagnostic warns for paths through the code in which a an exception is thrown from a function with a dynamic exception specification where the exception does not comply with the specification.

`-Wno-analyzer-undefined-behavior-ptrdiff`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-undefined-behavior-ptrdiff` to disable it.

This diagnostic warns for paths through the code in which a pointer subtraction occurs where the pointers refer to different chunks of memory. Such code relies on undefined behavior, as pointer subtraction is only defined for cases where both pointers point to within (or just after) the same array.

See CWE-469: Use of Pointer Subtraction to Determine Size (<https://cwe.mitre.org/data/definitions/469.html>).

`-Wno-analyzer-undefined-behavior-strtok`

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-undefined-behavior-strtok` to disable it.

This diagnostic warns for paths through the code in which a call is made to `strtok` with undefined behavior.

Specifically, passing NULL as the first parameter for the initial call to `strtok` within a process has undefined behavior.

-Wno-analyzer-unsafe-call-within-signal-handler

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-unsafe-call-within-signal-handler` to disable it.

This diagnostic warns for paths through the code in which a function known to be `async-signal-unsafe` (such as `fprintf`) is called from a signal handler.

See CWE-479: Signal Handler Use of a Non-reentrant Function (<https://cwe.mitre.org/data/definitions/479.html>).

-Wno-analyzer-use-after-free

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-use-after-free` to disable it.

This diagnostic warns for paths through the code in which a pointer is used after a deallocator is called on it: either `free`, or a deallocator referenced by attribute `malloc`.

See CWE-416: Use After Free (<https://cwe.mitre.org/data/definitions/416.html>).

-Wno-analyzer-use-of-pointer-in-stale-stack-frame

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-use-of-pointer-in-stale-stack-frame` to disable it.

This diagnostic warns for paths through the code in which a pointer is dereferenced that points to a variable in a stale stack frame.

-Wno-analyzer-va-arg-type-mismatch

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-va-arg-type-mismatch` to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects an attempt to use `va_arg` to extract a value passed to a variadic call, but uses a type that does not match that of the expression passed to the call.

See CWE-686: Function Call With Incorrect Argument Type (<https://cwe.mitre.org/data/definitions/686.html>).

-Wno-analyzer-va-list-exhausted

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-va-list-exhausted` to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects an attempt to use `va_arg` to access the next value passed to a variadic call, but all of the values in the `va_list` have already been consumed.

See CWE-685: Function Call With Incorrect Number of Arguments (<https://cwe.mitre.org/data/definitions/685.html>).

-Wno-analyzer-va-list-leak

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-va-list-leak` to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects that `va_start` or `va_copy` has been called on a `va_list` without a corresponding call to `va_end`.

-Wno-analyzer-va-list-use-after-va-end

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-va-list-use-after-va-end` to disable it.

This diagnostic warns for interprocedural paths through the code for which the analyzer detects an attempt to use a `va_list` after `va_end` has been called on it. `va_list`.

-Wno-analyzer-write-to-const

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-write-to-const` to disable it.

This diagnostic warns for paths through the code in which the analyzer detects an attempt to write through a pointer to a `const` object. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

-Wno-analyzer-write-to-string-literal

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-write-to-string-literal` to disable it.

This diagnostic warns for paths through the code in which the analyzer detects an attempt to write through a pointer to a string literal. However, the analyzer does not prioritize detection of such paths, so false negatives are more likely relative to other warnings.

-Wno-analyzer-use-of-uninitialized-value

This warning requires `-fanalyzer`, which enables it; use `-Wno-analyzer-use-of-uninitialized-value` to disable it.

This diagnostic warns for paths through the code in which an uninitialized value is used.

See CWE-457: Use of Uninitialized Variable (<https://cwe.mitre.org/data/definitions/457.html>).

The analyzer has hardcoded knowledge about the behavior of the following memory-management functions:

- `alloca`
- The built-in functions `__builtin_alloc`, `__builtin_alloc_with_align`, `__builtin_calloc`, `__builtin_free`, `__builtin_malloc`, `__builtin_memcpy`, `__builtin_memcpy_chk`, `__builtin_memset`, `__builtin_memset_chk`, `__builtin_realloc`, `__builtin_stack_restore`, and `__builtin_stack_save`
- `calloc`
- `free`
- `malloc`
- `memset`
- operator `delete`

- operator delete []
- operator new
- operator new []
- realloc
- strdup
- strdup

of the following functions for working with file descriptors:

- open
- close
- creat
- dup, dup2 and dup3
- isatty
- pipe, and pipe2
- read
- write
- socket, bind, listen, accept, and connect

of the following functions for working with `<stdio.h>` streams:

- The built-in functions `__builtin_fprintf`, `__builtin_fprintf_unlocked`, `__builtin_fputc`, `__builtin_fputc_unlocked`, `__builtin_fputs`, `__builtin_fputs_unlocked`, `__builtin_fwrite`, `__builtin_fwrite_unlocked`, `__builtin_printf`, `__builtin_printf_unlocked`, `__builtin_putc`, `__builtin_putchar`, `__builtin_putchar_unlocked`, `__builtin_putc_unlocked`, `__builtin_puts`, `__builtin_puts_unlocked`, `__builtin_vfprintf`, and `__builtin_vprintf`
- fopen
- fclose
- ferror
- fgets
- fgets_unlocked
- fileno
- fread
- getc
- getchar
- fprintf
- printf
- fwrite

and of the following functions:

- The built-in functions `__builtin_expect`, `__builtin_expect_with_probability`, `__builtin_strchr`, `__builtin_strcpy`, `__builtin_strcpy_chk`, `__builtin_strlen`, `__builtin_va_copy`, and `__builtin_va_start`

- The GNU extensions `error` and `error_at_line`
- `getpass`
- `longjmp`
- `putenv`
- `setjmp`
- `siglongjmp`
- `signal`
- `sigsetjmp`
- `strcat`
- `strchr`
- `strlen`

In addition, various functions with an `__analyzer_` prefix have special meaning to the analyzer, described in the GCC Internals manual. Various internal parameters, settable via `--param`, are used to control the exploration; these are also documented in the GCC Internals manual.

The following options control the analyzer.

`-fanalyzer-assume-nothrow`

By default, if `-fexceptions` is enabled, the analyzer will assume that a call to any function without attribute `nothrow` could throw an exception. This can help detect execution paths that leak due to exceptions bypassing clean-up code, but could lead to false positives when using headers where the author has not added the `nothrow` attribute.

If `-fanalyzer-assume-nothrow` is enabled, then the analyzer will assume that external functions do not throw exceptions. This may be useful for reducing “noise” from the analyzer when enabling `-fexceptions` on C code, but could hide true issues if an exception could be raised by something the C code calls.

`-fanalyzer-call-summaries`

Simplify interprocedural analysis by computing the effect of certain calls, rather than exploring all paths through the function from callsite to each possible return.

If enabled, call summaries are only used for functions with more than one call site, and that are sufficiently complicated.

`-fanalyzer-checker=name`

Restrict the analyzer to run just the named checker, and enable it.

`-fanalyzer-debug-text-art-headings`

This option is intended for analyzer developers. If enabled, the analyzer will add extra annotations to any diagrams it generates.

`-fno-analyzer-feasibility`

This option is intended for analyzer developers.

By default the analyzer verifies that there is a feasible control flow path for each diagnostic it emits: that the conditions that hold are not mutually exclusive.

Diagnostics for which no feasible path can be found are rejected. This filtering can be suppressed with `-fno-analyzer-feasibility`, for debugging issues in this code.

-fanalyzer-fine-grained

Does nothing. Preserved for backward compatibility.

-fanalyzer-show-duplicate-count

This option is intended for analyzer developers: if multiple diagnostics have been detected as being duplicates of each other, it emits a note when reporting the best diagnostic, giving the number of additional diagnostics that were suppressed by the deduplication logic.

-fanalyzer-show-events-in-system-headers

By default the analyzer emits simplified diagnostics paths by hiding events fully located within a system header. With `-fanalyzer-show-events-in-system-headers` such events are no longer suppressed.

-fno-analyzer-simplify-supergraph

This option is intended for analyzer developers.

By default, the analyzer performs various simplifications to the program supergraph before analyzing it. With `-fno-analyzer-simplify-supergraph` this simplification can be suppressed, for debugging issues with it.

-fno-analyzer-state-merge

This option is intended for analyzer developers.

By default the analyzer attempts to simplify analysis by merging sufficiently similar states at each program point as it builds its “exploded graph”. With `-fno-analyzer-state-merge` this merging can be suppressed, for debugging state-handling issues.

-fno-analyzer-state-purge

This option is intended for analyzer developers.

By default the analyzer attempts to simplify analysis by purging aspects of state at a program point that appear to no longer be relevant e.g. the values of locals that aren’t accessed later in the function and which aren’t relevant to leak analysis.

With `-fno-analyzer-state-purge` this purging of state can be suppressed, for debugging state-handling issues.

-fno-analyzer-suppress-followups

This option is intended for analyzer developers.

By default the analyzer will stop exploring an execution path after encountering certain diagnostics, in order to avoid potentially issuing a cascade of follow-up diagnostics.

The diagnostics that terminate analysis along a path are:

- `-Wanalyzer-null-argument`
- `-Wanalyzer-null-dereference`
- `-Wanalyzer-use-after-free`

- `-Wanalyzer-use-of-pointer-in-stale-stack-frame`
- `-Wanalyzer-use-of-uninitialized-value`

With `-fno-analyzer-suppress-followups` the analyzer will continue to explore such paths even after such diagnostics, which may be helpful for debugging issues in the analyzer, or for microbenchmarks for detecting undefined behavior.

`-fanalyzer-transitivity`

This option enables transitivity of constraints within the analyzer.

`-fno-analyzer-undo-inlining`

This option is intended for analyzer developers.

`-fanalyzer` runs relatively late compared to other code analysis tools, and some optimizations have already been applied to the code. In particular function inlining may have occurred, leading to the interprocedural execution paths emitted by the analyzer containing function frames that don't correspond to those in the original source code.

By default the analyzer attempts to reconstruct the original function frames, and to emit events showing the inlined calls.

With `-fno-analyzer-undo-inlining` this attempt to reconstruct the original frame information can be disabled, which may be of help when debugging issues in the analyzer.

`-fanalyzer-verbose-edges`

This option is intended for analyzer developers. It enables more verbose, lower-level detail in the descriptions of control flow within diagnostic paths.

`-fanalyzer-verbose-state-changes`

This option is intended for analyzer developers. It enables more verbose, lower-level detail in the descriptions of events relating to state machines within diagnostic paths.

`-fanalyzer-verbosity=level`

This option controls the complexity of the control flow paths that are emitted for analyzer diagnostics.

The *level* can be one of:

- | | |
|-----|---|
| ‘0’ | At this level, interprocedural call and return events are displayed, along with the most pertinent state-change events relating to a diagnostic. For example, for a <code>double-free</code> diagnostic, both calls to <code>free</code> will be shown. |
| ‘1’ | As per the previous level, but also show events for the entry to each function. |
| ‘2’ | As per the previous level, but also show events relating to control flow that are significant to triggering the issue (e.g. “true path taken” at a conditional).
This level is the default. |
| ‘3’ | As per the previous level, but show all control flow events, not just significant ones. |

‘4’ This level is intended for analyzer developers; it adds various other events intended for debugging the analyzer.

- fdump-analyzer
Dump internal details about what the analyzer is doing to *file.analyzer.txt*.
-fdump-analyzer-stderr overrides this option.
- fdump-analyzer-stderr
Dump internal details about what the analyzer is doing to stderr. This option overrides -fdump-analyzer.
- fdump-analyzer-callgraph
Dump a representation of the call graph suitable for viewing with GraphViz to *file.callgraph.dot*.
- fdump-analyzer-exploded-graph
Dump a representation of the “exploded graph” suitable for viewing with GraphViz to *file.eg.dot*. Nodes are color-coded based on state-machine states to emphasize state changes.
- fdump-analyzer-exploded-nodes
Emit diagnostics showing where nodes in the “exploded graph” are in relation to the program source.
- fdump-analyzer-exploded-nodes-2
Dump a textual representation of the “exploded graph” to *file.eg.txt*.
- fdump-analyzer-exploded-nodes-3
Dump a textual representation of the “exploded graph” to one dump file per node, to *file.eg-id.txt*. This is typically a large number of dump files.
- fdump-analyzer-exploded-paths
Dump a textual representation of the “exploded path” for each diagnostic to *file.idx.kind.epath.txt*.
- fdump-analyzer-feasibility
Dump internal details about the analyzer’s search for feasible paths. The details are written in a form suitable for viewing with GraphViz to filenames of the form *file.*.fg.dot*, *file.*.tg.dot*, and *file.*.fpath.txt*.
- fdump-analyzer-infinite-loop
Dump internal details about the analyzer’s search for infinite loops. The details are written in a form suitable for viewing with GraphViz to filenames of the form *file.*.infinite-loop.dot*.
- fdump-analyzer-json
Dump a compressed JSON representation of analyzer internals to *file.analyzer.json.gz*. The precise format is subject to change.
- fdump-analyzer-state-purge
As per -fdump-analyzer-supergraph, dump a representation of the “supergraph” suitable for viewing with GraphViz, but annotate the graph with information on what state will be purged at each node. The graph is written to *file.state-purge.dot*.

-fdump-analyzer-supergraph

Dump representations of the “supergraph” suitable for viewing with GraphViz to *file.supergraph.index.kind.dot*. These show all of the control flow graphs in the program, at various stages of the analysis. The precise set of dumps and what they show is subject to change.

-fdump-analyzer-untracked

Emit custom warnings with internal details intended for analyzer developers.

3.11 Options for Debugging Your Program

To tell GCC to emit extra information for use by a debugger, in almost all cases you need only to add **-g** to your other options. Some debug formats can co-exist (like DWARF with CTF) when each of them is enabled explicitly by adding the respective command-line option to your other options.

GCC allows you to use **-g** with **-O**. The shortcuts taken by optimized code may occasionally be surprising: some variables you declared may not exist at all; flow of control may briefly move where you did not expect it; some statements may not be executed because they compute constant results or their values are already at hand; some statements may execute in different places because they have been moved out of loops. Nevertheless it is possible to debug optimized output. This makes it reasonable to use the optimizer for programs that might have bugs.

If you are not using some other optimization option, consider using **-Og** (see Section 3.12 [Optimize Options], page 196) with **-g**. With no **-O** option at all, some compiler passes that collect information useful for debugging do not run at all, so that **-Og** may result in a better debugging experience.

-g

--debug Produce debugging information in the operating system’s native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.

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

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

-gdwarf**-gdwarf-version**

Produce debugging information in DWARF format (if that is supported). The value of *version* may be either 2, 3, 4 or 5; the default version for most targets is 5 (with the exception of VxWorks, TPF and Darwin / macOS, which default to version 2, and AIX, which defaults to version 4).

Note that with DWARF Version 2, some ports require and always use some non-conflicting DWARF 3 extensions in the unwind tables.

Version 4 may require GDB 7.0 and `-fvar-tracking-assignments` for maximum benefit. Version 5 requires GDB 8.0 or higher.

GCC no longer supports DWARF Version 1, which is substantially different than Version 2 and later. For historical reasons, some other DWARF-related options such as `-fno-dwarf2-cfi-asm` retain a reference to DWARF Version 2 in their names, but apply to all currently-supported versions of DWARF.

-gbtf Request BTF debug information. BTF is the default debugging format for the eBPF target. On other targets, like x86, BTF debug information can be generated along with DWARF debug information when both of the debug formats are enabled explicitly via their respective command-line options.

-gprune-btf

-gno-prune-btf

Prune BTF information before emission. When pruning, only type information for types used by global variables and file-scope functions will be emitted. If compiling for the BPF target with BPF CO-RE enabled, type information will also be emitted for types used in BPF CO-RE relocations. In addition, struct and union types which are only referred to via pointers from members of other struct or union types shall be pruned and replaced with BTF_KIND_FWD, as though those types were only present in the input as forward declarations.

This option substantially reduces the size of produced BTF information, but at significant loss in the amount of detailed type information. It is primarily useful when compiling for the BPF target, to minimize the size of the resulting object, and to eliminate BTF information which is not immediately relevant to the BPF program loading process.

This option is enabled by default for the BPF target when generating BTF information.

-gctf

-gctflevel

Request CTF debug information and use level to specify how much CTF debug information should be produced. If `-gctf` is specified without a value for level, the default level of CTF debug information is 2.

CTF debug information can be generated along with DWARF debug information when both of the debug formats are enabled explicitly via their respective command line options.

Level 0 produces no CTF debug information at all. Thus, `-gctf0` negates `-gctf`.

Level 1 produces CTF information for tracebacks only. This includes callsite information, but does not include type information.

Level 2 produces type information for entities (functions, data objects etc.) at file-scope or global-scope only.

-gvms

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

-gcodeview

Produce debugging information in CodeView debug format (if that is supported). This is the format used by Microsoft Visual C++ on Windows.

-glevel**-ggdblevel****-gvmslevel**

Request debugging information and also use *level* to specify how much information. The default level is 2.

Level 0 produces no debug information at all. Thus, **-g0** negates **-g**.

Level 1 produces minimal information, enough for making backtraces in parts of the program that you don't plan to debug. This includes descriptions of functions and external variables, and line number tables, but no information about local variables.

Level 2 is the normal setting to produce debuggable code. The debug output includes all the information from level 1, plus information about local variables and typedefs to allow stepping through code and examining its state. For C++ code, additional information is also emitted to assist in debugging namespace, class, and template functions.

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

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

-gdwarf does not accept a concatenated debug level, to avoid confusion with **-gdwarf-level1**. Instead use an additional **-glevel** option to change the debug level for DWARF.

-fno-eliminate-unused-debug-symbols

By default, no debug information is produced for symbols that are not actually used. Use this option if you want debug information for all symbols.

-femit-class-debug-always

Instead of emitting debugging information for a C++ class in only one object file, emit it in all object files using the class. This option should be used only with debuggers that are unable to handle the way GCC normally emits debugging information for classes because using this option increases the size of debugging information by as much as a factor of two.

-fno-merge-debug-strings

Direct the linker to not merge together strings in the debugging information that are identical in different object files. Merging is not supported by all assemblers or linkers. Merging decreases the size of the debug information in the output file at the cost of increasing link processing time. Merging is enabled by default.

-fdebug-prefix-map=old=new

When compiling files residing in directory *old*, record debugging information describing them as if the files resided in directory *new* instead. This can be

used to replace a build-time path with an install-time path in the debug info. It can also be used to change an absolute path to a relative path by using `.` for *new*. This can give more reproducible builds, which are location independent, but may require an extra command to tell GDB where to find the source files. See also `-ffile-prefix-map` and `-fcanon-prefix-map`.

`-fvar-tracking`

Run variable tracking pass. It computes where variables are stored at each position in code. Better debugging information is then generated (if the debugging information format supports this information).

It is enabled by default when compiling with optimization (`-Os`, `-O`, `-O2`, ...), debugging information (`-g`) and the debug info format supports it.

`-fvar-tracking-assignments`

Annotate assignments to user variables early in the compilation and attempt to carry the annotations over throughout the compilation all the way to the end, in an attempt to improve debug information while optimizing. Use of `-gdwarf-4` is recommended along with it.

It can be enabled even if `var-tracking` is disabled, in which case annotations are created and maintained, but discarded at the end. By default, this flag is enabled together with `-fvar-tracking`, except when selective scheduling is enabled.

`-fvar-tracking-uninit`

`-fno-var-tracking-uninit`

Perform variable tracking and also mark uninitialized variables in the debug information. This flag is enabled by default by `-fvar-tracking`; it also implies `-fvar-tracking`. To do variable tracking without marking uninitialized variables, use `-fvar-tracking -fno-var-tracking-uninit`.

`-gsplit-dwarf`

If DWARF debugging information is enabled, separate as much debugging information as possible into a separate output file with the extension `.dwo`. This option allows the build system to avoid linking files with debug information. To be useful, this option requires a debugger capable of reading `.dwo` files.

`-gdwarf32`

`-gdwarf64`

If DWARF debugging information is enabled, the `-gdwarf32` selects the 32-bit DWARF format and the `-gdwarf64` selects the 64-bit DWARF format. The default is target specific, on most targets it is `-gdwarf32` though. The 32-bit DWARF format is smaller, but can't support more than 2GiB of debug information in any of the DWARF debug information sections. The 64-bit DWARF format allows larger debug information and might not be well supported by all consumers yet.

`-gdescribe-dies`

Add description attributes to some DWARF DIEs that have no name attribute, such as artificial variables, external references and call site parameter DIEs.

- gpubnames**
Generate DWARF `.debug_pubnames` and `.debug_pubtypes` sections.
- ggnu-pubnames**
Generate `.debug_pubnames` and `.debug_pubtypes` sections in a format suitable for conversion into a GDB index. This option is only useful with a linker that can produce GDB index version 7.
- gno-pubnames**
Don't generate DWARF `.debug_pubnames` and `.debug_pubtypes` sections.
- fdebug-types-section**
When using DWARF Version 4 or higher, type DIEs can be put into their own `.debug_types` section instead of making them part of the `.debug_info` section. It is more efficient to put them in a separate comdat section since the linker can then remove duplicates. But not all DWARF consumers support `.debug_types` sections yet and on some objects `.debug_types` produces larger instead of smaller debugging information.
- grecord-gcc-switches**
- gno-record-gcc-switches**
This switch causes the command-line options used to invoke the compiler that may affect code generation to be appended to the `DW_AT_producer` attribute in DWARF debugging information. The options are concatenated with spaces separating them from each other and from the compiler version. It is enabled by default. See also **-frecord-gcc-switches** for another way of storing compiler options into the object file.
- gstrict-dwarf**
Disallow using extensions of later DWARF standard version than selected with **-gdwarf-version**. On most targets using non-conflicting DWARF extensions from later standard versions is allowed.
- gno-strict-dwarf**
Allow using extensions of later DWARF standard version than selected with **-gdwarf-version**.
- gas-loc-support**
Inform the compiler that the assembler supports `.loc` directives. It may then use them for the assembler to generate DWARF2+ line number tables.
This is generally desirable, because assembler-generated line-number tables are a lot more compact than those the compiler can generate itself.
This option is enabled by default if, at GCC configure time, the assembler is found to support such directives.
- gno-as-loc-support**
Force GCC to generate DWARF2+ line number tables internally, if DWARF2+ line number tables are to be generated.
- gas-locview-support**
Inform the compiler that the assembler supports `view` assignment and reset assertion checking in `.loc` directives.

This option is enabled by default if, at GCC configure time, the assembler is found to support them.

-gno-as-locview-support

Force GCC to assign view numbers internally, if **-gvariable-location-views** are explicitly requested.

-gcolumn-info

-gno-column-info

Emit location column information into DWARF debugging information, rather than just file and line. This option is enabled by default.

-gstatement-frontiers

-gno-statement-frontiers

This option causes GCC to create markers in the internal representation at the beginning of statements, and to keep them roughly in place throughout compilation, using them to guide the output of **is_stmt** markers in the line number table. This is enabled by default when compiling with optimization (**-Os**, **-O1**, **-O2**, ...), and outputting DWARF 2 debug information at the normal level.

-gvariable-location-views

-gvariable-location-views=incompat5

-gno-variable-location-views

Augment variable location lists with progressive view numbers implied from the line number table. This enables debug information consumers to inspect state at certain points of the program, even if no instructions associated with the corresponding source locations are present at that point. If the assembler lacks support for view numbers in line number tables, this causes the compiler to emit the line number table, which generally makes them somewhat less compact. The augmented line number tables and location lists are fully backward-compatible, so they can be consumed by debug information consumers that are not aware of these augmentations, but they won't derive any benefit from them either.

This is enabled by default when outputting DWARF 2 debug information at the normal level, as long as there is assembler support, **-fvar-tracking-assignments** is enabled and **-gstrict-dwarf** is not. When assembler support is not available, this may still be enabled, but it forces GCC to output internal line number tables, and if **-ginternal-reset-location-views** is not enabled, that most certainly leads to silently mismatching location views.

There is a proposed representation for view numbers that is not backward compatible with the location list format introduced in DWARF 5, that can be enabled with **-gvariable-location-views=incompat5**. This option may be removed in the future, is only provided as a reference implementation of the proposed representation. Debug information consumers are not expected to support this extended format, and they would be rendered unable to decode location lists using it.

-ginternal-reset-location-views**-gno-internal-reset-location-views**

Attempt to determine location views that can be omitted from location view lists. This requires the compiler to have very accurate instruction length estimates, which isn't always the case, and it may cause incorrect view lists to be generated silently when using an assembler that does not support location view lists. The GNU assembler flags any such error as a **view number mismatch**. This is only enabled on ports that define a reliable estimation function.

-ginline-points**-gno-inline-points**

Generate extended debug information for inlined functions. Location view tracking markers are inserted at inlined entry points, so that address and view numbers can be computed and output in debug information. This can be enabled independently of location views, in which case the view numbers won't be output, but it can only be enabled along with statement frontiers, and it is only enabled by default if location views are enabled.

-gz[=type]

Produce compressed debug sections in DWARF format, if that is supported. If *type* is not given, the default type depends on the capabilities of the assembler and linker used. *type* may be one of **'none'** (don't compress debug sections), **'zlib'** (use zlib compression in ELF gABI format), or **'zstd'** (use zstd compression in ELF gABI format). If the linker doesn't support writing compressed debug sections, the option is rejected. Otherwise, if the assembler does not support them, **-gz** is silently ignored when producing object files.

-femit-struct-debug-baseonly

Emit debug information for struct-like types only when the base name of the compilation source file matches the base name of file in which the struct is defined.

This option substantially reduces the size of debugging information, but at significant potential loss in type information to the debugger. See **-femit-struct-debug-reduced** for a less aggressive option. See **-femit-struct-debug-detailed** for more detailed control.

This option works only with DWARF debug output.

-femit-struct-debug-reduced

Emit debug information for struct-like types only when the base name of the compilation source file matches the base name of file in which the type is defined, unless the struct is a template or defined in a system header.

This option significantly reduces the size of debugging information, with some potential loss in type information to the debugger. See **-femit-struct-debug-baseonly** for a more aggressive option. See **-femit-struct-debug-detailed** for more detailed control.

This option works only with DWARF debug output.

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

Specify the struct-like types for which the compiler generates debug information. The intent is to reduce duplicate struct debug information between different object files within the same program.

This option is a detailed version of **-femit-struct-debug-reduced** and **-femit-struct-debug-baseonly**, which serves for most needs.

A specification has the syntax

```
['dir:' | 'ind:']['ord:' | 'gen:']('any' | 'sys' | 'base' | 'none')
```

The optional first word limits the specification to structs that are used directly (*'dir:'*) or used indirectly (*'ind:'*). A struct type is used directly when it is the type of a variable, member. Indirect uses arise through pointers to structs. That is, when use of an incomplete struct is valid, the use is indirect. An example is `'struct one direct; struct two * indirect;'`.

The optional second word limits the specification to ordinary structs (*'ord:'*) or generic structs (*'gen:'*). Generic structs are a bit complicated to explain. For C++, these are non-explicit specializations of template classes, or non-template classes within the above. Other programming languages have generics, but **-femit-struct-debug-detailed** does not yet implement them.

The third word specifies the source files for those structs for which the compiler should emit debug information. The values *'none'* and *'any'* have the normal meaning. The value *'base'* means that the base of name of the file in which the type declaration appears must match the base of the name of the main compilation file. In practice, this means that when compiling `foo.c`, debug information is generated for types declared in that file and `foo.h`, but not other header files. The value *'sys'* means those types satisfying *'base'* or declared in system or compiler headers.

You may need to experiment to determine the best settings for your application.

The default is **-femit-struct-debug-detailed=all**.

This option works only with DWARF debug output.

-fno-dwarf2-cfi-asm

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

-fno-eliminate-unused-debug-types

Normally, when producing DWARF output, GCC avoids producing debug symbol output for types that are nowhere used in the source file being compiled. Sometimes it is useful to have GCC emit debugging information for all types declared in a compilation unit, regardless of whether or not they are actually used in that compilation unit, for example if, in the debugger, you want to cast a value to a type that is not actually used in your program (but is declared). More often, however, this results in a significant amount of wasted space.

3.12 Options That Control Optimization

These options control various sorts of optimizations.

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you expect from the source code.

Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.

The compiler performs optimization based on the knowledge it has of the program. Compiling multiple files at once to a single output file mode allows the compiler to use information gained from all of the files when compiling each of them.

Not all optimizations are controlled directly by a flag. Only optimizations that have a flag are listed in this section.

Most optimizations are completely disabled at `-O0` or if an `-O` level is not set on the command line, even if individual optimization flags are specified. Similarly, `-Og` suppresses many optimization passes.

Depending on the target and how GCC was configured, a slightly different set of optimizations may be enabled at each `-O` level than those listed here. You can invoke GCC with `-Q --help=optimizers` to find out the exact set of optimizations that are enabled at each level. See Section 3.2 [Overall Options], page 34, for examples.

`-O`

`-O1`

`--optimize`

Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function.

With `-O`, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

`-O` is the recommended optimization level for large machine-generated code as a sensible balance between time taken to compile and memory use: higher optimization levels perform optimizations with greater algorithmic complexity than at `-O`.

`-O` turns on the following optimization flags:

- `-fauto-inc-dec`
- `-fbranch-count-reg`
- `-fcombine-stack-adjustments`
- `-fcompare-elim`
- `-fcprop-registers`
- `-fdce`
- `-fdefer-pop`
- `-fdelayed-branch`
- `-fdse`
- `-fforward-propagate`
- `-fguess-branch-probability`
- `-fif-conversion`
- `-fif-conversion2`
- `-finline-functions-called-once`
- `-fipa-modref`
- `-fipa-profile`

```

-fipa-pure-const
-fipa-reference
-fipa-reference-addressable
-fivopts
-fmerge-constants
-fmove-loop-invariants
-fmove-loop-stores
-fomit-frame-pointer
-freorder-blocks
-fshrink-wrap
-fshrink-wrap-separate
-fsplit-wide-types
-fssa-backprop
-fssa-phiopt
-ftime-bit-ccp
-ftime-ccp
-ftime-ch
-ftime-coalesce-vars
-ftime-copy-prop
-ftime-dce
-ftime-dominator-opts
-ftime-dse
-ftime-forwprop
-ftime-fre
-ftime-hiprop
-ftime-pta
-ftime-scev-cprop
-ftime-sink
-ftime-slsr
-ftime-sra
-ftime-ter
-funit-at-a-time

```

-O2 Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to **-O**, this option increases both compilation time and the performance of the generated code.

-O2 turns on all optimization flags specified by **-O1**. It also turns on the following optimization flags:

```

-falign-functions -falign-jumps
-falign-labels -falign-loops
-fcaller-saves
-fcode-hoisting
-fcrossjumping
-fcse-follow-jumps -fcse-skip-blocks
-fdelete-null-pointer-checks -fdep-fusion
-fdevirtualize -fdevirtualize-speculatively
-fexpensive-optimizations
-ffinite-loops
-fgcse -fgcse-lm
-fhoist-adjacent-loads
-finline-functions
-finline-small-functions
-findirect-inlining
-fipa-bit-cp -fipa-cp -fipa-icf
-fipa-ra -fipa-sra -fipa-vrp
-fisolate-erroneous-paths-dereference
-flra-remat

```

```

-foptimize-crc
-foptimize-sibling-calls
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-freorder-blocks-algorithm=stc
-freorder-blocks-and-partition -freorder-functions
-frerun-cse-after-loop
-fschedule-insns -fschedule-insns2
-fsched-interblock -fsched-spec
-fspeculatively-call-stored-functions
-fstore-merging
-fstrict-aliasing
-fthread-jumps
-ftree-builtin-call-dce
-ftree-loop-vectorize
-ftree-pre
-ftree-slp-vectorize
-ftree-switch-conversion -ftree-tail-merge
-ftree-vrp
-fvect-cost-model=very-cheap

```

Please note the warning under `-fgcse` about invoking `-O2` on programs that use computed gotos.

- O3** Optimize yet more. `-O3` turns on all optimizations specified by `-O2` and also turns on the following optimization flags:

```

-fgcse-after-reload
-fipa-cp-clone
-floop-interchange
-floop-unroll-and-jam
-fpeel-loops
-fpredictive-commoning
-fsplit-loops
-fsplit-paths
-ftree-loop-distribution
-ftree-partial-pre
-funswitch-loops
-fvect-cost-model=dynamic
-fversion-loops-for-strides

```

- O0** Reduce compilation time and make debugging produce the expected results. This is the default.

At `-O0`, GCC completely disables most optimization passes; they are not run even if you explicitly enable them on the command line, or are listed by `-Q --help=optimizers` as being enabled by default. Many optimizations performed by GCC depend on code analysis or canonicalization passes that are enabled by `-O`, and it would not be useful to run individual optimization passes in isolation.

- Os** Optimize for size. `-Os` enables all `-O2` optimizations except those that often increase code size:

```

-falign-functions -falign-jumps
-falign-labels -falign-loops
-fprefetch-loop-arrays -freorder-blocks-algorithm=stc

```

It also enables `-finline-functions`, causes the compiler to tune for code size rather than execution speed, and performs further optimizations designed to reduce code size.

- Ofast** Disregard strict standards compliance. `-Ofast` enables all `-O3` optimizations. It also enables optimizations that are not valid for all standard-compliant programs. It turns on `-ffast-math`, `-fallow-store-data-races` and the Fortran-specific `-fstack-arrays`, unless `-fmax-stack-var-size` is specified, and `-fno-protect-parens`. It turns off `-fsemantic-interposition`.
- Og** Optimize while keeping in mind debugging experience. `-Og` should be the optimization level of choice for the standard edit-compile-debug cycle, offering a reasonable blend of optimization, fast compilation and debugging experience especially for code with a high abstraction penalty. In contrast to `-O0`, this enables `-fvar-tracking-assignments` and `-fvar-tracking` which handle debug information in the prologue and epilogue of functions better than `-O0`.
 Like `-O0`, `-Og` completely skips a number of optimization passes so that individual options controlling them have no effect. Otherwise `-Og` enables all `-O1` optimization flags except for those known to greatly interfere with debugging:
 - `-fbranch-count-reg` `-fdelayed-branch`
 - `-fdse` `-fif-conversion` `-fif-conversion2`
 - `-finline-functions-called-once`
 - `-fmove-loop-invariants` `-fmove-loop-stores` `-fssa-phiopt`
 - `-ftree-bit-ccp` `-ftree-dse` `-ftree-pta` `-ftree-sra`
- Oz** Optimize aggressively for size rather than speed. This may increase the number of instructions executed if those instructions require fewer bytes to encode. `-Oz` behaves similarly to `-Os` including enabling most `-O2` optimizations.

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

Options of the form `-fflag` specify machine-independent flags. Most flags have both positive and negative forms; the negative form of `-ffoo` is `-fno-foo`. In the table below, only one of the forms is listed—the one you typically use. You can figure out the other form by either removing ‘no-’ or adding it.

The following options control specific optimizations. They are either activated by `-O` options or are related to ones that are. You can use the following flags in the rare cases when “fine-tuning” of optimizations to be performed is desired.

-fno-defer-pop

For machines that must pop arguments after a function call, always pop the arguments as soon as each function returns. At levels `-O1` and higher, `-fdefer-pop` is the default; this allows the compiler to let arguments accumulate on the stack for several function calls and pop them all at once.

-fforward-propagate

Perform a forward propagation pass on RTL. The pass tries to combine two instructions and checks if the result can be simplified. If loop unrolling is active, two passes are performed and the second is scheduled after loop unrolling.

This option is enabled by default at optimization levels `-O1`, `-O2`, `-O3`, `-Os`.

-favoid-store-forwarding**-fno-avoid-store-forwarding**

Many CPUs will stall for many cycles when a load partially depends on previous smaller stores. This pass tries to detect such cases and avoid the penalty by changing the order of the load and store and then fixing up the loaded value.

Disabled by default.

-ffp-contract=style

-ffp-contract=off disables floating-point expression contraction. **-ffp-contract=fast** enables floating-point expression contraction such as forming of fused multiply-add operations if the target has native support for them. **-ffp-contract=on** enables floating-point expression contraction if allowed by the language standard. This is implemented for C and C++, where it enables contraction within one expression, but not across different statements.

The default is **-ffp-contract=off** for C in a standards compliant mode (**-std=c11** or similar), **-ffp-contract=fast** otherwise.

-ffp-int-builtin-inexact

Allow the built-in functions **ceil**, **floor**, **round** and **trunc**, and their **float** and **long double** variants, to generate code that raises the “inexact” floating-point exception for noninteger arguments. ISO C99 and C11 allow these functions to raise the “inexact” exception, but ISO/IEC TS 18661-1:2014, the C bindings to IEEE 754-2008, as integrated into ISO C23, does not allow these functions to do so.

The default is **-fno-fp-int-builtin-inexact**, disallowing the exception to be raised, unless C17 or an earlier C standard is selected. This option does nothing unless **-ftrapping-math** is in effect.

Even if **-fno-fp-int-builtin-inexact** is used, if the functions generate a call to a library function then the “inexact” exception may be raised if the library implementation does not follow TS 18661.

-fomit-frame-pointer

Omit the frame pointer in functions that don’t need one. This avoids the instructions to save, set up and restore the frame pointer; on many targets it also makes an extra register available.

On some targets this flag has no effect because the standard calling sequence always uses a frame pointer, so it cannot be omitted.

Note that **-fno-omit-frame-pointer** doesn’t guarantee the frame pointer is used in all functions. Several targets always omit the frame pointer in leaf functions.

Enabled by default at **-O1** and higher.

-foptimize-crc

Detect loops calculating CRC (performing polynomial long division) and replace them with a faster implementation. Detect 8, 16, 32, and 64 bit CRC, with a constant polynomial without the leading 1 bit, for both bit-forward and bit-reversed cases. If the target supports a CRC instruction and the polynomial used in the source code matches the polynomial used in the CRC instruction,

generate that CRC instruction. Otherwise, if the target supports a carry-less-multiplication instruction, generate CRC using it; otherwise generate table-based CRC.

Enabled by default at `-O2` and higher.

`-foptimize-sibling-calls`

Optimize sibling and tail recursive calls.

Enabled at levels `-O2`, `-O3`, `-Os`.

`-foptimize-strlen`

Optimize various standard C string functions (e.g. `strlen`, `strchr` or `strcpy`) and their `_FORTIFY_SOURCE` counterparts into faster alternatives.

Enabled at levels `-O2`, `-O3`.

`-finline-atomics`

`-fno-inline-atomics`

Inline ‘`__atomic`’ operations when a lock-free instruction sequence is available. This optimization is enabled by default.

`-finline-stringops[=fn]`

Expand memory and string operations (for now, only `memset`) inline, even when the length is variable or big enough as to require looping. This is most useful along with `-ffreestanding` and `-fno-builtin`.

In some circumstances, it enables the compiler to generate code that takes advantage of known alignment and length multipliers, but even then it may be less efficient than optimized runtime implementations, and grow code size so much that even a less performant but shared implementation runs faster due to better use of code caches. This option is disabled by default.

`-fno-inline`

Do not expand any functions inline apart from those marked with the `always_inline` attribute. This is the default when not optimizing.

Single functions can be exempted from inlining by marking them with the `noinline` attribute.

`-finline-small-functions`

Integrate functions into their callers when their body is smaller than expected function call code (so overall size of program gets smaller). The compiler heuristically decides which functions are simple enough to be worth integrating in this way. This inlining applies to all functions, even those not declared inline.

Enabled at levels `-O2`, `-O3`, `-Os`.

`-findirect-inlining`

Inline also indirect calls that are discovered to be known at compile time thanks to previous inlining. This option has any effect only when inlining itself is turned on by the `-finline-functions` or `-finline-small-functions` options.

Enabled at levels `-O2`, `-O3`, `-Os`.

`-finline-functions`

Consider all functions for inlining, even if they are not declared inline. The compiler heuristically decides which functions are worth integrating in this way.

If all calls to a given function are integrated, and the function is declared **static**, then the function is normally not output as assembler code in its own right.

Enabled at levels **-O2**, **-O3**, **-Os**. Also enabled by **-fprofile-use** and **-fauto-profile**.

-finline-functions-called-once

Consider all **static** functions called once for inlining into their caller even if they are not marked **inline**. If a call to a given function is integrated, then the function is not output as assembler code in its own right.

Enabled at levels **-O1**, **-O2**, **-O3** and **-Os**, but not **-Og**.

-fearly-inlining

Inline functions marked by **always_inline** and functions whose body seems smaller than the function call overhead early before doing **-fprofile-generate** instrumentation and real inlining pass. Doing so makes profiling significantly cheaper and usually inlining faster on programs having large chains of nested wrapper functions.

Enabled by default.

-fipa-sra

Perform interprocedural scalar replacement of aggregates, removal of unused parameters and replacement of parameters passed by reference by parameters passed by value.

Enabled at levels **-O2**, **-O3** and **-Os**.

-finline-limit=n

By default, GCC limits the size of functions that can be inlined. This flag allows coarse control of this limit. *n* is the size of functions that can be inlined in number of pseudo instructions.

Inlining is actually controlled by a number of internal parameters, which are documented in the GCC Internals manual and should not normally be set directly.

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 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 6.4.1 [Common Attributes], page 595.

-fkeep-inline-functions

In C, emit **static** functions that are declared **inline** into the object file, even if the function has been inlined into all of its callers. This switch does not affect

functions using the `extern inline` extension in GNU C90. In C++, emit any and all inline functions into the object file.

-fkeep-static-functions

Emit `static` functions into the object file, even if the function is never used.

-fkeep-static-consts

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

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

-fmerge-constants

Attempt to merge identical constants (string constants and floating-point constants) across compilation units.

This option is the default for optimized compilation if the assembler and linker support it. Use `-fno-merge-constants` to inhibit this behavior.

Enabled at levels `-O1`, `-O2`, `-O3`, `-Os`.

-fmerge-all-constants

Attempt to merge identical constants and identical variables.

This option implies `-fmerge-constants`. In addition to `-fmerge-constants` this considers e.g. even constant initialized arrays or initialized constant variables with integral or floating-point types. Languages like C or C++ require each variable, including multiple instances of the same variable in recursive calls, to have distinct locations, so using this option results in non-conforming behavior.

-fmodulo-sched

Perform swing modulo scheduling immediately before the first scheduling pass. This pass looks at innermost loops and reorders their instructions by overlapping different iterations.

-fmodulo-sched-allow-regmoves

Perform more aggressive SMS-based modulo scheduling with register moves allowed. By setting this flag certain anti-dependences edges are deleted, which triggers the generation of reg-moves based on the life-range analysis. This option is effective only with `-fmodulo-sched` enabled.

-fno-branch-count-reg

Disable the optimization pass that scans for opportunities to use “decrement and branch” instructions on a count register instead of instruction sequences that decrement a register, compare it against zero, and then branch based upon the result. This option is only meaningful on architectures that support such instructions, which include x86, PowerPC, IA-64 and S/390. Note that the `-fno-branch-count-reg` option doesn't remove the decrement and branch instructions from the generated instruction stream introduced by other optimization passes.

The default is `-fbranch-count-reg` at `-O1` and higher, except for `-Og`.

-fno-function-cse

Do not put function addresses in registers; make each instruction that calls a constant function contain the function's address explicitly.

This option results in less efficient code, but some strange hacks that alter the assembler output may be confused by the optimizations performed when this option is not used.

The default is **-ffunction-cse**.

-ffuse-ops-with-volatile-access

Allow limited optimization of operations with volatile memory access when doing so does not change the semantics outlined in See Section 6.10 [When is a Volatile Object Accessed?], page 720.

The default is **-ffuse-ops-with-volatile-access**

-fno-zero-initialized-in-bss

If the target supports a BSS section, GCC by default puts variables that are initialized to zero into BSS. This can save space in the resulting code.

This option turns off this behavior because some programs explicitly rely on variables going to the data section—e.g., so that the resulting executable can find the beginning of that section and/or make assumptions based on that.

The default is **-fzero-initialized-in-bss** except in Ada.

-fthread-jumps

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

Enabled at levels **-O1**, **-O2**, **-O3**, **-Os**.

-fsplit-wide-types

When using a type that occupies multiple registers, such as **long long** on a 32-bit system, split the registers apart and allocate them independently. This normally generates better code for those types, but may make debugging more difficult.

Enabled at levels **-O1**, **-O2**, **-O3**, **-Os**.

-fsplit-wide-types-early

Fully split wide types early, instead of very late. This option has no effect unless **-fsplit-wide-types** is turned on.

This is the default on some targets.

-fcse-follow-jumps

In common subexpression elimination (CSE), scan through jump instructions when the target of the jump is not reached by any other path. For example, when CSE encounters an **if** statement with an **else** clause, CSE follows the jump when the condition tested is false.

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

-fcse-skip-blocks

This is similar to **-fcse-follow-jumps**, but causes CSE to follow jumps that conditionally skip over blocks. When CSE encounters a simple **if** statement with no else clause, **-fcse-skip-blocks** causes CSE to follow the jump around the body of the **if**.

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

-frerun-cse-after-loop

Re-run common subexpression elimination after loop optimizations are performed.

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

-fgcse

Perform a global common subexpression elimination pass. This pass also performs global constant and copy propagation.

Note: When compiling a program using computed gotos, a GCC extension, you may get better run-time performance if you disable the global common subexpression elimination pass by adding **-fno-gcse** to the command line.

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

-fgcse-lm

When **-fgcse-lm** is enabled, global common subexpression elimination attempts to move loads that are only killed by stores into themselves. This allows a loop containing a load/store sequence to be changed to a load outside the loop, and a copy/store within the loop.

Enabled by default when **-fgcse** is enabled.

-fgcse-sm

When **-fgcse-sm** is enabled, a store motion pass is run after global common subexpression elimination. This pass attempts to move stores out of loops. When used in conjunction with **-fgcse-lm**, loops containing a load/store sequence can be changed to a load before the loop and a store after the loop.

Not enabled at any optimization level.

-fgcse-las

When **-fgcse-las** is enabled, the global common subexpression elimination pass eliminates redundant loads that come after stores to the same memory location (both partial and full redundancies).

Not enabled at any optimization level.

-fgcse-after-reload

When **-fgcse-after-reload** is enabled, a redundant load elimination pass is performed after reload. The purpose of this pass is to clean up redundant spilling.

Enabled by **-O3**, **-fprofile-use** and **-fauto-profile**.

-faggressive-loop-optimizations

This option tells the loop optimizer to use language constraints to derive bounds for the number of iterations of a loop. This assumes that loop code does not invoke undefined behavior by for example causing signed integer overflows or

out-of-bound array accesses. The bounds for the number of iterations of a loop are used to guide loop unrolling and peeling and loop exit test optimizations. This option is enabled by default.

-funconstrained-commons

This option tells the compiler that variables declared in common blocks (e.g. Fortran) may later be overridden with longer trailing arrays. This prevents certain optimizations that depend on knowing the array bounds.

-fcrossjumping

Perform cross-jumping transformation. This transformation unifies equivalent code and saves code size. The resulting code may or may not perform better than without cross-jumping.

Enabled at levels `-O2`, `-O3`, `-Os`.

-fauto-inc-dec

Combine increments or decrements of addresses with memory accesses. This pass is always skipped on architectures that do not have instructions to support this. Enabled by default at `-O1` and higher on architectures that support this.

-fdce Perform dead code elimination (DCE) on RTL. Enabled by default at `-O1` and higher.

-fdse Perform dead store elimination (DSE) on RTL. Enabled by default at `-O1` and higher.

-fif-conversion

Attempt to transform conditional jumps into branch-less equivalents. This includes use of conditional moves, min, max, set flags and abs instructions, and some tricks doable by standard arithmetics. The use of conditional execution on chips where it is available is controlled by `-fif-conversion2`.

Enabled at levels `-O1`, `-O2`, `-O3`, `-Os`, but not with `-Og`.

-fif-conversion2

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

Enabled at levels `-O1`, `-O2`, `-O3`, `-Os`, but not with `-Og`.

-fdeclone-ctor-dtor

The C++ ABI requires multiple entry points for constructors and destructors: one for a base subobject, one for a complete object, and one for a virtual destructor that calls operator delete afterwards. For a hierarchy with virtual bases, the base and complete variants are clones, which means two copies of the function. With this option, the base and complete variants are changed to be thunks that call a common implementation.

Enabled by `-Os`.

-fdelete-null-pointer-checks

Assume that programs cannot safely dereference null pointers, and that no code or data element resides at address zero. This option enables simple constant folding optimizations at all optimization levels. In addition, other optimization

passes in GCC use this flag to control global dataflow analyses that eliminate useless checks for null pointers; these assume that a memory access to address zero always results in a trap, so that if a pointer is checked after it has already been dereferenced, it cannot be null.

Note however that in some environments this assumption is not true. Use **-fno-delete-null-pointer-checks** to disable this optimization for programs that depend on that behavior.

This option is enabled by default on most targets. On AVR and MSP430, this option is completely disabled.

Passes that use the dataflow information are enabled independently at different optimization levels.

-fdevirtualize

Attempt to convert calls to virtual functions to direct calls. This is done both within a procedure and interprocedurally as part of indirect inlining (**-findirect-inlining**) and interprocedural constant propagation (**-fipa-cp**). Enabled at levels **-O2**, **-O3**, **-Os**.

-fdevirtualize-speculatively

Attempt to convert calls to virtual functions to speculative direct calls. Based on the analysis of the type inheritance graph, determine for a given call the set of likely targets. If the set is small, preferably of size 1, change the call into a conditional deciding between direct and indirect calls. The speculative calls enable more optimizations, such as inlining. When they seem useless after further optimization, they are converted back into original form.

-fdevirtualize-at-ltrans

Stream extra information needed for aggressive devirtualization when running the link-time optimizer in local transformation mode. This option enables more devirtualization but significantly increases the size of streamed data. For this reason it is disabled by default.

-fexpensive-optimizations

Perform a number of minor optimizations that are relatively expensive.

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

-fext-dce

-fno-ext-dce

Perform dead code elimination on zero and sign extensions, with special dataflow analysis.

-free

Attempt to remove redundant extension instructions. This is especially helpful for the x86-64 architecture, which implicitly zero-extends in 64-bit registers after writing to their lower 32-bit half.

Enabled for Alpha, AArch64, LoongArch, PowerPC, RISC-V, SPARC, h83000 and x86 at levels **-O2**, **-O3**, **-Os**.

-fno-lifetime-dse

In C++ the value of an object is only affected by changes within its lifetime: when the constructor begins, the object has an indeterminate value, and any

changes during the lifetime of the object are dead when the object is destroyed. Normally dead store elimination will take advantage of this; if your code relies on the value of the object storage persisting beyond the lifetime of the object, you can use this flag to disable this optimization. To preserve stores before the constructor starts (e.g. because your operator new clears the object storage) but still treat the object as dead after the destructor, you can use `-flifetime-dse=1`. The default behavior can be explicitly selected with `-flifetime-dse=2`. `-flifetime-dse=0` is equivalent to `-fno-lifetime-dse`.

`-flive-range-shrinkage`

Attempt to decrease register pressure through register live range shrinkage. This is helpful for fast processors with small or moderate size register sets.

`-fira-algorithm=algorithm`

Use the specified coloring algorithm for the integrated register allocator. The *algorithm* argument can be ‘priority’, which specifies Chow’s priority coloring, or ‘CB’, which specifies Chaitin-Briggs coloring. Chaitin-Briggs coloring is not implemented for all architectures, but for those targets that do support it, it is the default because it generates better code.

`-fira-region=region`

Use specified regions for the integrated register allocator. The *region* argument should be one of the following:

- ‘all’ Use all loops as register allocation regions. This can give the best results for machines with a small and/or irregular register set.
- ‘mixed’ Use all loops except for loops with small register pressure as the regions. This value usually gives the best results in most cases and for most architectures, and is enabled by default when compiling with optimization for speed (`-O`, `-O2`, ...).
- ‘one’ Use all functions as a single region. This typically results in the smallest code size, and is enabled by default for `-Os` or `-O0`.

`-fira-hoist-pressure`

Use IRA to evaluate register pressure in the code hoisting pass for decisions to hoist expressions. This option usually results in smaller code, but it can slow the compiler down.

This option is enabled at level `-Os` for all targets.

`-fira-loop-pressure`

Use IRA to evaluate register pressure in loops for decisions to move loop invariants. This option usually results in generation of faster and smaller code on machines with large register files (≥ 32 registers), but it can slow the compiler down.

This option is enabled at level `-O3` for some targets.

`-fno-ira-share-save-slots`

Disable sharing of stack slots used for saving call-used hard registers living through a call. Each hard register gets a separate stack slot, and as a result function stack frames are larger.

-fno-ira-share-spill-slots

Disable sharing of stack slots allocated for pseudo-registers. Each pseudo-register that does not get a hard register gets a separate stack slot, and as a result function stack frames are larger.

-flra-remat

Enable CFG-sensitive rematerialization in LRA. Instead of loading values of spilled pseudos, LRA tries to rematerialize (recalculate) values if it is profitable.

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

-fdelayed-branch

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

Enabled at levels **-O1**, **-O2**, **-O3**, **-Os**, but not at **-Og**.

-fschedule-insns

If supported for the target machine, attempt to reorder instructions to eliminate execution stalls due to required data being unavailable. This helps machines that have slow floating point or memory load instructions by allowing other instructions to be issued until the result of the load or floating-point instruction is required.

Conventionally enabled at optimization levels **-O2** and **-O3**. However, many targets override this behavior. For example, on x86, it is disabled at all levels, while on AArch64, it is enabled only at **-O3**.

-fschedule-insns2

Similar to **-fschedule-insns**, but requests an additional pass of instruction scheduling after register allocation has been done. This is especially useful on machines with a relatively small number of registers and where memory load instructions take more than one cycle.

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

-fno-sched-interblock

Disable instruction scheduling across basic blocks, which is normally enabled when scheduling before register allocation, i.e. with **-fschedule-insns** or at **-O2** or higher.

-fno-sched-spec

Disable speculative motion of non-load instructions, which is normally enabled when scheduling before register allocation, i.e. with **-fschedule-insns** or at **-O2** or higher.

-fsched-pressure

Enable register pressure sensitive insn scheduling before register allocation. This only makes sense when scheduling before register allocation is enabled, i.e. with **-fschedule-insns** or at **-O2** or higher. Usage of this option can improve the generated code and decrease its size by preventing register pressure increase above the number of available hard registers and subsequent spills in register allocation.

-fsched-spec-load

Allow speculative motion of some load instructions. This only makes sense when scheduling before register allocation, i.e. with **-fschedule-insns** or at **-O2** or higher.

-fsched-spec-load-dangerous

Allow speculative motion of more load instructions. This only makes sense when scheduling before register allocation, i.e. with **-fschedule-insns** or at **-O2** or higher.

-fsched-stalled-insns**-fsched-stalled-insns=*n***

Define how many insns (if any) can be moved prematurely from the queue of stalled insns into the ready list during the second scheduling pass. **-fno-sched-stalled-insns** means that no insns are moved prematurely, **-fsched-stalled-insns=0** means there is no limit on how many queued insns can be moved prematurely. **-fsched-stalled-insns** without a value is equivalent to **-fsched-stalled-insns=1**.

-fsched-stalled-insns-dep**-fsched-stalled-insns-dep=*n***

Define how many insn groups (cycles) are examined for a dependency on a stalled insn that is a candidate for premature removal from the queue of stalled insns. This has an effect only during the second scheduling pass, and only if **-fsched-stalled-insns** is used. **-fno-sched-stalled-insns-dep** is equivalent to **-fsched-stalled-insns-dep=0**. **-fsched-stalled-insns-dep** without a value is equivalent to **-fsched-stalled-insns-dep=1**.

-fsched2-use-superblocks

When scheduling after register allocation, use superblock scheduling. This allows motion across basic block boundaries, resulting in faster schedules. This option is experimental, as not all machine descriptions used by GCC model the CPU closely enough to avoid unreliable results from the algorithm.

This only makes sense when scheduling after register allocation, i.e. with **-fschedule-insns2** or at **-O2** or higher.

-fsched-group-heuristic

Enable the group heuristic in the scheduler. This heuristic favors the instruction that belongs to a schedule group. This is enabled by default when scheduling is enabled, i.e. with **-fschedule-insns** or **-fschedule-insns2** or at **-O2** or higher.

-fsched-critical-path-heuristic

Enable the critical-path heuristic in the scheduler. This heuristic favors instructions on the critical path. This is enabled by default when scheduling is enabled, i.e. with **-fschedule-insns** or **-fschedule-insns2** or at **-O2** or higher.

-fsched-spec-insn-heuristic

Enable the speculative instruction heuristic in the scheduler. This heuristic favors speculative instructions with greater dependency weakness. This is en-

abled 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.

`-fspeculatively-call-stored-functions`

Attempt to convert indirect calls of function pointers to pointers loaded from a structure field if all visible stores to that field store just a single candidate. When doing so, turn the call into a conditional deciding between the direct call and the original indirect one. These speculative calls often enable more optimizations, such as inlining. When they seem useless after further optimization, they are converted back into original form.

`-freschedule-modulo-scheduled-loops`

Modulo scheduling is performed before traditional scheduling. If a loop is modulo scheduled, later scheduling passes may change its schedule. Use this option to control that behavior.

`-fselective-scheduling`

Schedule instructions using selective scheduling algorithm. Selective scheduling runs instead of the first scheduler pass.

`-fselective-scheduling2`

Schedule instructions using selective scheduling algorithm. Selective scheduling runs instead of the second scheduler pass.

`-fsl-sched-pipelining`

Enable software pipelining of innermost loops during selective scheduling. This option has no effect unless one of `-fselective-scheduling` or `-fselective-scheduling2` is turned on.

`-fsl-sched-pipelining-outer-loops`

When pipelining loops during selective scheduling, also pipeline outer loops. This option has no effect unless `-fsl-sched-pipelining` is turned on.

-fsemantic-interposition

Some object formats, like ELF, allow interposing of symbols by the dynamic linker. This means that for symbols exported from the DSO, the compiler cannot perform interprocedural propagation, inlining and other optimizations in anticipation that the function or variable in question may change. While this feature is useful, for example, to rewrite memory allocation functions by a debugging implementation, it is expensive in the terms of code quality. With **-fno-semantic-interposition** the compiler assumes that if interposition happens for functions the overwriting function will have precisely the same semantics (and side effects). Similarly if interposition happens for variables, the constructor of the variable will be the same. The flag has no effect for functions explicitly declared inline (where it is never allowed for interposition to change semantics) and for symbols explicitly declared weak.

-fshrink-wrap

Emit function prologues only before parts of the function that need it, rather than at the top of the function. This flag is enabled by default at **-O** and higher.

-fshrink-wrap-separate

Shrink-wrap separate parts of the prologue and epilogue separately, so that those parts are only executed when needed. This option is on by default, but has no effect unless **-fshrink-wrap** is also turned on and the target supports this.

-fcaller-saves

Enable allocation of values to registers that are clobbered by function calls, by emitting extra instructions to save and restore the registers around such calls. Such allocation is done only when it seems to result in better code.

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

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

-fcombine-stack-adjustments

Tracks stack adjustments (pushes and pops) and stack memory references and then tries to find ways to combine them.

Enabled by default at **-O1** and higher.

-fipa-ra Use caller save registers for allocation if those registers are not used by any called function. In that case it is not necessary to save and restore them around calls. This is only possible if called functions are part of same compilation unit as current function and they are compiled before it.

Enabled at levels **-O2**, **-O3**, **-Os**, however the option is disabled if generated code will be instrumented for profiling (**-p**, or **-pg**) or if callee's register usage cannot be known exactly (this happens on targets that do not expose prologues and epilogues in RTL).

-fconserve-stack

Attempt to minimize stack usage. The compiler attempts to use less stack space, even if that makes the program slower. This option implies setting the

`large-stack-frame` parameter to 100 and the `large-stack-frame-growth` parameter to 400.

-ftree-reassoc

Perform reassociation on trees. This flag is enabled by default at `-O1` and higher.

-fcode-hoisting

Perform code hoisting. Code hoisting tries to move the evaluation of expressions executed on all paths to the function exit as early as possible. This is especially useful as a code size optimization, but it often helps for code speed as well. This flag is enabled by default at `-O2` and higher.

-ftree-pre

Perform partial redundancy elimination (PRE) on trees. This flag is enabled by default at `-O2` and `-O3`.

-ftree-partial-pre

Make partial redundancy elimination (PRE) more aggressive. This flag is enabled by default at `-O3`.

-ftree-forwprop

Perform forward propagation on trees. This flag is enabled by default at `-O1` and higher.

-ftree-fre

Perform full redundancy elimination (FRE) on trees. The difference between FRE and PRE is that FRE only considers expressions that are computed on all paths leading to the redundant computation. This analysis is faster than PRE, though it exposes fewer redundancies. This flag is enabled by default at `-O1` and higher.

-ftree-phi-prop

Perform hoisting of loads from conditional pointers on trees. This pass is enabled by default at `-O1` and higher.

-fhoist-adjacent-loads

Speculatively hoist loads from both branches of an if-then-else if the loads are from adjacent locations in the same structure and the target architecture has a conditional move instruction. This flag is enabled by default at `-O2` and higher.

-ftree-copy-prop

Perform copy propagation on trees. This pass eliminates unnecessary copy operations. This flag is enabled by default at `-O1` and higher.

-fipa-pure-const

Discover which functions are pure or constant. Enabled by default at `-O1` and higher.

-fipa-reference

Discover which static variables do not escape the compilation unit. Enabled by default at `-O1` and higher.

-fipa-reference-addressable

Discover read-only, write-only and non-addressable static variables. Enabled by default at -O1 and higher.

-fipa-reorder-for-locality

Group call chains close together in the binary layout to improve code locality and minimize jump distances between frequently called functions. Unlike **-freorder-functions** this pass considers the call chains between functions and groups them together, rather than grouping all hot/normal/cold/never-executed functions into separate sections. Unlike **-fprofile-reorder-functions** it aims to improve code locality throughout the runtime of the program rather than focusing on program startup. This option is incompatible with an explicit **-flto-partition=** option since it enforces a custom partitioning scheme. If using this option it is recommended to also use profile feedback, but this option is not enabled by default otherwise.

-fipa-stack-alignment

Reduce stack alignment on call sites if possible. Enabled by default.

-fipa-pta

Perform interprocedural pointer analysis and interprocedural modification and reference analysis. This option can cause excessive memory and compile-time usage on large compilation units. It is not enabled by default at any optimization level.

-fipa-profile

Perform interprocedural profile propagation. The functions called only from cold functions are marked as cold. Also functions executed once (such as **cold**, **noreturn**, static constructors or destructors) are identified. Cold functions and loop less parts of functions executed once are then optimized for size. Enabled by default at -O1 and higher.

-fipa-modref

Perform interprocedural mod/ref analysis. This optimization analyzes the side effects of functions (memory locations that are modified or referenced) and enables better optimization across the function call boundary. This flag is enabled by default at -O1 and higher.

-fipa-cp

Perform interprocedural constant propagation. This optimization analyzes the program to determine when values passed to functions are constants and then optimizes accordingly. This optimization can substantially increase performance if the application has constants passed to functions. This flag is enabled by default at -O2, -Os and -O3. It is also enabled by **-fprofile-use** and **-fauto-profile**.

-fipa-cp-clone

Perform function cloning to make interprocedural constant propagation stronger. When enabled, interprocedural constant propagation performs function cloning when externally visible function can be called with constant arguments. Because this optimization can create multiple copies of functions,

it may significantly increase code size. This flag is enabled by default at `-O3`. It is also enabled by `-fprofile-use` and `-fauto-profile`.

`-fipa-bit-cp`

When enabled, perform interprocedural bitwise constant propagation. This flag is enabled by default at `-O2` and by `-fprofile-use` and `-fauto-profile`. It requires that `-fipa-cp` is enabled.

`-fipa-vrp`

When enabled, perform interprocedural propagation of value ranges. This flag is enabled by default at `-O2`. It requires that `-fipa-cp` is enabled.

`-fipa-icf-functions`

`-fipa-icf-variables`

`-fipa-icf`

Perform Identical Code Folding for functions (`-fipa-icf-functions`), read-only variables (`-fipa-icf-variables`), or both (`-fipa-icf`). The optimization reduces code size and may disturb unwind stacks by replacing a function by an equivalent one with a different name. The optimization works more effectively with link-time optimization enabled.

Although the behavior is similar to the Gold Linker's ICF optimization, GCC ICF works on different levels and thus the optimizations are not same - there are equivalences that are found only by GCC and equivalences found only by Gold.

`-fipa-icf` is enabled by default at `-O2` and `-Os`.

`-flate-combine-instructions`

Enable two instruction combination passes that run relatively late in the compilation process. One of the passes runs before register allocation and the other after register allocation. The main aim of the passes is to substitute definitions into all uses.

Most targets enable this flag by default at `-O2` and `-Os`.

`-flive-patching=level`

Control GCC's optimizations to produce output suitable for live-patching.

If the compiler's optimization uses a function's body or information extracted from its body to optimize/change another function, the latter is called an impacted function of the former. If a function is patched, its impacted functions should be patched too.

The impacted functions are determined by the compiler's interprocedural optimizations. For example, a caller is impacted when inlining a function into its caller, cloning a function and changing its caller to call this new clone, or extracting a function's pureness/constness information to optimize its direct or indirect callers, etc.

Usually, the more IPA optimizations enabled, the larger the number of impacted functions for each function. In order to control the number of impacted functions and more easily compute the list of impacted function, IPA optimizations can be partially enabled at two different levels.

The *level* argument should be one of the following:

‘inline-clone’

Only enable inlining and cloning optimizations, which includes inlining, cloning, interprocedural scalar replacement of aggregates and partial inlining. As a result, when patching a function, all its callers and its clones’ callers are impacted, therefore need to be patched as well.

-flive-patching=inline-clone disables the following optimization flags:

```
-fwhole-program -fipa-pta -fipa-reference -fipa-ra
-fipa-icf -fipa-icf-functions -fipa-icf-variables
-fipa-bit-cp -fipa-vrp -fipa-pure-const
-fipa-reference-addressable
-fipa-stack-alignment -fipa-modref
```

‘inline-only-static’

Only enable inlining of static functions. As a result, when patching a static function, all its callers are impacted and so need to be patched as well.

In addition to all the flags that **-flive-patching=inline-clone** disables, **-flive-patching=inline-only-static** disables the following additional optimization flags:

```
-fipa-cp-clone -fipa-sra -fpartial-inlining -fipa-cp
```

When **-flive-patching** is specified without any value, the default value is *inline-clone*.

This flag is disabled by default.

Note that **-flive-patching** is not supported with link-time optimization (**-flto**).

-fisolate-erroneous-paths-dereference

Detect paths that trigger erroneous or undefined behavior due to dereferencing a null pointer (with **-fdelete-null-pointer-checks** enabled) or a division by zero. Isolate those paths from the main control flow and turn the statement with erroneous or undefined behavior into a trap. This flag is enabled by default at **-O2** and higher.

-fisolate-erroneous-paths-attribute

Detect paths that trigger erroneous or undefined behavior due to a null value being used in a way forbidden by a **returns_nonnull** or **nonnull** attribute. Isolate those paths from the main control flow and turn the statement with erroneous or undefined behavior into a trap. This is not currently enabled, but may be enabled by **-O2** in the future.

-ftree-sink

Perform forward store motion on trees. This flag is enabled by default at **-O1** and higher.

-ftree-bit-ccp

Perform sparse conditional bit constant propagation on trees and propagate pointer alignment information. This pass only operates on local scalar variables

and is enabled by default at `-O1` and higher, except for `-Og`. It requires that `-ftree-ccp` is enabled.

-ftree-ccp

Perform sparse conditional constant propagation (CCP) on trees. This pass only operates on local scalar variables and is enabled by default at `-O1` and higher.

-fssa-backprop

Propagate information about uses of a value up the definition chain in order to simplify the definitions. For example, this pass strips sign operations if the sign of a value never matters. The flag is enabled by default at `-O1` and higher.

-fssa-phiopt

Perform pattern matching on SSA PHI nodes to optimize conditional code. This pass is enabled by default at `-O1` and higher, except for `-Og`.

-ftree-switch-conversion

Perform conversion of simple initializations in a switch to initializations from a scalar array. This flag is enabled by default at `-O2` and higher.

-ftree-tail-merge

Look for identical code sequences. When found, replace one with a jump to the other. This optimization is known as tail merging or cross jumping. This flag is enabled by default at `-O2` and higher. The compilation time in this pass can be limited using `max-tail-merge-comparisons` parameter and `max-tail-merge-iterations` parameter.

-ftree-cselim

Perform conditional store elimination on trees. This flag is enabled by default at `-O1` and higher on targets that have conditional move instructions.

-ftree-dce

Perform dead code elimination (DCE) on trees. This flag is enabled by default at `-O1` and higher.

-ftree-builtin-call-dce

Perform conditional dead code elimination (DCE) for calls to built-in functions that may set `errno` but are otherwise free of side effects. This flag is enabled by default at `-O2` and higher if `-Os` is not also specified.

-ffinite-loops

Assume that a loop with an exit will eventually take the exit and not loop indefinitely. This allows the compiler to remove loops that otherwise have no side-effects, not considering eventual endless looping as such.

This option is enabled by default at `-O2` for C++ with `-std=c++11` or higher.

-ftree-dominator-opts

Perform a variety of simple scalar cleanups (constant/copy propagation, redundancy elimination, range propagation and expression simplification) based on a dominator tree traversal. This also performs jump threading (to reduce jumps to jumps). This flag is enabled by default at `-O1` and higher.

-ftree-dse

Perform dead store elimination (DSE) on trees. A dead store is a store into a memory location that is later overwritten by another store without any intervening loads. In this case the earlier store can be deleted. This flag is enabled by default at **-O1** and higher.

-ftree-ch

Perform loop header copying on trees. This is beneficial since it increases effectiveness of code motion optimizations. It also saves one jump. This flag is enabled by default at **-O1** and higher. It is not enabled for **-Os**, since it usually increases code size.

-ftree-loop-optimize

Perform loop optimizations on trees. This flag is enabled by default at **-O1** and higher.

-ftree-loop-linear**-floop-strip-mine****-floop-block**

Perform loop nest optimizations. Same as **-floop-nest-optimize**. To use this code transformation, GCC has to be configured with **--with-isl** to enable the Graphite loop transformation infrastructure.

-fgraphite-identity

Enable the identity transformation for graphite. For every SCoP we generate the polyhedral representation and transform it back to gimple. Using **-fgraphite-identity** we can check the costs or benefits of the GIMPLE -> GRAPHITE -> GIMPLE transformation. Some minimal optimizations are also performed by the code generator isl, like index splitting and dead code elimination in loops.

-floop-nest-optimize

Enable the isl based loop nest optimizer. This is a generic loop nest optimizer based on the Pluto optimization algorithms. It calculates a loop structure optimized for data-locality and parallelism. This option is experimental.

-floop-parallelize-all

Use the Graphite data dependence analysis to identify loops that can be parallelized. Parallelize all the loops that can be analyzed to not contain loop carried dependences without checking that it is profitable to parallelize the loops.

-ftree-coalesce-vars

While transforming the program out of the SSA representation, attempt to reduce copying by coalescing versions of different user-defined variables, instead of just compiler temporaries. This may severely limit the ability to debug an optimized program compiled with **-fno-var-tracking-assignments**. In the negated form, this flag prevents SSA coalescing of user variables. This option is enabled by default if optimization is enabled, and it does very little otherwise.

-ftree-loop-if-convert

Attempt to transform conditional jumps in the innermost loops to branch-less equivalents. The intent is to remove control-flow from the innermost loops in

order to improve the ability of the vectorization pass to handle these loops. This is enabled by default if vectorization is enabled.

-ftree-loop-distribution

Perform loop distribution. This flag can improve cache performance on big loop bodies and allow further loop optimizations, like parallelization or vectorization, to take place. For example, the loop

```
DO I = 1, N
  A(I) = B(I) + C
  D(I) = E(I) * F
ENDDO
```

is transformed to

```
DO I = 1, N
  A(I) = B(I) + C
ENDDO
DO I = 1, N
  D(I) = E(I) * F
ENDDO
```

This flag is enabled by default at `-O3`. It is also enabled by `-fprofile-use` and `-fauto-profile`.

-ftree-loop-distribute-patterns

Perform loop distribution of patterns that can be code generated with calls to a library. This flag is enabled by default at `-O2` and higher, and by `-fprofile-use` and `-fauto-profile`.

This pass distributes the initialization loops and generates a call to `memset` zero. For example, the loop

```
DO I = 1, N
  A(I) = 0
  B(I) = A(I) + I
ENDDO
```

is transformed to

```
DO I = 1, N
  A(I) = 0
ENDDO
DO I = 1, N
  B(I) = A(I) + I
ENDDO
```

and the initialization loop is transformed into a call to `memset` zero.

-floop-interchange

Perform loop interchange outside of `graphite`. This flag can improve cache performance on loop nest and allow further loop optimizations, like vectorization, to take place. For example, the loop

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

is transformed to

```
for (int i = 0; i < N; i++)
  for (int k = 0; k < N; k++)
```

```

    for (int j = 0; j < N; j++)
        c[i][j] = c[i][j] + a[i][k]*b[k][j];

```

This flag is enabled by default at `-O3`. It is also enabled by `-fprofile-use` and `-fauto-profile`.

`-floop-unroll-and-jam`

Apply unroll and jam transformations on feasible loops. In a loop nest this unrolls the outer loop by some factor and fuses the resulting multiple inner loops. This flag is enabled by default at `-O3`. It is also enabled by `-fprofile-use` and `-fauto-profile`.

`-ftree-loop-im`

Perform loop invariant motion on trees. This pass moves only invariants that are hard to handle at RTL level (function calls, operations that expand to nontrivial sequences of insns). With `-funswitch-loops` it also moves operands of conditions that are invariant out of the loop, so that we can use just trivial invariantness analysis in loop unswitching. The pass also includes store motion.

`-ftree-loop-ivcanon`

Create a canonical counter for number of iterations in loops for which determining number of iterations requires complicated analysis. Later optimizations then may determine the number easily. Useful especially in connection with unrolling.

`-ftree-scev-cprop`

Perform final value replacement. If a variable is modified in a loop in such a way that its value when exiting the loop can be determined using only its initial value and the number of loop iterations, replace uses of the final value by such a computation, provided it is sufficiently cheap. This reduces data dependencies and may allow further simplifications. Enabled by default at `-O1` and higher.

`-fivopts` Perform induction variable optimizations (strength reduction, induction variable merging and induction variable elimination) on trees. Enabled by default at `-O1` and higher.

`-ftree-parallelize-loops`

`-ftree-parallelize-loops=n`

Parallelize loops, i.e., split their iteration space to run in multiple 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`.

When a positive value *n* is specified, the number of threads is fixed at compile time and cannot be changed after compilation. The compiler generates `"#pragma omp parallel num_threads(n)"`.

When used without `=n` (i.e., `-ftree-parallelize-loops`), the number of threads is determined at program execution time via the `OMP_NUM_THREADS` environment variable. If `OMP_NUM_THREADS` is not set, the OpenMP runtime automatically detects the number of available processors and uses that value.

This enables creating binaries that adapt to different hardware configurations without recompilation.

-ftree-pta

Perform function-local points-to analysis on trees. This flag is enabled by default at **-O1** and higher, except for **-Og**.

-ftree-sra

Perform scalar replacement of aggregates. This pass replaces structure references with scalars to prevent committing structures to memory too early. This flag is enabled by default at **-O1** and higher, except for **-Og**.

-fstore-merging

Perform merging of narrow stores to consecutive memory addresses. This pass merges contiguous stores of immediate values narrower than a word into fewer wider stores to reduce the number of instructions. This is enabled by default at **-O2** and higher as well as **-Os**.

-ftree-ter

Perform temporary expression replacement during the SSA->normal phase. Single use/single def temporaries are replaced at their use location with their defining expression. This results in non-GIMPLE code, but gives the expanders much more complex trees to work on resulting in better RTL generation. This is enabled by default at **-O1** and higher.

-ftree-slsr

Perform straight-line strength reduction on trees. This recognizes related expressions involving multiplications and replaces them by less expensive calculations when possible. This is enabled by default at **-O1** and higher.

-ftree-vectorize

Perform vectorization on trees. This flag enables **-ftree-loop-vectorize** and **-ftree-slp-vectorize** if not explicitly specified.

-ftree-loop-vectorize

Perform loop vectorization on trees. This flag is enabled by default at **-O2** and by **-ftree-vectorize**, **-fprofile-use**, and **-fauto-profile**.

-ftree-slp-vectorize

Perform basic block vectorization on trees. This flag is enabled by default at **-O2** and by **-ftree-vectorize**, **-fprofile-use**, and **-fauto-profile**.

-ftrivial-auto-var-init=choice

Initialize automatic variables or temporary objects with either a pattern or with zeroes to increase the security and predictability of a program by preventing uninitialized memory disclosure and use. GCC still considers an automatic variable that doesn't have an explicit initializer as uninitialized, **-Wuninitialized** and **-Wanalyzer-use-of-uninitialized-value** will still report warning messages on such automatic variables or temporary objects and the compiler will perform optimization as if the variable were uninitialized. With this option, GCC will also initialize any padding of automatic variables or temporary objects that have structure or union types to zeroes. However, the current implementation cannot initialize automatic variables whose initialization is bypassed

through `switch` or `goto` statement. Using `-Wtrivial-auto-var-init` to report all such cases.

The three values of *choice* are:

- ‘**uninitialized**’ doesn’t initialize any automatic variables.
- ‘**pattern**’ Initialize automatic variables with values which will likely transform logic bugs into crashes down the line, are easily recognized in a crash dump and without being values that programmers can rely on for useful program semantics. The current value is byte-repeatable pattern with byte “0xFE”. The values used for pattern initialization might be changed in the future.
- ‘**zero**’ Initialize automatic variables with zeroes.

The default is ‘**uninitialized**’ except for C++26, in which case if `-ftrivial-auto-var-init=` is not specified at all automatic variables or temporary objects are zero initialized, but zero initialization of padding bits does not happen.

Note that the initializer values, whether ‘**zero**’ or ‘**pattern**’, refer to data representation (in memory or machine registers), rather than to their interpretation as numerical values. This distinction may be important in languages that support types with biases or implicit multipliers, and with such extensions as ‘**hardbool**’ (see Section 6.4.1 [Common Attributes], page 595). For example, a variable that uses 8 bits to represent (biased) quantities in the range 160..400 will be initialized with the bit patterns 0x00 or 0xFE, depending on *choice*, whether or not these representations stand for values in that range, and even if they do, the interpretation of the value held by the variable will depend on the bias. A ‘**hardbool**’ variable that uses say 0x5A and 0xA5 for **false** and **true**, respectively, will trap with either ‘*choice*’ of trivial initializer, i.e., ‘**zero**’ initialization will not convert to the representation for **false**, even if it would for a **static** variable of the same type. This means the initializer pattern doesn’t generally depend on the type of the initialized variable. One notable exception is that (non-hardened) boolean variables that fit in registers are initialized with **false** (zero), even when ‘**pattern**’ is requested.

You can control this behavior for a specific variable by using the variable attribute **uninitialized** standard attribute (see Section 6.4.1 [Common Attributes], page 595) or the C++26 `[[indeterminate]]`.

`-fvect-cost-model=model`

Alter the cost model used for vectorization. The *model* argument should be one of ‘**unlimited**’, ‘**dynamic**’, ‘**cheap**’ or ‘**very-cheap**’. With the ‘**unlimited**’ model the vectorized code-path is assumed to be profitable while with the ‘**dynamic**’ model a runtime check guards the vectorized code-path to enable it only for iteration counts that will likely execute faster than when executing the original scalar loop. The ‘**cheap**’ model disables vectorization of loops where doing so would be cost prohibitive for example due to required runtime checks for data dependence or alignment but otherwise is equal to the ‘**dynamic**’ model. The ‘**very-cheap**’ model disables vectorization of loops when any runtime check for data dependence or alignment is required, it also disables vectorization of epilogue loops but otherwise is equal to the ‘**cheap**’ model.

The default cost model depends on other optimization flags and is either ‘dynamic’ or ‘cheap’.

-fsimd-cost-model=*model*

Alter the cost model used for vectorization of loops marked with the OpenMP `simd` directive. The *model* argument should be one of ‘unlimited’, ‘dynamic’, ‘cheap’. All values of *model* have the same meaning as described in `-fvect-cost-model` and by default a cost model defined with `-fvect-cost-model` is used.

-ftree-vrp

Perform Value Range Propagation on trees. This is similar to the constant propagation pass, but instead of values, ranges of values are propagated. This allows the optimizers to remove unnecessary range checks like array bound checks and null pointer checks. This is enabled by default at `-O2` and higher. Null pointer check elimination is only done if `-fdelete-null-pointer-checks` is enabled.

-fsplit-paths

Split paths leading to loop backedges. This can improve dead code elimination and common subexpression elimination. This is enabled by default at `-O3` and above.

-fsplit-ivs-in-unroller

Enables expression of values of induction variables in later iterations of the unrolled loop using the value in the first iteration. This breaks long dependency chains, thus improving efficiency of the scheduling passes.

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

This optimization is enabled by default.

-fvariable-expansion-in-unroller

With this option, the compiler creates multiple copies of some local variables when unrolling a loop, which can result in superior code.

This optimization is enabled by default for PowerPC targets, but disabled by default otherwise.

-fpartial-inlining

Inline parts of functions. This option has any effect only when inlining itself is turned on by the `-finline-functions` or `-finline-small-functions` options.

Enabled at levels `-O2`, `-O3`, `-Os`.

-fpredictive-commoning

Perform predictive commoning optimization, i.e., reusing computations (especially memory loads and stores) performed in previous iterations of loops.

This option is enabled at level `-O3`. It is also enabled by `-fprofile-use` and `-fauto-profile`.

-fprefetch-loop-arrays

If supported by the target machine, generate instructions to prefetch memory to improve the performance of loops that access large arrays.

This option may generate better or worse code; results are highly dependent on the structure of loops within the source code.

Disabled at level **-Os**.

-fno-printf-return-value

Do not substitute constants for known return value of formatted output functions such as `sprintf`, `snprintf`, `vsprintf`, and `vsnprintf` (but not `printf` or `fprintf`). This transformation allows GCC to optimize or even eliminate branches based on the known return value of these functions called with arguments that are either constant, or whose values are known to be in a range that makes determining the exact return value possible. For example, when **-fprintf-return-value** is in effect, both the branch and the body of the `if` statement (but not the call to `snprintf`) can be optimized away when `i` is a 32-bit or smaller integer because the return value is guaranteed to be at most 8.

```
char buf[9];
if (snprintf (buf, "%08x", i) >= sizeof buf)
    ...
```

The **-fprintf-return-value** option relies on other optimizations and yields best results with **-O2** and above. It works in tandem with the **-Wformat-overflow** and **-Wformat-truncation** options. The **-fprintf-return-value** option is enabled by default.

-fno-peephole**-fno-peephole2**

Disable any machine-specific peephole optimizations. The difference between **-fno-peephole** and **-fno-peephole2** is in how they are implemented in the compiler; some targets use one, some use the other, a few use both.

-fpeephole is enabled by default. **-fpeephole2** enabled at levels **-O2**, **-O3**, **-Os**.

-fno-guess-branch-probability

Do not guess branch probabilities using heuristics.

GCC uses heuristics to guess branch probabilities if they are not provided by profiling feedback (**-fprofile-arcs**). These heuristics are based on the control flow graph. If some branch probabilities are specified by `__builtin_expect`, then the heuristics are used to guess branch probabilities for the rest of the control flow graph, taking the `__builtin_expect` info into account. The interactions between the heuristics and `__builtin_expect` can be complex, and in some cases, it may be useful to disable the heuristics so that the effects of `__builtin_expect` are easier to understand.

It is also possible to specify expected probability of the expression with `__builtin_expect_with_probability` built-in function.

The default is **-fguess-branch-probability** at levels **-O**, **-O2**, **-O3**, **-Os**.

-freorder-blocks

Reorder basic blocks in the compiled function in order to reduce number of taken branches and improve code locality.

Enabled at levels **-O1**, **-O2**, **-O3**, **-Os**.

-freorder-blocks-algorithm=algorithm

Use the specified algorithm for basic block reordering. The *algorithm* argument can be **'simple'**, which does not increase code size (except sometimes due to secondary effects like alignment), or **'stc'**, the “software trace cache” algorithm, which tries to put all often executed code together, minimizing the number of branches executed by making extra copies of code.

The default is **'simple'** at levels **-O1**, **-Os**, and **'stc'** at levels **-O2**, **-O3**.

-freorder-blocks-and-partition

In addition to reordering basic blocks in the compiled function, in order to reduce number of taken branches, partitions hot and cold basic blocks into separate sections of the assembly and **.o** files, to improve paging and cache locality performance.

This optimization is automatically turned off in the presence of exception handling or unwind tables (on targets using setjump/longjump or target specific scheme), for linkonce sections, for functions with a user-defined section attribute and on any architecture that does not support named sections. When **-fsplit-stack** is used this option is not enabled by default (to avoid linker errors), but may be enabled explicitly (if using a working linker).

Enabled for x86 at levels **-O2**, **-O3**, **-Os**.

-freorder-functions

Reorder functions in the object file in order to improve code locality. Unlike **-fipa-reorder-for-locality** this option prioritises grouping all functions within a category (hot/normal/cold/never-executed) together. This is implemented by using special subsections **.text.hot** for most frequently executed functions and **.text.unlikely** for unlikely executed functions. Reordering is done by the linker so object file format must support named sections and linker must place them in a reasonable way.

This option isn't effective unless you either provide profile feedback (see **-fprofile-arcs** for details) or manually annotate functions with **hot** or **cold** attributes (see Section 6.4.1 [Common Attributes], page 595).

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, accessing an object of one type via an expression of a different type is not allowed, unless the types are *compatible types*, differ only in signedness or qualifiers, or the expression has a character type. Accessing scalar objects via a corresponding vector type is also allowed.

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 in C, provided the memory is accessed through the union type. In ISO C++, type-punning through a union type is undefined behavior, but GCC supports it as an extension. So, the code above works as expected. See Section 4.10 [Structures unions enumerations and bit-fields implementation], page 568. However, this code might not:

```
int f() {
    union a_union t;
    int* ip;
    t.d = 3.0;
    ip = &t.i;
    return *ip;
}
```

Similarly, access by taking the address, casting the resulting pointer and dereferencing the result has undefined behavior, even if the cast uses a union type, e.g.:

```
int f() {
    double d = 3.0;
    return ((union a_union *) &d)->i;
}
```

The `-fstrict-aliasing` option is enabled at levels `-O2`, `-O3`, `-Os`.

`-fipa-strict-aliasing`

Controls whether rules of `-fstrict-aliasing` are applied across function boundaries. Note that if multiple functions gets inlined into a single function the memory accesses are no longer considered to be crossing a function boundary.

The `-fipa-strict-aliasing` option is enabled by default and is effective only in combination with `-fstrict-aliasing`.

`-falign-functions`

`-falign-functions=n`

`-falign-functions=n:m`

`-falign-functions=n:m:n2`

`-falign-functions=n:m:n2:m2`

Align the start of functions to the next power-of-two greater than or equal to n , skipping up to $m-1$ bytes. This ensures that at least the first m bytes of the function can be fetched by the CPU without crossing an n -byte alignment boundary. This is an optimization of code performance and alignment is ignored

for functions considered cold. If alignment is required for all functions, use `-fmin-function-alignment`.

If m is not specified, it defaults to n .

Examples: `-falign-functions=32` aligns functions to the next 32-byte boundary, `-falign-functions=24` aligns to the next 32-byte boundary only if this can be done by skipping 23 bytes or less, `-falign-functions=32:7` aligns to the next 32-byte boundary only if this can be done by skipping 6 bytes or less.

The second pair of $n2:m2$ values allows you to specify a secondary alignment: `-falign-functions=64:7:32:3` aligns to the next 64-byte boundary if this can be done by skipping 6 bytes or less, otherwise aligns to the next 32-byte boundary if this can be done by skipping 2 bytes or less. If $m2$ is not specified, it defaults to $n2$.

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

`-fno-align-functions` and `-falign-functions=1` are equivalent and mean that functions are not aligned.

If n is not specified or is zero, use a machine-dependent default. The maximum allowed n option value is 65536.

Enabled at levels `-O2`, `-O3`.

`-flimit-function-alignment`

If this option is enabled, the compiler tries to avoid unnecessarily overaligning functions. It attempts to instruct the assembler to align by the amount specified by `-falign-functions`, but not to skip more bytes than the size of the function.

`-falign-labels`

`-falign-labels=n`

`-falign-labels=n:m`

`-falign-labels=n:m:n2`

`-falign-labels=n:m:n2:m2`

Align all branch targets to a power-of-two boundary.

Parameters of this option are analogous to the `-falign-functions` option. `-fno-align-labels` and `-falign-labels=1` are equivalent and mean that labels are not aligned.

If `-falign-loops` or `-falign-jumps` are applicable and are greater than this value, then their values are used instead.

If n is not specified or is zero, use a machine-dependent default which is very likely to be '1', meaning no alignment. The maximum allowed n option value is 65536.

Enabled at levels `-O2`, `-O3`.

`-falign-loops`
`-falign-loops=n`
`-falign-loops=n:m`
`-falign-loops=n:m:n2`
`-falign-loops=n:m:n2:m2`

Align loops to a power-of-two boundary. If the loops are executed many times, this makes up for any execution of the dummy padding instructions. This is an optimization of code performance and alignment is ignored for loops considered cold.

If `-falign-labels` is greater than this value, then its value is used instead.

Parameters of this option are analogous to the `-falign-functions` option. `-fno-align-loops` and `-falign-loops=1` are equivalent and mean that loops are not aligned. The maximum allowed *n* option value is 65536.

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

Enabled at levels `-O2`, `-O3`.

`-falign-jumps`
`-falign-jumps=n`
`-falign-jumps=n:m`
`-falign-jumps=n:m:n2`
`-falign-jumps=n:m:n2:m2`

Align branch targets to a power-of-two boundary, for branch targets where the targets can only be reached by jumping. In this case, no dummy operations need be executed. This is an optimization of code performance and alignment is ignored for jumps considered cold.

If `-falign-labels` is greater than this value, then its value is used instead.

Parameters of this option are analogous to the `-falign-functions` option. `-fno-align-jumps` and `-falign-jumps=1` are equivalent and mean that loops are not aligned.

If *n* is not specified or is zero, use a machine-dependent default. The maximum allowed *n* option value is 65536.

Enabled at levels `-O2`, `-O3`.

`-fmin-function-alignment`

Specify minimal alignment of functions to the next power-of-two greater than or equal to *n*. Unlike `-falign-functions` this alignment is applied also to all functions (even those considered cold). The alignment is also not affected by `-flimit-function-alignment`

`-fno-allocation-dce`

Do not remove unused C++ allocations (using operator `new` and operator `delete`) in dead code elimination.

See also `-fmalloc-dce`.

`-fallow-store-data-races`

Allow the compiler to perform optimizations that may introduce new data races on stores, without proving that the variable cannot be concurrently accessed by

other threads. Does not affect optimization of local data. It is safe to use this option if it is known that global data will not be accessed by multiple threads. Examples of optimizations enabled by `-fallow-store-data-races` include hoisting or if-conversions that may cause a value that was already in memory to be re-written with that same value. Such re-writing is safe in a single threaded context but may be unsafe in a multi-threaded context. Note that on some processors, if-conversions may be required in order to enable vectorization.

Enabled at level `-Ofast`.

`-funit-at-a-time`

This option is left for compatibility reasons. `-funit-at-a-time` has no effect, while `-fno-unit-at-a-time` implies `-fno-toplevel-reorder` and `-fno-section-anchors`.

Enabled by default.

`-fno-toplevel-reorder`

Do not reorder top-level functions, variables, and `asm` statements. Output them in the same order that they appear in the input file. When this option is used, unreferenced static variables are not removed. This option is intended to support existing code that relies on a particular ordering. For new code, it is better to use attributes when possible.

`-ftoplevel-reorder` is the default at `-O1` and higher, and also at `-O0` if `-fsection-anchors` is explicitly requested. Additionally `-fno-toplevel-reorder` implies `-fno-section-anchors`.

`-funreachable-traps`

With this option, the compiler turns calls to `__builtin_unreachable` into traps, instead of using them for optimization. This also affects any such calls implicitly generated by the compiler.

This option has the same effect as `-fsanitize=unreachable -fsanitize-trap=unreachable`, but does not affect the values of those options. If `-fsanitize=unreachable` is enabled, that option takes priority over this one.

This option is enabled by default at `-O0` and `-Og`.

`-fweb`

Constructs webs as commonly used for register allocation purposes and assign each web individual pseudo register. This allows the register allocation pass to operate on pseudos directly, but also strengthens several other optimization passes, such as CSE, loop optimizer and trivial dead code remover. It can, however, make debugging impossible, since variables no longer stay in a “home register”.

Enabled by default with `-funroll-loops`.

`-fwhole-program`

Assume that the current compilation unit represents the whole program being compiled. All public functions and variables with the exception of `main` and those merged by attribute `externally_visible` become static functions and in effect are optimized more aggressively by interprocedural optimizers.

With `-flto` this option has a limited use. In most cases the precise list of symbols used or exported from the binary is known the resolution info passed to the link-time optimizer by the linker plugin. It is still useful if no linker plugin is used or during incremental link step when final code is produced (with `-flto -flinker-output=nolto-rel`).

`-flto[=n]`

This option runs the standard link-time optimizer. When invoked with source code, it generates GIMPLE (one of GCC's internal representations) and writes it to special ELF sections in the object file. When the object files are linked together, all the function bodies are read from these ELF sections and instantiated as if they had been part of the same translation unit.

To use the link-time optimizer, `-flto` and optimization options should be specified at compile time and during the final link. It is recommended that you compile all the files participating in the same link with the same options and also specify those options at link time. For example:

```
gcc -c -O2 -flto foo.c
gcc -c -O2 -flto bar.c
gcc -o myprog -flto -O2 foo.o bar.o
```

The first two invocations to GCC save a bytecode representation of GIMPLE into special ELF sections inside `foo.o` and `bar.o`. The final invocation reads the GIMPLE bytecode from `foo.o` and `bar.o`, merges the two files into a single internal image, and compiles the result as usual. Since both `foo.o` and `bar.o` are merged into a single image, this causes all the interprocedural analyses and optimizations in GCC to work across the two files as if they were a single one. This means, for example, that the inliner is able to inline functions in `bar.o` into functions in `foo.o` and vice-versa.

Another (simpler) way to enable link-time optimization is:

```
gcc -o myprog -flto -O2 foo.c bar.c
```

The above generates bytecode for `foo.c` and `bar.c`, merges them together into a single GIMPLE representation and optimizes them as usual to produce `myprog`.

The important thing to keep in mind is that to enable link-time optimizations you need to use the GCC driver to perform the link step. GCC automatically performs link-time optimization if any of the objects involved were compiled with the `-flto` command-line option. You can always override the automatic decision to do link-time optimization by passing `-fno-lto` to the link command.

To make whole-program optimization effective, it is necessary to make certain assumptions. The compiler needs to know what functions and variables can be accessed by libraries and runtime outside of the link-time optimized unit. When supported by the linker, the linker plugin (see `-fuse-linker-plugin`) passes information to the compiler about used and externally visible symbols. When the linker plugin is not available, `-fwhole-program` should be used to allow the compiler to make these assumptions, which leads to more aggressive optimization decisions.

When a file is compiled with `-flto` without `-fuse-linker-plugin`, the generated object file is larger than a regular object file because it contains GIMPLE

bytecodes and the usual final code (see `-ffat-lto-objects`). This means that object files with LTO information can be linked as normal object files; if `-fno-lto` is passed to the linker, no interprocedural optimizations are applied. Note that when `-fno-fat-lto-objects` is enabled the compile stage is faster but you cannot perform a regular, non-LTO link on them.

When producing the final binary, GCC only applies link-time optimizations to those files that contain bytecode. Therefore, you can mix and match object files and libraries with GIMPLE bytecodes and final object code. GCC automatically selects which files to optimize in LTO mode and which files to link without further processing.

Generally, options specified at link time override those specified at compile time, although in some cases GCC attempts to infer link-time options from the settings used to compile the input files.

If you do not specify an optimization level option `-O` at link time, then GCC uses the highest optimization level used when compiling the object files. Note that it is generally ineffective to specify an optimization level option only at link time and not at compile time, for two reasons. First, compiling without optimization suppresses compiler passes that gather information needed for effective optimization at link time. Second, some early optimization passes can be performed only at compile time and not at link time.

There are some code generation flags preserved by GCC when generating bytecodes, as they need to be used during the final link. Currently, the following options and their settings are taken from the first object file that explicitly specifies them: `-fcommon`, `-fexceptions`, `-fnon-call-exceptions`, `-fgnu-tm` and all the `-m` target flags.

The following options `-fPIC`, `-fpic`, `-fpie` and `-fPIE` are combined based on the following scheme:

```
-fPIC + -fpic = -fpic
-fPIC + -fno-pic = -fno-pic
-fpic/-fPIC + (no option) = (no option)
-fPIC + -fPIE = -fPIE
-fpic + -fPIE = -fpie
-fPIC/-fpic + -fpie = -fpie
```

Certain ABI-changing flags are required to match in all compilation units, and trying to override this at link time with a conflicting value is ignored. This includes options such as `-freg-struct-return` and `-fpcc-struct-return`.

Other options such as `-ffp-contract`, `-fno-strict-overflow`, `-fwrapv`, `-fno-trapv` or `-fno-strict-aliasing` are passed through to the link stage and merged conservatively for conflicting translation units. Specifically `-fno-strict-overflow`, `-fwrapv` and `-fno-trapv` take precedence; and for example `-ffp-contract=off` takes precedence over `-ffp-contract=fast`. You can override them at link time.

Diagnostic options such as `-Wstringop-overflow` are passed through to the link stage and their setting matches that of the compile-step at function granularity. Note that this matters only for diagnostics emitted during optimization.

Note that code transforms such as inlining can lead to warnings being enabled or disabled for regions if code not consistent with the setting at compile time.

When you need to pass options to the assembler via `-Wa` or `-Xassembler` make sure to either compile such translation units with `-fno-lto` or consistently use the same assembler options on all translation units. You can alternatively also specify assembler options at LTO link time.

To enable debug info generation you need to supply `-g` at compile time. If any of the input files at link time were built with debug info generation enabled the link will enable debug info generation as well. Any elaborate debug info settings like the dwarf level `-gdwarf-5` need to be explicitly repeated at the linker command line and mixing different settings in different translation units is discouraged.

If LTO encounters objects with C linkage declared with incompatible types in separate translation units to be linked together (undefined behavior according to ISO C99 6.2.7), a non-fatal diagnostic may be issued. The behavior is still undefined at run time. Similar diagnostics may be raised for other languages.

Another feature of LTO is that it is possible to apply interprocedural optimizations on files written in different languages:

```
gcc -c -flto foo.c
g++ -c -flto bar.cc
gfortran -c -flto baz.f90
g++ -o myprog -flto -O3 foo.o bar.o baz.o -lgfortran
```

Notice that the final link is done with `g++` to get the C++ runtime libraries and `-lgfortran` is added to get the Fortran runtime libraries. In general, when mixing languages in LTO mode, you should use the same link command options as when mixing languages in a regular (non-LTO) compilation.

If object files containing GIMPLE bytecode are stored in a library archive, say `libfoo.a`, it is possible to extract and use them in an LTO link if you are using a linker with plugin support. To create static libraries suitable for LTO, use `gcc-ar` and `gcc-ranlib` instead of `ar` and `ranlib`; to show the symbols of object files with GIMPLE bytecode, use `gcc-nm`. Those commands require that `ar`, `ranlib` and `nm` have been compiled with plugin support. At link time, use the flag `-fuse-linker-plugin` to ensure that the library participates in the LTO optimization process:

```
gcc -o myprog -O2 -flto -fuse-linker-plugin a.o b.o -lfoo
```

With the linker plugin enabled, the linker extracts the needed GIMPLE files from `libfoo.a` and passes them on to the running GCC to make them part of the aggregated GIMPLE image to be optimized.

If you are not using a linker with plugin support and/or do not enable the linker plugin, then the objects inside `libfoo.a` are extracted and linked as usual, but they do not participate in the LTO optimization process. In order to make a static library suitable for both LTO optimization and usual linkage, compile its object files with `-flto -ffat-lto-objects`.

Link-time optimizations do not require the presence of the whole program to operate. If the program does not require any symbols to be exported, it is possible to combine `-flto` and `-fwhole-program` to allow the interprocedural

optimizers to use more aggressive assumptions which may lead to improved optimization opportunities. Use of `-fwhole-program` is not needed when linker plugin is active (see `-fuse-linker-plugin`).

The current implementation of LTO makes no attempt to generate bytecode that is portable between different types of hosts. The bytecode files are versioned and there is a strict version check, so bytecode files generated in one version of GCC do not work with an older or newer version of GCC.

Link-time optimization does not work well with generation of debugging information on systems other than those using a combination of ELF and DWARF.

If you specify the optional *n*, the optimization and code generation done at link time is executed in parallel using *n* parallel jobs by utilizing an installed `make` program. The environment variable `MAKE` may be used to override the program used.

You can also specify `-flto=jobserver` to use GNU make's job server mode to determine the number of parallel jobs. This is useful when the Makefile calling GCC is already executing in parallel. You must prepend a '+' to the command recipe in the parent Makefile for this to work. This option likely only works if `MAKE` is GNU make. Even without the option value, GCC tries to automatically detect a running GNU make's job server.

Use `-flto=auto` to use GNU make's job server, if available, or otherwise fall back to autodetection of the number of CPU threads present in your system.

`-flto-partition=alg`

Specify the partitioning algorithm used by the link-time optimizer. The value is either `'1to1'` to specify a partitioning mirroring the original source files or `'balanced'` to specify partitioning into equally sized chunks (whenever possible) or `'max'` to create new partition for every symbol where possible or `'cache'` to balance chunk sizes while keeping related symbols together for better caching in incremental LTO. Specifying `'none'` as an algorithm disables partitioning and streaming completely. The default value is `'balanced'`. While `'1to1'` can be used as a workaround for various code ordering issues, the `'max'` partitioning is intended for internal testing only. The value `'one'` specifies that exactly one partition should be used while the value `'none'` bypasses partitioning and executes the link-time optimization step directly from the WPA phase.

`-flto-incremental=path`

Enable incremental LTO, with its cache in given existing directory. Can significantly shorten edit-compile cycles with LTO.

When used with LTO (`-flto`), the output of translation units inside LTO is cached. Cached translation units are likely to be encountered again when recompiling with small code changes, leading to recompile time reduction.

Multiple GCC instances can use the same cache in parallel.

`-flto-incremental-cache-size=n`

Specifies number of cache entries in incremental LTO after which to prune old entries. This is a soft limit, temporarily there may be more entries.

-flto-compression-level=n

This option specifies the level of compression used for intermediate language written to LTO object files, and is only meaningful in conjunction with LTO mode (**-flto**). GCC currently supports two LTO compression algorithms. For **zstd**, valid values are 0 (no compression) to 19 (maximum compression), while **zlib** supports values from 0 to 9. Values outside this range are clamped to either minimum or maximum of the supported values. If the option is not given, a default balanced compression setting is used.

-flto-toplevel-asm-heuristics

Enables heuristics to find symbols used in top-level basic **asm**. This will restrict link-time optimizations that could cause renaming or deletion of such symbols which would result in missing symbol errors by linker.

This flag is intended for projects that have not converted to using top-level extended **asm** (see Section 6.11.2 [Extended Asm], page 723), which specify the usage directly without any false positives.

The heuristics are simple and do not parse the assembly. The heuristics scan through top-level assembly for all possible identifiers; if an identifier is found among declared symbols, the symbol will be marked to restrict link-time optimizations. Static symbols disable more optimizations. Identifiers followed by **':'** disable more optimizations as well, because they might be a locally defined symbol in assembly, even when the declaration is marked **'extern'**.

-fuse-linker-plugin

Enables the use of a linker plugin during link-time optimization. This option relies on plugin support in the linker, which is available in gold or in GNU ld 2.21 or newer.

This option enables the extraction of object files with GIMPLE bytecode out of library archives. This improves the quality of optimization by exposing more code to the link-time optimizer. This information specifies what symbols can be accessed externally (by non-LTO object or during dynamic linking). Resulting code quality improvements on binaries (and shared libraries that use hidden visibility) are similar to **-fwhole-program**. See **-flto** for a description of the effect of this flag and how to use it.

This option is enabled by default when LTO support in GCC is enabled and GCC was configured for use with a linker supporting plugins (GNU ld 2.21 or newer or gold).

-ffat-lto-objects

Fat LTO objects are object files that contain both the intermediate language and the object code. This makes them usable for both LTO linking and normal linking. This option is effective only when compiling with **-flto** and is ignored at link time.

-fno-fat-lto-objects improves compilation time over plain LTO, but requires the complete toolchain to be aware of LTO. It requires a linker with linker plugin support for basic functionality. Additionally, **nm**, **ar** and **ranlib** need to support linker plugins to allow a full-featured build environment (capable of building static libraries etc). GCC provides the **gcc-ar**, **gcc-nm**, **gcc-ranlib**

wrappers to pass the right options to these tools. With non fat LTO makefiles need to be modified to use them.

Note that modern binutils provide plugin auto-load mechanism. Installing the linker plugin into `$libdir/bfd-plugins` has the same effect as usage of the command wrappers (`gcc-ar`, `gcc-nm` and `gcc-ranlib`).

The default is `-fno-fat-lto-objects` on targets with linker plugin support.

`-fcompare-elim`

After register allocation and post-register allocation instruction splitting, identify arithmetic instructions that compute processor flags similar to a comparison operation based on that arithmetic. If possible, eliminate the explicit comparison operation.

This pass only applies to certain targets that cannot explicitly represent the comparison operation before register allocation is complete.

Enabled at levels `-O1`, `-O2`, `-O3`, `-Os`.

`-ffold-mem-offsets`

`-fno-fold-mem-offsets`

Try to eliminate add instructions by folding them in memory loads/stores.

Enabled at levels `-O2`, `-O3`.

`-fcprop-registers`

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

Enabled at levels `-O1`, `-O2`, `-O3`, `-Os`.

`-fprofile-correction`

Profiles collected using an instrumented binary for multi-threaded programs may be inconsistent due to missed counter updates. When this option is specified, GCC uses heuristics to correct or smooth out such inconsistencies. By default, GCC emits an error message when an inconsistent profile is detected.

This option is enabled by `-fauto-profile`.

`-fprofile-partial-training`

With `-fprofile-use` all portions of programs not executed during training runs are optimized aggressively for size rather than speed. In some cases it is not practical to train all possible hot paths in the program. (For example, it may contain functions specific to a given hardware and training may not cover all hardware configurations the program later runs on.) With `-fprofile-partial-training` profile feedback is ignored for all functions not executed during the training runs, causing them to be optimized as if they were compiled without profile feedback. This leads to better performance when the training is not representative at the cost of significantly bigger code.

`-fprofile-use`

`-fprofile-use=path`

Enable profile feedback-directed optimizations, and the following optimizations, many of which are generally profitable only with profile feedback available:

`-fbranch-probabilities` `-fprofile-values`

```
-funroll-loops -fpeel-loops -ftracer -fvpt
-finline-functions -fipa-cp -fipa-cp-clone -fipa-bit-cp
-fpredictive-commoning -fsplit-loops -funswitch-loops
-fgcse-after-reload -ftree-loop-vectorize -ftree-slp-vectorize
-fvect-cost-model=dynamic -ftree-loop-distribute-patterns
-fprofile-reorder-functions
```

Before you can use this option, you must first generate profiling information. See Section 3.13 [Instrumentation Options], page 245, for information about the `-fprofile-generate` option.

By default, GCC emits an error message if the feedback profiles do not match the source code. This error can be turned into a warning by using `-Wno-error=coverage-mismatch`. Note this may result in poorly optimized code. Additionally, by default, GCC also emits a warning message if the feedback profiles do not exist (see `-Wmissing-profile`).

If *path* is specified, GCC looks at the *path* to find the profile feedback data files. See `-fprofile-dir`.

`-fauto-profile`

`-fauto-profile=path`

Enable sampling-based feedback-directed optimizations, and the following optimizations, many of which are generally profitable only with profile feedback available:

```
-fbranch-probabilities -fprofile-values
-funroll-loops -fpeel-loops -ftracer -fvpt
-finline-functions -fipa-cp -fipa-cp-clone -fipa-bit-cp
-fpredictive-commoning -fsplit-loops -funswitch-loops
-fgcse-after-reload -ftree-loop-vectorize -ftree-slp-vectorize
-fvect-cost-model=dynamic -ftree-loop-distribute-patterns
-fprofile-correction
```

path is the name of a file containing AutoFDO profile information. If omitted, it defaults to `fbdata.afdo` in the current directory.

Producing an AutoFDO profile data file requires running your program with the `perf` utility on a supported GNU/Linux target system. For more information, see <https://perfwiki.github.io/main/>.

E.g.

```
perf record -e br_inst_retired:near_taken -b -o perf.data \
-- your_program
```

Then use the `create_gcov` tool to convert the raw profile data to a format that can be used by GCC. You must also supply the unstripped binary for your program to this tool. See <https://github.com/google/autofdo>.

E.g.

```
create_gcov --binary=your_program.unstripped --profile=perf.data \
--gcov=profile.afdo
```

`-fauto-profile-inlining`

When auto-profile is available inline all relevant functions which was inlined in the tran run before reading the profile feedback. This improves context sensitivity of the profile. Enabled by default.

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

-fexcess-precision=style

This option allows control over excess precision on machines where floating-point operations occur in a format with more precision or range than the IEEE standard and interchange floating-point types. An example of such a target is x87 floating point on x86 processors, which uses an 80-bit representation internally instead of the 64-bit IEEE format. For most programs, the excess precision is harmless, but some programs may rely on the requirements of the C or C++ language standards for handling IEEE values.

By default, **-fexcess-precision=fast** is in effect; this means that operations may be carried out in a wider precision than the types specified in the source if that would result in faster code, and it is unpredictable when rounding to the types specified in the source code takes place. When compiling C or C++, if **-fexcess-precision=standard** is specified then excess precision follows the rules specified in ISO C99 or C++; in particular, both casts and assignments cause values to be rounded to their semantic types (whereas **-ffloat-store** only affects assignments). This option is enabled by default for C or C++ if a strict conformance option such as **-std=c99** or **-std=c++17** is used. **-ffast-math** enables **-fexcess-precision=fast** by default regardless of whether a strict conformance option is used. If **-fexcess-precision=16** is specified, constants and the results of expressions with types `_Float16` and `__bf16` are computed without excess precision.

-fexcess-precision=standard is not implemented for languages other than C or C++. On the x86, it has no effect if **-mfpmath=sse** or **-mfpmath=sse+387** is specified; in the former case, IEEE semantics apply without excess precision, and in the latter, rounding is unpredictable.

-ffloat-store

Do not store floating-point variables in registers, and inhibit other options that might change whether a floating-point value is taken from a register or memory. This option has generally been subsumed by **-fexcess-precision=standard**, which is more general. If you do use **-ffloat-store**, you may need to modify your program to explicitly store intermediate computations in temporary variables since **-ffloat-store** handles rounding to IEEE format only on assignments and not casts as **-fexcess-precision=standard** does.

-ffast-math

Sets the options **-fno-math-errno**, **-funsafe-math-optimizations**, **-ffinite-math-only**, **-fno-rounding-math**, **-fno-signaling-nans**, **-fcx-limited-range** and **-fexcess-precision=fast**.

This option causes the preprocessor macro `__FAST_MATH__` to be defined.

This option is not turned on by any **-O** option besides **-Ofast** since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications.

-fno-math-errno

Do not set `errno` after calling math functions that are executed with a single instruction, e.g., `sqrt`. A program that relies on IEEE exceptions for math error handling may want to use this flag for speed while maintaining IEEE arithmetic compatibility.

This option is not turned on by any `-O` option besides `-Ofast` since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications.

The default is `-fmath-errno`.

On Darwin systems, the math library never sets `errno`. There is therefore no reason for the compiler to consider the possibility that it might, and `-fno-math-errno` is the default.

-funsafe-math-optimizations

Allow optimizations for floating-point arithmetic that (a) assume that arguments and results are valid and (b) may violate IEEE or ANSI standards. When used at link time, it may include libraries or startup files that change the default FPU control word or other similar optimizations.

This option is not turned on by any `-O` option besides `-Ofast` since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications. Enables `-fno-signed-zeros`, `-fno-trapping-math`, `-fassociative-math` and `-freciprocal-math`.

The default is `-fno-unsafe-math-optimizations`.

-fassociative-math

Allow re-association of operands in series of floating-point operations. This violates the ISO C and C++ language standard by possibly changing computation result. NOTE: re-ordering may change the sign of zero as well as ignore NaNs and inhibit or create underflow or overflow (and thus cannot be used on code that relies on rounding behavior like $(x + 2^{52}) - 2^{52}$). May also reorder floating-point comparisons and thus may not be used when ordered comparisons are required. This option requires that both `-fno-signed-zeros` and `-fno-trapping-math` be in effect. Moreover, it doesn't make much sense with `-frounding-math`. For Fortran the option is automatically enabled when both `-fno-signed-zeros` and `-fno-trapping-math` are in effect.

The default is `-fno-associative-math`.

-freciprocal-math

Allow the reciprocal of a value to be used instead of dividing by the value if this enables optimizations. For example x / y can be replaced with $x * (1/y)$, which is useful if $(1/y)$ is subject to common subexpression elimination. Note that this loses precision and increases the number of flops operating on the value.

The default is `-fno-reciprocal-math`.

`-ffinite-math-only`

Allow optimizations for floating-point arithmetic that assume that arguments and results are not NaNs or `+Infs`.

This option is not turned on by any `-O` option besides `-Ofast` since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions. It may, however, yield faster code for programs that do not require the guarantees of these specifications.

The default is `-fno-finite-math-only`.

`-fno-signed-zeros`

Allow optimizations for floating-point arithmetic that ignore the signedness of zero. IEEE arithmetic specifies the behavior of distinct `+0.0` and `-0.0` values, which then prohibits simplification of expressions such as `x+0.0` or `0.0*x` (even with `-ffinite-math-only`). This option implies that the sign of a zero result isn't significant.

The default is `-fsigned-zeros`.

`-fno-trapping-math`

Compile code assuming that floating-point operations cannot generate user-visible traps. These traps include division by zero, overflow, underflow, inexact result and invalid operation. This option requires that `-fno-signaling-nans` be in effect. Setting this option may allow faster code if one relies on “non-stop” IEEE arithmetic, for example.

This option is not turned on by any `-O` option besides `-Ofast` since it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions.

The default is `-ftrapping-math`.

Future versions of GCC may provide finer control of this setting using C99's `FENV_ACCESS` pragma. This command-line option will be used along with `-frounding-math` to specify the default state for `FENV_ACCESS`.

`-frounding-math`

Disable transformations and optimizations that assume default floating-point rounding behavior (round-to-nearest). This option should be specified for programs that change the FP rounding mode dynamically, or that may be executed with a non-default rounding mode. This option disables constant folding of floating-point expressions at compile time (which may be affected by rounding mode) and arithmetic transformations that are unsafe in the presence of sign-dependent rounding modes.

The default is `-fno-rounding-math`.

This option is experimental and does not currently guarantee to disable all GCC optimizations that are affected by rounding mode. Future versions of GCC may provide finer control of this setting using C99's `FENV_ACCESS` pragma. This command-line option will be used along with `-ftrapping-math` to specify the default state for `FENV_ACCESS`.

-fsignaling-nans

Compile code assuming that IEEE signaling NaNs may generate user-visible traps during floating-point operations. Setting this option disables optimizations that may change the number of exceptions visible with signaling NaNs. This option implies **-ftrapping-math**.

This option causes the preprocessor macro `__SUPPORT_SNAN__` to be defined.

The default is **-fno-signaling-nans**.

This option is experimental and does not currently guarantee to disable all GCC optimizations that affect signaling NaN behavior.

-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 option 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.

-fcx-method=*method*

Complex multiplication and division follow the stated *method*. The *method* argument should be one of `'limited-range'`, `'fortran'` or `'stdc'`.

The default is to honor language specific constraints which means `'fortran'` for Fortran and `'stdc'` otherwise.

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 3.13 [Instrumentation Options], page 245), you can compile it a second time using **-fbranch-probabilities**, to improve optimizations based on the number of times each branch was taken. When a program compiled with **-fprofile-arcs** exits, it saves arc execution counts to a file called *sourcename.gcda* for each source file. The information in this data file is very dependent on the structure of the generated code, so you must use the same source code and the same optimization options for both compilations. See details about the file naming in **-fprofile-arcs**.

With `-fbranch-probabilities`, GCC puts a ‘REG_BR_PROB’ note on each ‘JUMP_INSN’ and ‘CALL_INSN’. These can be used to improve optimization. Currently, they are only used in one place: in `reorg.cc`, instead of guessing which path a branch is most likely to take, the ‘REG_BR_PROB’ values are used to exactly determine which path is taken more often.

Enabled by `-fprofile-use` and `-fauto-profile`.

`-fprofile-values`

If combined with `-fprofile-arcs`, it adds code so that some data about values of expressions in the program is gathered.

With `-fbranch-probabilities`, it reads back the data gathered from profiling values of expressions for usage in optimizations.

Enabled by `-fprofile-generate`, `-fprofile-use`, and `-fauto-profile`.

`-fprofile-reorder-functions`

Function reordering based on profile instrumentation collects first time of execution of a function and orders these functions in ascending order, aiming to optimize program startup through more efficient loading of text segments.

Enabled with `-fprofile-use`.

`-fvpt`

If combined with `-fprofile-arcs`, this option instructs the compiler to add code to gather information about values of expressions.

With `-fbranch-probabilities`, it reads back the data gathered and actually performs the optimizations based on them. Currently the optimizations include specialization of division operations using the knowledge about the value of the denominator.

Enabled with `-fprofile-use` and `-fauto-profile`.

`-frename-registers`

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

Enabled by default with `-funroll-loops`.

`-fschedule-fusion`

Performs a target dependent pass over the instruction stream to schedule instructions of same type together because target machine can execute them more efficiently if they are adjacent to each other in the instruction flow.

Enabled at levels `-O2`, `-O3`, `-Os`.

`-fddep-fusion`

Detect macro-op fusible pairs consisting of single-use instructions and their uses, and place such pairs together in the instruction stream to increase fusion opportunities in hardware. This pass is executed once before register allocation, and another time before register renaming.

Enabled at levels `-O2`, `-O3`, `-Os`.

- ftracer** Perform tail duplication to enlarge superblock size. This transformation simplifies the control flow of the function allowing other optimizations to do a better job.
Enabled by **-fprofile-use** and **-fauto-profile**.
- funroll-loops**
Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. **-funroll-loops** implies **-frerun-cse-after-loop**, **-fweb** and **-frename-registers**. It also turns on complete loop peeling (i.e. complete removal of loops with a small constant number of iterations). This option makes code larger, and may or may not make it run faster.
Enabled by **-fprofile-use** and **-fauto-profile**.
- funroll-all-loops**
Unroll all loops, even if their number of iterations is uncertain when the loop is entered. This usually makes programs run more slowly. **-funroll-all-loops** implies the same options as **-funroll-loops**.
- fpeel-loops**
Peels loops for which there is enough information that they do not roll much (from profile feedback or static analysis). It also turns on complete loop peeling (i.e. complete removal of loops with small constant number of iterations).
Enabled by **-O3**, **-fprofile-use**, and **-fauto-profile**.
- fmalloc-dce**
Control whether **malloc** (and its variants such as **calloc** or **strdup**), can be optimized away provided its return value is only used as a parameter of **free** call or compared with **NULL**. If **-fmalloc-dce=1** is used, only calls to **free** are allowed while with **-fmalloc-dce=2** also comparisons with **NULL** pointer are considered safe to remove.
The default is **-fmalloc-dce=2**. See also **-fallocation-dce**.
- fmove-loop-invariants**
Enables the loop invariant motion pass in the RTL loop optimizer. Enabled at level **-O1** and higher, except for **-Og**.
- fmove-loop-stores**
Enables the loop store motion pass in the GIMPLE loop optimizer. This moves invariant stores to after the end of the loop in exchange for carrying the stored value in a register across the iteration. Note for this option to have an effect **-ftree-loop-im** has to be enabled as well. Enabled at level **-O1** and higher, except for **-Og**.
- fsplit-loops**
Split a loop into two if it contains a condition that's always true for one side of the iteration space and false for the other.
Enabled by **-fprofile-use** and **-fauto-profile**.
- funswitch-loops**
Move branches with loop invariant conditions out of the loop, with duplicates of the loop on both branches (modified according to result of the condition).

Enabled by `-fprofile-use` and `-fauto-profile`.

`-fversion-loops-for-strides`

If a loop iterates over an array with a variable stride, create another version of the loop that assumes the stride is always one. For example:

```
for (int i = 0; i < n; ++i)
    x[i * stride] = ...;
```

becomes:

```
if (stride == 1)
    for (int i = 0; i < n; ++i)
        x[i] = ...;
else
    for (int i = 0; i < n; ++i)
        x[i * stride] = ...;
```

This is particularly useful for assumed-shape arrays in Fortran where (for example) it allows better vectorization assuming contiguous accesses. This flag is enabled by default at `-O3`. It is also enabled by `-fprofile-use` and `-fauto-profile`.

`-ffunction-sections`

`-fdata-sections`

Place each function or data item into its own section in the output file if the target supports arbitrary sections. The name of the function or the name of the data item determines the section's name in the output file.

Use these options on systems where the linker can perform optimizations to improve locality of reference in the instruction space. Most systems using the ELF object format have linkers with such optimizations. On AIX, the linker rearranges sections (CSECTs) based on the call graph. The performance impact varies.

Together with a linker garbage collection (linker `--gc-sections` option) these options may lead to smaller statically-linked executables (after stripping).

On ELF/DWARF systems these options do not degenerate the quality of the debug information. There could be issues with other object files/debug info formats.

Only use these options when there are significant benefits from doing so. When you specify these options, the assembler and linker create larger object and executable files and are also slower. These options affect code generation. They prevent optimizations by the compiler and assembler using relative locations inside a translation unit since the locations are unknown until link time. An example of such an optimization is relaxing calls to short call instructions.

`-fstdarg-opt`

Optimize the prologue of variadic argument functions with respect to usage of those arguments.

`-fsection-anchors`

Try to reduce the number of symbolic address calculations by using shared “anchor” symbols to address nearby objects. This transformation can help to reduce the number of GOT entries and GOT accesses on some targets.

For example, the implementation of the following function `foo`:

```
static int a, b, c;
int foo (void) { return a + b + c; }
```

usually calculates the addresses of all three variables, but if you compile it with `-fsection-anchors`, it accesses the variables from a common anchor point instead. The effect is similar to the following pseudocode (which isn't valid C):

```
int foo (void)
{
    register int *xr = &x;
    return xr[&a - &x] + xr[&b - &x] + xr[&c - &x];
}
```

Not all targets support this option.

`-fzero-call-used-regs=choice`

Zero call-used registers at function return to increase program security by either mitigating Return-Oriented Programming (ROP) attacks or preventing information leakage through registers.

The possible values of *choice* are the same as for the `zero_call_used_regs` attribute (see Section 6.4.1 [Common Attributes], page 595). The default is 'skip'.

You can control this behavior for a specific function by using the function attribute `zero_call_used_regs` (see Section 6.4.1 [Common Attributes], page 595).

3.13 Program Instrumentation Options

GCC supports a number of command-line options that control adding run-time instrumentation to the code it normally generates. For example, one purpose of instrumentation is collect profiling statistics for use in finding program hot spots, code coverage analysis, or profile-guided optimizations. Another class of program instrumentation is adding run-time checking to detect programming errors like invalid pointer dereferences or out-of-bounds array accesses, as well as deliberately hostile attacks such as stack smashing or C++ vtable hijacking. There is also a general hook which can be used to implement other forms of tracing or function-level instrumentation for debug or program analysis purposes.

`-p`
`--profile`
`-fprofile`
`-pg`

Generate extra code to write profile information suitable for the analysis program `prof` (for `-p`, `--profile`, and `-fprofile`) or `gprof` (for `-pg`). You must use this option when compiling the source files you want data about, and you must also use it when linking.

You can use the function attribute `no_instrument_function` to suppress profiling of individual functions when compiling with these options. See Section 6.4.1 [Common Attributes], page 595.

`-fprofile-arcs`

Add code so that program flow arcs are instrumented. During execution the program records how many times each branch and call is executed and how

many times it is taken or returns. On targets that support constructors with priority support, profiling properly handles constructors, destructors and C++ constructors (and destructors) of classes which are used as a type of a global variable.

When the compiled program exits it saves this data to a file called **auxname.gcda** for each source file. The data may be used for profile-directed optimizations (**-fbranch-probabilities**), or for test coverage analysis (**-ftest-coverage**). Each object file's *auxname* is generated from the name of the output file, if explicitly specified and it is not the final executable, otherwise it is the basename of the source file. In both cases any suffix is removed (e.g. **foo.gcda** for input file **dir/foo.c**, or **dir/foo.gcda** for output file specified as **-o dir/foo.o**).

Note that if a command line directly links source files, the corresponding *.gcda* files will be prefixed with the unsuffixed name of the output file. E.g. **gcc a.c b.c -o binary** would generate **binary-a.gcda** and **binary-b.gcda** files.

-fcondition-coverage

Add code so that program conditions are instrumented. During execution the program records what terms in a conditional contributes to a decision, which can be used to verify that all terms in a Boolean function are tested and have an independent effect on the outcome of a decision. The result can be read with **gcov --conditions**.

-fpath-coverage

Add code so that the paths taken are tracked. During execution the program records the prime paths taken. The number of paths grows very fast with complexity, and to avoid exploding compile times GCC will give up instrumentation if the approximate number of paths exceeds the limit controlled by **-fpath-coverage-limit**. The result can be read with **gcov --prime-paths --prime-paths-lines --prime-paths-source**, See [gcov prime paths example], page 1081.

-fpath-coverage-limit=limit

The threshold at which point **-fpath-coverage** gives up on instrumenting a function. This limit is approximate and conservative, as GCC uses a pessimistic heuristic which slightly overcounts the running number of paths, and gives up if the threshold is reached before finding all the paths. This option is not for fine grained control over which functions to instrument - rather it is intended to limit the effect of path explosion and keep compile times reasonable. The default is *250000*.

See Section 11.5 [Cross-profiling], page 1085.

--coverage

-coverage

This option is used to compile and link code instrumented for coverage analysis. The options **-coverage** and **--coverage** are equivalent; both are a synonym for **-fprofile-arcs -ftest-coverage** (when compiling) and **-lgcov** (when linking). See the documentation for those options for more details.

- Compile the source files with **-fprofile-arcs** plus optimization and code generation options. For test coverage analysis, use the additional **-ftest-coverage** option. You do not need to profile every source file in a program.
- Compile the source files additionally with **-fprofile-abs-path** to create absolute path names in the **.gcno** files. This allows **gcov** to find the correct sources in projects where compilations occur with different working directories.
- Link your object files with **-lgcov** or **-fprofile-arcs** (the latter implies the former).
- Run the program on a representative workload to generate the arc profile information. This may be repeated any number of times. You can run concurrent instances of your program, and provided that the file system supports locking, the data files will be correctly updated. Unless a strict ISO C dialect option is in effect, **fork** calls are detected and correctly handled without double counting.

Moreover, an object file can be recompiled multiple times and the corresponding **.gcda** file merges as long as the source file and the compiler options are unchanged.

- For profile-directed optimizations, compile the source files again with the same optimization and code generation options plus **-fbranch-probabilities** (see Section 3.12 [Options that Control Optimization], page 196).
- For test coverage analysis, use **gcov** to produce human readable information from the **.gcno** and **.gcda** files. Refer to the **gcov** documentation for further information.

With **-fprofile-arcs**, for each function of your program GCC creates a program flow graph, then finds a spanning tree for the graph. Only arcs that are not on the spanning tree have to be instrumented: the compiler adds code to count the number of times that these arcs are executed. When an arc is the only exit or only entrance to a block, the instrumentation code can be added to the block; otherwise, a new basic block must be created to hold the instrumentation code.

With **-fcondition-coverage**, for each conditional in your program GCC creates a bitset and records the exercised boolean values that have an independent effect on the outcome of that expression.

With **-fpath-coverage**, GCC finds and enumerates and records the taken prime paths of each function, unless the number of paths would exceed the limit controlled by **-fpath-coverage-limit**. If the limit is exceeded the function is not instrumented as if **-fpath-coverage** was not used. A prime path is the longest sequence of unique blocks, except possibly the first and last, which is not a subpath of any other path.

-ftest-coverage

Produce a notes file that the `gcov` code-coverage utility (see Chapter 11 [gcov—a Test Coverage Program], page 1067) can use to show program coverage. Each source file's note file is called `auxname.gcno`. Refer to the `-fprofile-arcs` option above for a description of `auxname` and instructions on how to generate test coverage data. Coverage data matches the source files more closely if you do not optimize.

-fprofile-abs-path

Automatically convert relative source file names to absolute path names in the `.gcno` files. This allows `gcov` to find the correct sources in projects where compilations occur with different working directories.

-fprofile-dir=path

Set the directory to search for the profile data files in to *path*. This option affects only the profile data generated by `-fprofile-generate`, `-ftest-coverage`, `-fprofile-arcs` and used by `-fprofile-use` and `-fbranch-probabilities` and its related options. Both absolute and relative paths can be used. By default, GCC uses the current directory as *path*, thus the profile data file appears in the same directory as the object file. In order to prevent the file name clashing, if the object file name is not an absolute path, we mangle the absolute path of the `sourcename.gcda` file and use it as the file name of a `.gcda` file. See details about the file naming in `-fprofile-arcs`. See similar option `-fprofile-note`.

When an executable is run in a massive parallel environment, it is recommended to save profile to different folders. That can be done with variables in *path* that are exported during run-time:

`%p` process ID.

`%q{VAR}` value of environment variable *VAR*

-fprofile-generate**-fprofile-generate=path**

Enable options usually used for instrumenting application to produce profile useful for later recompilation with profile feedback based optimization. You must use `-fprofile-generate` both when compiling and when linking your program.

The following options are enabled: `-fprofile-arcs`, `-fprofile-values`, `-finline-functions`, and `-fipa-bit-cp`.

If *path* is specified, GCC looks at the *path* to find the profile feedback data files. See `-fprofile-dir`.

To optimize the program based on the collected profile information, use `-fprofile-use`. See Section 3.12 [Optimize Options], page 196, for more information.

-fprofile-info-section**-fprofile-info-section=name**

Register the profile information in the specified section instead of using a constructor/destructor. The section name is *name* if it is specified, otherwise the

section name defaults to `.gcov_info`. A pointer to the profile information generated by `-fprofile-arcs` is placed in the specified section for each translation unit. This option disables the profile information registration through a constructor and it disables the profile information processing through a destructor. This option is not intended to be used in hosted environments such as GNU/Linux. It targets freestanding environments (for example embedded systems) with limited resources which do not support constructors/destructors or the C library file I/O.

The linker could collect the input sections in a continuous memory block and define start and end symbols. A GNU linker script example which defines a linker output section follows:

```
.gcov_info      :
{
    PROVIDE (__gcov_info_start = .);
    KEEP (*(.gcov_info))
    PROVIDE (__gcov_info_end = .);
}
```

The program could dump the profiling information registered in this linker set for example like this:

```
#include <gcov.h>
#include <stdio.h>
#include <stdlib.h>

extern const struct gcov_info *const __gcov_info_start[];
extern const struct gcov_info *const __gcov_info_end[];

static void
dump (const void *d, unsigned n, void *arg)
{
    const unsigned char *c = d;

    for (unsigned i = 0; i < n; ++i)
        printf ("%02x", c[i]);
}

static void
filename (const char *f, void *arg)
{
    __gcov_filename_to_gcfn (f, dump, arg );
}

static void *
allocate (unsigned length, void *arg)
{
    return malloc (length);
}

static void
dump_gcov_info (void)
{
    const struct gcov_info *const *info = __gcov_info_start;
    const struct gcov_info *const *end = __gcov_info_end;

    /* Obfuscate variable to prevent compiler optimizations. */
```

```

__asm__ (" : "+r" (info));

while (info != end)
{
    void *arg = NULL;
    __gcov_info_to_gcda (*info, filename, dump, allocate, arg);
    putchar ('\n');
    ++info;
}

int
main (void)
{
    dump_gcov_info ();
    return 0;
}

```

The `merge-stream` subcommand of `gcov-tool` may be used to deserialize the data stream generated by the `__gcov_filename_to_gcfn` and `__gcov_info_to_gcda` functions and merge the profile information into `.gcda` files on the host filesystem.

-fprofile-note=*path*

If *path* is specified, GCC saves `.gcno` file into *path* location. If you combine the option with multiple source files, the `.gcno` file will be overwritten.

-fprofile-prefix-path=*path*

This option can be used in combination with `profile-generate=profile_dir` and `profile-use=profile_dir` to inform GCC where is the base directory of built source tree. By default *profile_dir* will contain files with mangled absolute paths of all object files in the built project. This is not desirable when directory used to build the instrumented binary differs from the directory used to build the binary optimized with profile feedback because the profile data will not be found during the optimized build. In such setups `-fprofile-prefix-path=`*path* with *path* pointing to the base directory of the build can be used to strip the irrelevant part of the path and keep all file names relative to the main build directory.

-fprofile-prefix-map=*old=new*

When compiling files residing in directory *old*, record profiling information (with `--coverage`) describing them as if the files resided in directory *new* instead. See also `-ffile-prefix-map` and `-fcanon-prefix-map`.

-fprofile-update=*method*

Alter the update method for an application instrumented for profile feedback based optimization. The *method* argument should be one of `'single'`, `'atomic'` or `'prefer-atomic'`. The first one is useful for single-threaded applications, while the second one prevents profile corruption by emitting thread-safe code.

Warning: When an application does not properly join all threads (or creates an detached thread), a profile file can be still corrupted.

Using `'prefer-atomic'` would be transformed either to `'atomic'`, when supported by a target, or to `'single'` otherwise. The GCC driver automatically

selects ‘**prefer-atomic**’ when **-pthread** is present in the command line, otherwise the default method is ‘**single**’.

If ‘**atomic**’ is selected, then the profile information is updated using atomic operations on a best-effort basis. Ideally, the profile information is updated through atomic operations in hardware. If the target platform does not support the required atomic operations in hardware, however, **libatomic** is available, then the profile information is updated through calls to **libatomic**. If the target platform neither supports the required atomic operations in hardware nor **libatomic**, then the profile information is not atomically updated and a warning is issued. In this case, the obtained profiling information may be corrupt for multi-threaded applications.

For performance reasons, if 64-bit counters are used for the profiling information and the target platform only supports 32-bit atomic operations in hardware, then the performance critical profiling updates are done using two 32-bit atomic operations for each counter update. If a signal interrupts these two operations updating a counter, then the profiling information may be in an inconsistent state.

-fprofile-filter-files=regex

Instrument only functions from files whose name matches any of the regular expressions (separated by semi-colons).

For example, **-fprofile-filter-files=main*.c;module.**.c** will instrument only **main.c** and all C files starting with ‘**module**’.

-fprofile-exclude-files=regex

Instrument only functions from files whose name does not match any of the regular expressions (separated by semi-colons).

For example, **-fprofile-exclude-files=/usr/*.*** will prevent instrumentation of all files that are located in the **/usr/** folder.

-fprofile-reproducible=[multithreaded|parallel-runs|serial]

Control level of reproducibility of profile gathered by **-fprofile-generate**. This makes it possible to rebuild program with same outcome which is useful, for example, for distribution packages.

With **-fprofile-reproducible=serial** the profile gathered by **-fprofile-generate** is reproducible provided the trained program behaves the same at each invocation of the train run, it is not multi-threaded and profile data streaming is always done in the same order. Note that profile streaming happens at the end of program run but also before **fork** function is invoked.

Note that it is quite common that execution counts of some part of programs depends, for example, on length of temporary file names or memory space randomization (that may affect hash-table collision rate). Such non-reproducible part of programs may be annotated by **no_instrument_function** function attribute. **gcov-dump** with **-l** can be used to dump gathered data and verify that they are indeed reproducible.

With **-fprofile-reproducible=parallel-runs** collected profile stays reproducible regardless the order of streaming of the data into gcda files. This setting

makes it possible to run multiple instances of instrumented program in parallel (such as with `make -j`). This reduces quality of gathered data, in particular of indirect call profiling.

`-fsanitize=address`

Enable AddressSanitizer, a fast memory error detector. Memory access instructions are instrumented to detect out-of-bounds and use-after-free bugs. The option enables `-fsanitize-address-use-after-scope`. See <https://github.com/google/sanitizers/wiki/AddressSanitizer> for more details. The run-time behavior can be influenced using the `ASAN_OPTIONS` environment variable. When set to `help=1`, the available options are shown at startup of the instrumented program. See <https://github.com/google/sanitizers/wiki/AddressSanitizerFlags#run-time-flags> for a list of supported options. The option cannot be combined with `-fsanitize=thread` or `-fsanitize=hwaddress`. Note that the only targets `-fsanitize=hwaddress` is currently supported on are x86-64 (only with `-mlam=u48` or `-mlam=u57` options) and AArch64, in both cases only in ABIs with 64-bit pointers. Similarly, `-fsanitize=memtag-stack` is currently only supported on AArch64 ABIs with 64-bit pointers.

When compiling with `-fsanitize=address`, you should also use `-g` to produce more meaningful output. To get more accurate stack traces, it is possible to use options such as `-O0`, `-O1`, or `-Og` (which, for instance, prevent most function inlining), `-fno-optimize-sibling-calls` (which prevents optimizing sibling and tail recursive calls; this option is implicit for `-O0`, `-O1`, or `-Og`), or `-fno-ipa-icf` (which disables Identical Code Folding for functions). Using `-fno-omit-frame-pointer` also improves stack traces. Since multiple runs of the program may yield backtraces with different addresses due to ASLR (Address Space Layout Randomization), it may be desirable to turn ASLR off. On Linux, this can be achieved with `'setarch `uname -m` -R ./prog'`.

`-fsanitize=kernel-address`

Enable AddressSanitizer for Linux kernel. See <https://github.com/google/kernel-sanitizers> for more details.

`-fsanitize=hwaddress`

Enable Hardware-assisted AddressSanitizer, which uses a hardware ability to ignore the top byte of a pointer to allow the detection of memory errors with a low memory overhead. Memory access instructions are instrumented to detect out-of-bounds and use-after-free bugs. The option enables `-fsanitize-address-use-after-scope`. See <https://clang.llvm.org/docs/HardwareAssistedAddressSanitizerDesign.html> for more details. The run-time behavior can be influenced using the `HWASAN_OPTIONS` environment variable. When set to `help=1`, the available options are shown at startup of the instrumented program. The option cannot be combined with `-fsanitize=thread` or `-fsanitize=address`, and is currently only available on AArch64.

-fsanitize=kernel-hwaddress

Enable Hardware-assisted AddressSanitizer for compilation of the Linux kernel. Similar to **-fsanitize=kernel-address** but using an alternate instrumentation method, and similar to **-fsanitize=hwaddress** but with instrumentation differences necessary for compiling the Linux kernel. These differences are to avoid hwasan library initialization calls and to account for the stack pointer having a different value in its top byte.

Note: This option has different defaults than **-fsanitize=hwaddress**. Instrumenting the stack and alloca calls are not on by default. Using a random frame tag is not implemented for kernel instrumentation.

-fsanitize=mentag-stack

Use Memory Tagging Extension instructions instead of instrumentation to allow the detection of memory errors. Similar to HWASAN, it is also a probabilistic method. This option is available only on those AArch64 architectures that support Memory Tagging Extensions.

-fsanitize=pointer-compare

Instrument comparison operation (<, <=, >, >=) with pointer operands. The option must be combined with either **-fsanitize=kernel-address** or **-fsanitize=address**. The option cannot be combined with **-fsanitize=thread**. *Note:* By default the check is disabled at run time. To enable it, add `detect_invalid_pointer_pairs=2` to the environment variable `ASAN_OPTIONS`. Using `detect_invalid_pointer_pairs=1` detects invalid operation only when both pointers are non-null.

-fsanitize=pointer-subtract

Instrument subtraction with pointer operands. The option must be combined with either **-fsanitize=kernel-address** or **-fsanitize=address**. The option cannot be combined with **-fsanitize=thread**. *Note:* By default the check is disabled at run time. To enable it, add `detect_invalid_pointer_pairs=2` to the environment variable `ASAN_OPTIONS`. Using `detect_invalid_pointer_pairs=1` detects invalid operation only when both pointers are non-null.

-fsanitize=shadow-call-stack

Enable ShadowCallStack, a security enhancement mechanism used to protect programs against return address overwrites (e.g. stack buffer overflows.) It works by saving a function's return address to a separately allocated shadow call stack in the function prologue and restoring the return address from the shadow call stack in the function epilogue. Instrumentation only occurs in functions that need to save the return address to the stack.

Currently it only supports the aarch64 platform. It is specifically designed for linux kernels that enable the `CONFIG_SHADOW_CALL_STACK` option. For the user space programs, runtime support is not currently provided in `libc` and `libgcc`. Users who want to use this feature in user space need to provide their own support for the runtime. It should be noted that this may cause the ABI rules to be broken.

On aarch64, the instrumentation makes use of the platform register `x18`. This generally means that any code that may run on the same thread as code com-

piled with `ShadowCallStack` must be compiled with the flag `-ffixed-x18`, otherwise functions compiled without `-ffixed-x18` might clobber `x18` and so corrupt the shadow stack pointer.

Also, because there is no userspace runtime support, code compiled with `ShadowCallStack` cannot use exception handling. Use `-fno-exceptions` to turn off exceptions.

See <https://clang.llvm.org/docs/ShadowCallStack.html> for more details.

`-fsanitize=thread`

Enable ThreadSanitizer, a fast data race detector. Memory access instructions are instrumented to detect data race bugs. See <https://github.com/google/sanitizers/wiki#threadsanitizer> for more details. The run-time behavior can be influenced using the `TSAN_OPTIONS` environment variable; see <https://github.com/google/sanitizers/wiki/ThreadSanitizerFlags> for a list of supported options. The option cannot be combined with `-fsanitize=address`, `-fsanitize=leak`.

When compiling with `-fsanitize=thread`, you should also use `-g` to produce more meaningful output.

Note that sanitized atomic builtins cannot throw exceptions when operating on invalid memory addresses with non-call exceptions (`-fnon-call-exceptions`).

`-fsanitize=leak`

Enable LeakSanitizer, a memory leak detector. This option only matters for linking of executables. The executable is linked against a library that overrides `malloc` and other allocator functions. See <https://github.com/google/sanitizers/wiki/AddressSanitizerLeakSanitizer> for more details. The run-time behavior can be influenced using the `LSAN_OPTIONS` environment variable. The option cannot be combined with `-fsanitize=thread`.

`-fsanitize=undefined`

Enable UndefinedBehaviorSanitizer, a fast undefined behavior detector. Various computations are instrumented to detect undefined behavior at run-time. See <https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html> for more details. The run-time behavior can be influenced using the `UBSAN_OPTIONS` environment variable. Current suboptions are:

`-fsanitize=shift`

This option enables checking that the result of a shift operation is not undefined. Note that what exactly is considered undefined differs slightly between C and C++, as well as between ISO C90 and C99, etc. This option has two suboptions, `-fsanitize=shift-base` and `-fsanitize=shift-exponent`.

`-fsanitize=shift-exponent`

This option enables checking that the second argument of a shift operation is not negative and is smaller than the precision of the promoted first argument.

-fsanitize=shift-base

If the second argument of a shift operation is within range, check that the result of a shift operation is not undefined. Note that what exactly is considered undefined differs slightly between C and C++, as well as between ISO C90 and C99, etc.

-fsanitize=integer-divide-by-zero

Detect integer division by zero.

-fsanitize=unreachable

With this option, the compiler turns the `__builtin_unreachable` call into a diagnostics message call instead. When reaching the `__builtin_unreachable` call, the behavior is undefined.

-fsanitize=vla-bound

This option instructs the compiler to check that the size of a variable length array is positive.

-fsanitize=null

This option enables pointer checking. Particularly, the application built with this option turned on will issue an error message when it tries to dereference a NULL pointer, or if a reference (possibly an rvalue reference) is bound to a NULL pointer, or if a method is invoked on an object pointed by a NULL pointer.

-fsanitize=return

This option enables return statement checking. Programs built with this option turned on will issue an error message when the end of a non-void function is reached without actually returning a value. This option works in C++ only.

-fsanitize=signed-integer-overflow

This option enables signed integer overflow checking. We check that the result of `+`, `*`, and both unary and binary `-` does not overflow in the signed arithmetics. This also detects `INT_MIN / -1` signed division. Note, integer promotion rules must be taken into account. That is, the following is not an overflow:

```
signed char a = SCHAR_MAX;
a++;
```

-fsanitize=bounds

This option enables instrumentation of array bounds. Various out of bounds accesses are detected. Flexible array members, flexible array member-like arrays, and initializers of variables with static storage are not instrumented, with the exception of flexible array member-like arrays for which `-fstrict-flex-arrays` or `-fstrict-flex-arrays=` options or `strict_flex_array` attributes say they shouldn't be treated like flexible array member-like arrays.

-fsanitize=bounds-strict

This option enables strict instrumentation of array bounds. Most out of bounds accesses are detected, including flexible array member-like arrays. Initializers of variables with static storage are not instrumented.

-fsanitize=alignment

This option enables checking of alignment of pointers when they are dereferenced, or when a reference is bound to insufficiently aligned target, or when a method or constructor is invoked on insufficiently aligned object.

-fsanitize=object-size

This option enables instrumentation of memory references using the `__builtin_dynamic_object_size` function. Various out of bounds pointer accesses are detected.

-fsanitize=float-divide-by-zero

Detect floating-point division by zero. Unlike other similar options, **-fsanitize=float-divide-by-zero** is not enabled by **-fsanitize=undefined**, since floating-point division by zero can be a legitimate way of obtaining infinities and NaNs.

-fsanitize=float-cast-overflow

This option enables floating-point type to integer conversion checking. We check that the result of the conversion does not overflow. Unlike other similar options, **-fsanitize=float-cast-overflow** is not enabled by **-fsanitize=undefined**. This option does not work well with `FE_INVALID` exceptions enabled.

-fsanitize=nonnull-attribute

This option enables instrumentation of calls, checking whether null values are not passed to arguments marked as requiring a non-null value by the `nonnull` function attribute.

-fsanitize=returns-nonnull-attribute

This option enables instrumentation of return statements in functions marked with `returns_nonnull` function attribute, to detect returning of null values from such functions.

-fsanitize=bool

This option enables instrumentation of loads from `bool`. If a value other than 0/1 is loaded, a run-time error is issued.

-fsanitize=enum

This option enables instrumentation of loads from an `enum` type. If a value outside the range of values for the `enum` type is loaded, a run-time error is issued.

-fsanitize=vptr

This option enables instrumentation of C++ member function calls, member accesses and some conversions between pointers to base

and derived classes, to verify the referenced object has the correct dynamic type.

-fsanitize=pointer-overflow

This option enables instrumentation of pointer arithmetics. If the pointer arithmetics overflows, a run-time error is issued.

-fsanitize=builtin

This option enables instrumentation of arguments to selected builtin functions. If an invalid value is passed to such arguments, a run-time error is issued. E.g. passing 0 as the argument to `__builtin_ctz` or `__builtin_clz` invokes undefined behavior and is diagnosed by this option.

Note that sanitizers tend to increase the rate of false positive warnings, most notably those around `-Wmaybe-uninitialized`. We recommend against combining `-Werror` and [the use of] sanitizers.

While `-ftrapv` causes traps for signed overflows to be emitted, `-fsanitize=undefined` gives a diagnostic message. This currently works only for the C family of languages.

-fno-sanitize=all

This option disables all previously enabled sanitizers. `-fsanitize=all` is not allowed, as some sanitizers cannot be used together.

-fasan-shadow-offset=number

This option forces GCC to use custom shadow offset in AddressSanitizer checks. It is useful for experimenting with different shadow memory layouts in Kernel AddressSanitizer.

-fsanitize-sections=s1,s2,...

Sanitize global variables in selected user-defined sections. *si* may contain wild-cards.

-fsanitize-recover[=opts]

`-fsanitize-recover=` controls error recovery mode for sanitizers mentioned in comma-separated list of *opts*. Enabling this option for a sanitizer component causes it to attempt to continue running the program as if no error happened. This means multiple runtime errors can be reported in a single program run, and the exit code of the program may indicate success even when errors have been reported. The `-fno-sanitize-recover=` option can be used to alter this behavior: only the first detected error is reported and program then exits with a non-zero exit code.

Currently this feature only works for `-fsanitize=undefined` (and its suboptions except for `-fsanitize=unreachable` and `-fsanitize=return`), `-fsanitize=float-cast-overflow`, `-fsanitize=float-divide-by-zero`, `-fsanitize=bounds-strict`, `-fsanitize=kernel-address` and `-fsanitize=address`. For these sanitizers error recovery is turned on by default, except `-fsanitize=address`, for which this feature is experimental. `-fsanitize-recover=all` and `-fno-sanitize-recover=all` is also accepted,

the former enables recovery for all sanitizers that support it, the latter disables recovery for all sanitizers that support it.

Even if a recovery mode is turned on the compiler side, it needs to be also enabled on the runtime library side, otherwise the failures are still fatal. The runtime library defaults to `halt_on_error=0` for ThreadSanitizer and UndefinedBehaviorSanitizer, while default value for AddressSanitizer is `halt_on_error=1`. This can be overridden through setting the `halt_on_error` flag in the corresponding environment variable.

Syntax without an explicit *opts* parameter is deprecated. It is equivalent to specifying an *opts* list of:

```
undefined,float-cast-overflow,float-divide-by-zero,bounds-strict
```

-fsanitize-address-use-after-scope

Enable sanitization of local variables to detect use-after-scope bugs. The option sets `-fstack-reuse` to 'none'.

-fsanitize-trap[=opts]

The `-fsanitize-trap=` option instructs the compiler to report for sanitizers mentioned in comma-separated list of *opts* undefined behavior using `__builtin_trap` rather than a `libubsan` library routine. If this option is enabled for certain sanitizer, it takes precedence over the `-fsanitizer-recover=` for that sanitizer, `__builtin_trap` will be emitted and be fatal regardless of whether recovery is enabled or disabled using `-fsanitize-recover=`.

The advantage of this is that the `libubsan` library is not needed and is not linked in, so this is usable even in freestanding environments.

Currently this feature works with `-fsanitize=undefined` (and its suboptions except for `-fsanitize=vptr`), `-fsanitize=float-cast-overflow`, `-fsanitize=float-divide-by-zero` and `-fsanitize=bounds-strict`. `-fsanitize-trap=all` can be also specified, which enables it for undefined suboptions, `-fsanitize=float-cast-overflow`, `-fsanitize=float-divide-by-zero` and `-fsanitize=bounds-strict`. If `-fsanitize-trap=undefined` or `-fsanitize-trap=all` is used and `-fsanitize=vptr` is enabled on the command line, the instrumentation is silently ignored as the instrumentation always needs `libubsan` support, `-fsanitize-trap=vptr` is not allowed.

-fsanitize-undefined-trap-on-error

The `-fsanitize-undefined-trap-on-error` option is deprecated equivalent of `-fsanitize-trap=all`.

-fsanitize-coverage=trace-pc

Enable coverage-guided fuzzing code instrumentation. Inserts a call to `__sanitizer_cov_trace_pc` into every basic block.

-fsanitize-coverage=trace-cmp

Enable dataflow guided fuzzing code instrumentation. Inserts a call to `__sanitizer_cov_trace_cmp1`, `__sanitizer_cov_trace_cmp2`, `__sanitizer_cov_trace_cmp4` or `__sanitizer_cov_trace_cmp8` for integral comparison with both operands variable or `__sanitizer_cov_trace_const_cmp1`, `__sanitizer_cov_trace_const_cmp2`, `__sanitizer_cov_`

`trace_const_cmp4` or `__sanitizer_cov_trace_const_cmp8` for integral comparison with one operand constant, `__sanitizer_cov_trace_cmpf` or `__sanitizer_cov_trace_cmpd` for float or double comparisons and `__sanitizer_cov_trace_switch` for switch statements.

`-fcf-protection=[full|branch|return|none|check]`

`-fcf-protection`

Enable code instrumentation to increase program security by checking that target addresses of control-flow transfer instructions (such as indirect function call, function return, indirect jump) are valid. This prevents diverting the flow of control to an unexpected target. This is intended to protect against such threats as Return-oriented Programming (ROP), and similarly call/jmp-oriented programming (COP/JOP).

The `-fcf-protection=` keywords are interpreted as follows.

The value **branch** tells the compiler to implement checking of validity of control-flow transfer at the point of indirect branch instructions, i.e. call/jmp instructions.

The value **return** implements checking of validity at the point of returning from a function.

The value **full** is an alias for specifying both **branch** and **return**.

The value **check** is used for the final link with link-time optimization (LTO). An error is issued if LTO object files are compiled with different `-fcf-protection` values. The value **check** is ignored at the compile time.

The value **none** turns off instrumentation.

`-fcf-protection` is an alias for `-fcf-protection=full`. To override a previous `-fcf-protection` option on the command line, add `-fcf-protection=none` and then `-fcf-protection=kind`.

The macro `__CET__` is defined when `-fcf-protection` is used. The first bit of `__CET__` is set to 1 for the value **branch** and the second bit of `__CET__` is set to 1 for the **return**.

You can also use the `nocf_check` attribute to identify which functions and calls should be skipped from instrumentation (see Section 6.4.1 [Common Attributes], page 595).

Currently the x86 GNU/Linux target provides an implementation based on Intel Control-flow Enforcement Technology (CET) which works for i686 processor or newer.

`-fharden-compares`

For every logical test that survives gimple optimizations and is *not* the condition in a conditional branch (for example, conditions tested for conditional moves, or to store in boolean variables), emit extra code to compute and verify the reversed condition, and to call `__builtin_trap` if the results do not match. Use with `'-fharden-conditional-branches'` to cover all conditionals.

`-fharden-conditional-branches`

For every non-vectorized conditional branch that survives gimple optimizations, emit extra code to compute and verify the reversed condition, and to call `__`

`builtin_trap` if the result is unexpected. Use with ‘`-fharden-compares`’ to cover all conditionals.

-fharden-control-flow-redundancy

Emit extra code to set booleans when entering basic blocks, and to verify and trap, at function exits, when the booleans do not form an execution path that is compatible with the control flow graph.

Verification takes place before returns, before mandatory tail calls (see below) and, optionally, before escaping exceptions with `-fhardcfr-check-exceptions`, before returning calls with `-fhardcfr-check-returning-calls`, and before `noreturn` calls with `-fhardcfr-check-noreturn-calls`).

Tail call optimization takes place too late to affect control flow redundancy, but calls annotated as mandatory tail calls by language front-ends, and any calls marked early enough as potential tail calls would also have verification issued before the call, but these possibilities are merely theoretical, as these conditions can only be met when using custom compiler plugins.

-fhardcfr-skip-leaf

Disable `-fharden-control-flow-redundancy` in leaf functions.

-fhardcfr-check-exceptions

When `-fharden-control-flow-redundancy` is active, check the recorded execution path against the control flow graph at exception escape points, as if the function body was wrapped with a cleanup handler that performed the check and reraised. This option is enabled by default; use `-fno-hardcfr-check-exceptions` to disable it.

-fhardcfr-check-returning-calls

When `-fharden-control-flow-redundancy` is active, check the recorded execution path against the control flow graph before any function call immediately followed by a return of its result, if any, so as to not prevent tail-call optimization, whether or not it is ultimately optimized to a tail call.

This option is enabled by default whenever sibling call optimizations are enabled (see `-foptimize-sibling-calls`), but it can be enabled (or disabled, using its negated form) explicitly, regardless of the optimizations.

-fhardcfr-check-noreturn-calls=[always|no-xthrow|nothrow|never]

When `-fharden-control-flow-redundancy` is active, check the recorded execution path against the control flow graph before `noreturn` calls, either all of them (`always`), those that aren’t expected to return control to the caller through an exception (`no-xthrow`, the default), those that may not return control to the caller through an exception either (`nothrow`), or none of them (`never`).

Checking before a `noreturn` function that may return control to the caller through an exception may cause checking to be performed more than once, if the exception is caught in the caller, whether by a handler or a cleanup. When `-fhardcfr-check-exceptions` is also enabled, the compiler will avoid associating a `noreturn` call with the implicitly-added cleanup handler, since it would be redundant with the check performed before the call, but other handlers

or cleanups in the function, if activated, will modify the recorded execution path and check it again when another checkpoint is hit. The checkpoint may even be another `noreturn` call, so checking may end up performed multiple times.

Various optimizers may cause calls to be marked as `noreturn` and/or `nothrow`, even in the absence of the corresponding attributes, which may affect the placement of checks before calls, as well as the addition of implicit cleanup handlers for them. This unpredictability, and the fact that raising and reraising exceptions frequently amounts to implicitly calling `noreturn` functions, have made `no-xthrow` the default setting for this option: it excludes from the `noreturn` treatment only internal functions used to (re)raise exceptions, that are not affected by these optimizations.

`-fhardened`

Enable a set of flags for C and C++ that improve the security of the generated code without affecting its ABI. The precise flags enabled may change between major releases of GCC, but are currently:

```
-D_FORTIFY_SOURCE=3
-D_GLIBCXX_ASSERTIONS
-ftrivial-auto-var-init=zero
-fPIE -pie -Wl,-z,relro,-z,now
-fstack-protector-strong
-fstack-clash-protection
-fcf-protection=full (x86 GNU/Linux only)
```

The list of options enabled by `-fhardened` can be generated using the `--help=hardened` option.

When the system glibc is older than 2.35, `-D_FORTIFY_SOURCE=2` is used instead.

This option is intended to be used in production builds, not merely in debug builds.

Currently, `-fhardened` is only supported on GNU/Linux targets.

`-fhardened` only enables a particular option if it wasn't already specified anywhere on the command line. For instance, `-fhardened -fstack-protector` will only enable `-fstack-protector`, but not `-fstack-protector-strong`.

`-fstack-protector`

Emit extra code to check for buffer overflows, such as stack smashing attacks. This is done by adding a guard variable to functions with vulnerable objects. This includes functions that call `alloca`, and functions with buffers larger than or equal to 8 bytes. The guards are initialized when a function is entered and then checked when the function exits. If a guard check fails, an error message is printed and the program exits. Only variables that are actually allocated on the stack are considered, optimized away variables or variables allocated in registers don't count.

`-fstack-protector-all`

Like `-fstack-protector` except that all functions are protected.

-fstack-protector-strong

Like **-fstack-protector** but includes additional functions to be protected — those that have local array definitions, or have references to local frame addresses. Only variables that are actually allocated on the stack are considered, optimized away variables or variables allocated in registers don't count.

-fstack-protector-explicit

Like **-fstack-protector** but only protects those functions which have the `stack_protect` attribute.

-fstack-check

Generate code to verify that you do not go beyond the boundary of the stack. You should specify this flag if you are running in an environment with multiple threads, but you only rarely need to specify it in a single-threaded environment since stack overflow is automatically detected on nearly all systems if there is only one stack.

Note that this switch does not actually cause checking to be done; the operating system or the language runtime must do that. The switch causes generation of code to ensure that they see the stack being extended.

You can additionally specify a string parameter: **'no'** means no checking, **'generic'** means force the use of old-style checking, **'specific'** means use the best checking method and is equivalent to bare **-fstack-check**.

Old-style checking is a generic mechanism that requires no specific target support in the compiler but comes with the following drawbacks:

1. Modified allocation strategy for large objects: they are always allocated dynamically if their size exceeds a fixed threshold. Note this may change the semantics of some code.
2. Fixed limit on the size of the static frame of functions: when it is topped by a particular function, stack checking is not reliable and a warning is issued by the compiler.
3. Inefficiency: because of both the modified allocation strategy and the generic implementation, code performance is hampered.

Note that old-style stack checking is also the fallback method for **'specific'** if no target support has been added in the compiler.

'-fstack-check=' is designed for Ada's needs to detect infinite recursion and stack overflows. **'specific'** is an excellent choice when compiling Ada code. It is not generally sufficient to protect against stack-clash attacks. To protect against those you want **'-fstack-clash-protection'**.

-fstack-clash-protection

Generate code to prevent stack clash style attacks. When this option is enabled, the compiler will only allocate one page of stack space at a time and each page is accessed immediately after allocation. Thus, it prevents allocations from jumping over any stack guard page provided by the operating system.

Most targets do not fully support stack clash protection. However, on those targets **-fstack-clash-protection** will protect dynamic stack allocations.

`-fstack-clash-protection` may also provide limited protection for static stack allocations if the target supports `-fstack-check=specific`.

`-fstack-limit-register=reg`

`-fstack-limit-symbol=sym`

`-fno-stack-limit`

Generate code to ensure that the stack does not grow beyond a certain value, either the value of a register or the address of a symbol. If a larger stack is required, a signal is raised at run time. For most targets, the signal is raised before the stack overruns the boundary, so it is possible to catch the signal without taking special precautions.

For instance, if the stack starts at absolute address ‘0x80000000’ and grows downwards, you can use the flags `-fstack-limit-symbol=__stack_limit` and `-Wl,--defsym,__stack_limit=0x7ffe0000` to enforce a stack limit of 128KB. Note that this may only work with the GNU linker.

You can locally override stack limit checking by using the `no_stack_limit` function attribute (see Section 6.4.1 [Common Attributes], page 595).

`-fsplit-stack`

Generate code to automatically split the stack before it overflows. The resulting program has a discontinuous stack which can only overflow if the program is unable to allocate any more memory. This is most useful when running threaded programs, as it is no longer necessary to calculate a good stack size to use for each thread. This is currently only implemented for the x86 targets running GNU/Linux.

When code compiled with `-fsplit-stack` calls code compiled without `-fsplit-stack`, there may not be much stack space available for the latter code to run. If compiling all code, including library code, with `-fsplit-stack` is not an option, then the linker can fix up these calls so that the code compiled without `-fsplit-stack` always has a large stack. Support for this is implemented in the gold linker in GNU Binutils release 2.21 and later.

`-fstrub=disable`

Disable stack scrubbing entirely, ignoring any `strub` attributes. See Section 6.4.1 [Common Attributes], page 595.

`-fstrub=strict`

Functions default to `strub` mode `disabled`, and apply `strictly` the restriction that only functions associated with `strub-callable` modes (`at-calls`, `callable` and `always_inline internal`) are callable by functions with `strub-enabled` modes (`at-calls` and `internal`).

`-fstrub=relaxed`

Restore the default stack scrub (`strub`) setting, namely, `strub` is only enabled as required by `strub` attributes associated with function and data types. `Relaxed` means that `strub` contexts are only prevented from calling functions explicitly associated with `strub` mode `disabled`. This option is only useful to override other `-fstrub=*` options that precede it in the command line.

-fstrub=at-calls

Enable **at-calls strub** mode where viable. The primary use of this option is for testing. It exercises the **strub** machinery in scenarios strictly local to a translation unit. This **strub** mode modifies function interfaces, so any function that is visible to other translation units, or that has its address taken, will *not* be affected by this option. Optimization options may also affect viability. See the **strub** attribute documentation for details on viability and eligibility requirements.

-fstrub=internal

Enable **internal strub** mode where viable. The primary use of this option is for testing. This option is intended to exercise thoroughly parts of the **strub** machinery that implement the less efficient, but interface-preserving **strub** mode. Functions that would not be affected by this option are quite uncommon.

-fstrub=all

Enable some **strub** mode where viable. When both **strub** modes are viable, **at-calls** is preferred. **-fdump-ipa-strubm** adds function attributes that tell which mode was selected for each function. The primary use of this option is for testing, to exercise thoroughly the **strub** machinery.

-fvtable-verify=[std|preinit|none]

This option is only available when compiling C++ code. It turns on (or off, if using **-fvtable-verify=none**) the security feature that verifies at run time, for every virtual call, that the vtable pointer through which the call is made is valid for the type of the object, and has not been corrupted or overwritten. If an invalid vtable pointer is detected at run time, an error is reported and execution of the program is immediately halted.

This option causes run-time data structures to be built at program startup, which are used for verifying the vtable pointers. The options ‘**std**’ and ‘**preinit**’ control the timing of when these data structures are built. In both cases the data structures are built before execution reaches **main**. Using **-fvtable-verify=std** causes the data structures to be built after shared libraries have been loaded and initialized. **-fvtable-verify=preinit** causes them to be built before shared libraries have been loaded and initialized.

If this option appears multiple times in the command line with different values specified, ‘**none**’ takes highest priority over both ‘**std**’ and ‘**preinit**’; ‘**preinit**’ takes priority over ‘**std**’.

-fvtv-debug

When used in conjunction with **-fvtable-verify=std** or **-fvtable-verify=preinit**, causes debug versions of the runtime functions for the vtable verification feature to be called. This flag also causes the compiler to log information about which vtable pointers it finds for each class. This information is written to a file named **vtv_set_ptr_data.log** in the directory named by the environment variable **VTV_LOGS_DIR** if that is defined or the current working directory otherwise.

Note: This feature *appends* data to the log file. If you want a fresh log file, be sure to delete any existing one.

-fvtn-counts

This is a debugging flag. When used in conjunction with **-fvtable-verify=std** or **-fvtable-verify=preinit**, this causes the compiler to keep track of the total number of virtual calls it encounters and the number of verifications it inserts. It also counts the number of calls to certain run-time library functions that it inserts and logs this information for each compilation unit. The compiler writes this information to a file named `vtv_count_data.log` in the directory named by the environment variable `VTV_LOGS_DIR` if that is defined or the current working directory otherwise. It also counts the size of the vtable pointer sets for each class, and writes this information to `vtv_class_set_sizes.log` in the same directory.

Note: This feature *appends* data to the log files. To get fresh log files, be sure to delete any existing ones.

-finstrument-functions

Generate instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions are called with the address of the current function and its call site. (On some platforms, `__builtin_return_address` does not work beyond the current function, so the call site information may not be available to the profiling functions otherwise.)

```
void __cyg_profile_func_enter (void *this_fn,
                             void *call_site);
void __cyg_profile_func_exit  (void *this_fn,
                             void *call_site);
```

The first argument is the address of the start of the current function, which may be looked up exactly in the symbol table.

This instrumentation is also done for functions expanded inline in other functions. The profiling calls indicate where, conceptually, the inline function is entered and exited. This means that addressable versions of such functions must be available. If all your uses of a function are expanded inline, this may mean an additional expansion of code size. If you use **extern inline** in your C code, an addressable version of such functions must be provided. (This is normally the case anyway, but if you get lucky and the optimizer always expands the functions inline, you might have gotten away without providing static copies.)

A function may be given the attribute `no_instrument_function`, in which case this instrumentation is not done. This can be used, for example, for the profiling functions listed above, high-priority interrupt routines, and any functions from which the profiling functions cannot safely be called (perhaps signal handlers, if the profiling routines generate output or allocate memory). See Section 6.4.1 [Common Attributes], page 595.

-finstrument-functions-once

This is similar to **-finstrument-functions**, but the profiling functions are called only once per instrumented function, i.e. the first profiling function

is called after the first entry into the instrumented function and the second profiling function is called before the exit corresponding to this first entry.

The definition of **once** for the purpose of this option is a little vague because the implementation is not protected against data races. As a result, the implementation only guarantees that the profiling functions are called at *least* once per process and at *most* once per thread, but the calls are always paired, that is to say, if a thread calls the first function, then it will call the second function, unless it never reaches the exit of the instrumented function.

-finstrument-functions-exclude-file-list=*file,file,...*

Set the list of functions that are excluded from instrumentation (see the description of **-finstrument-functions**). If the file that contains a function definition matches with one of *file*, then that function is not instrumented. The match is done on substrings: if the *file* parameter is a substring of the file name, it is considered to be a match.

For example:

```
-finstrument-functions-exclude-file-list=/bits/stl,include/sys
```

excludes any inline function defined in files whose pathnames contain **/bits/stl** or **include/sys**.

If, for some reason, you want to include letter **'** in one of *sym*, write **'\,'**. For example, **-finstrument-functions-exclude-file-list='\,\,tmp'** (note the single quote surrounding the option).

-finstrument-functions-exclude-function-list=*sym,sym,...*

This is similar to **-finstrument-functions-exclude-file-list**, but this option sets the list of function names to be excluded from instrumentation. The function name to be matched is its user-visible name, such as **vector<int> blah(const vector<int> &)**, not the internal mangled name (e.g., **_Z4blahRSt6vectorIiSaIiEE**). The match is done on substrings: if the *sym* parameter is a substring of the function name, it is considered to be a match. For C99 and C++ extended identifiers, the function name must be given in UTF-8, not using universal character names.

-fpatchable-function-entry=*N[,M]*

Generate *N* NOPs right at the beginning of each function, with the function entry point before the *M*th NOP. If *M* is omitted, it defaults to 0 so the function entry points to the address just at the first NOP. The NOP instructions reserve extra space which can be used to patch in any desired instrumentation at run time, provided that the code segment is writable. The amount of space is controllable indirectly via the number of NOPs; the NOP instruction used corresponds to the instruction emitted by the internal GCC back-end interface **gen_nop**. This behavior is target-specific and may also depend on the architecture variant and/or other compilation options.

For run-time identification, the starting addresses of these areas, which correspond to their respective function entries minus *M*, are additionally collected in the **__patchable_function_entries** section of the resulting binary.

Note that the value of **__attribute__((patchable_function_entry(N,M)))** takes precedence over command-line option **-fpatchable-function-**

entry=N,M. This can be used to increase the area size or to remove it completely on a single function. If $N=0$, no pad location is recorded.

The NOP instructions are inserted at—and maybe before, depending on M —the function entry address, even before the prologue. On PowerPC with the ELFv2 ABI, for a function with dual entry points, the local entry point is this function entry address by default. See the `-msplit-patch-nops` option to change this.

The maximum value of N and M is 65535. On PowerPC with the ELFv2 ABI, for a function with dual entry points, the supported values for M are 0, 2, 6 and 14 when not using `-msplit-patch-nops`.

3.14 Options Controlling the Preprocessor

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

If you use the `-E` option, nothing is done except preprocessing. Some of these options make sense only together with `-E` because they cause the preprocessor output to be unsuitable for actual compilation.

In addition to the options listed here, there are a number of options to control search paths for include files documented in Section 3.17 [Directory Options], page 282. Options to control preprocessor diagnostics are listed in Section 3.9 [Warning Options], page 101.

`-D name`

`--define-macro=name`

`--define-macro name`

Predefine *name* as a macro, with definition 1.

`-D name=definition`

`--define-macro=name=definition`

`--define-macro name=definition`

The contents of *definition* are tokenized and processed as if they appeared during translation phase three in a `#define` directive. In particular, the definition is truncated by embedded newline characters.

If you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax.

If you wish to define a function-like macro on the command line, write its argument list with surrounding parentheses before the equals sign (if any). Parentheses are meaningful to most shells, so you should quote the option. With `sh` and `csh`, `-D'name(args...)=definition'` works.

`-D` and `-U` options are processed in the order they are given on the command line. All `-imacros file` and `-include file` options are processed after all `-D` and `-U` options.

- `-U name`
- `--undefine-macro=name`
- `--undefine-macro name`
 - Cancel any previous definition of *name*, either built in or provided with a `-D` option.
- `-include file`
- `--include=file`
- `--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`
- `--imacros=file`
- `--imacros file`
 - Exactly like `-include`, except that any output produced by scanning *file* is thrown away. Macros it defines remain defined. This allows you to acquire all the macros from a header without also processing its declarations.
 - All files specified by `-imacros` are processed before all files specified by `-include`.
- `-undef`
 - Do not predefine any system-specific or GCC-specific macros. The standard predefined macros remain defined.
- `-pthread`
 - Define additional macros required for using the POSIX threads library. You should use this option consistently for both compilation and linking. This option is supported on GNU/Linux targets, most other Unix derivatives, and also on x86 Cygwin and MinGW targets.
- `-M`
- `--dependencies`
 - 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 3.21 [Environment Variables], page 552). Debug output is still sent to the regular output stream as normal.

Passing `-M` to the driver implies `-E`, and suppresses warnings with an implicit `-w`.

`-MM`

`--user-dependencies`

Like `-M` but do not mention header files that are found in system header directories, nor header files that are included, directly or indirectly, from such a header.

This implies that the choice of angle brackets or double quotes in an `#include` directive does not in itself determine whether that header appears in `-MM` dependency output.

`-MF file` When used with `-M` or `-MM`, specifies a file to write the dependencies to. If no `-MF` switch is given the preprocessor sends the rules to the same place it would send preprocessed output.

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

If *file* is `-`, then the dependencies are written to `stdout`.

`-MG`

`--print-missing-file-dependencies`

In conjunction with an option such as `-M` requesting dependency generation, `-MG` assumes missing header files are generated files and adds them to the dependency list without raising an error. The dependency filename is taken directly from the `#include` directive without prepending any path. `-MG` also suppresses preprocessed output, as a missing header file renders this useless.

This feature is used in automatic updating of makefiles.

`-Mno-modules`

Disable dependency generation for compiled module interfaces.

`-MP`

This option instructs CPP to add a phony target for each dependency other than the main file, causing each to depend on nothing. These dummy rules work around errors `make` gives if you remove header files without updating the `Makefile` to match.

This is typical output:

```
test.o: test.c test.h
```

```
test.h:
```

`-MT target`

Change the target of the rule emitted by dependency generation. By default CPP takes the name of the main input file, deletes any directory components and any file suffix such as `.c`, and appends the platform's usual object suffix. The result is the target.

An `-MT` option sets the target to be exactly the string you specify. If you want multiple targets, you can specify them as a single argument to `-MT`, or use multiple `-MT` options.

For example, `-MT '$(objpfx)foo.o'` might give

```
$(objpfx)foo.o: foo.c
```

`-MQ target`

Same as `-MT`, but it quotes any characters which are special to Make.

`-MQ '$(objpfx)foo.o'` gives

```
$$$(objpfx)foo.o: foo.c
```

The default target is automatically quoted, as if it were given with `-MQ`.

`-MD`

`--write-dependencies`

`-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 269), 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`

`--write-user-dependencies`

Like `-MD` except mention only user header files, not system header files.

`-fpreprocessed`

Indicate to the preprocessor that the input file has already been preprocessed. This suppresses things like macro expansion, trigraph conversion, escaped new-line splicing, and processing of most directives. The preprocessor still recognizes and removes comments, so that you can pass a file preprocessed with `-C` to the compiler without problems. In this mode the integrated preprocessor is little more than a tokenizer for the front ends.

`-fpreprocessed` is implicit if the input file has one of the extensions `‘.i’`, `‘.ii’` or `‘.mi’`. These are the extensions that GCC uses for preprocessed files created by `-save-temps`.

`-fdirectives-only`

When preprocessing, handle directives, but do not expand macros.

The option's behavior depends on the `-E` and `-fpreprocessed` options.

With `-E`, preprocessing is limited to the handling of directives such as `#define`, `#ifdef`, and `#error`. Other preprocessor operations, such as macro expansion and trigraph conversion are not performed. In addition, the `-dD` option is implicitly enabled.

With `-fpreprocessed`, predefinition of command line and most builtin macros is disabled. Macros such as `__LINE__`, which are contextually dependent, are handled normally. This enables compilation of files previously preprocessed with `-E -fdirectives-only`.

With both `-E` and `-fpreprocessed`, the rules for `-fpreprocessed` take precedence. This enables full preprocessing of files previously preprocessed with `-E -fdirectives-only`.

`-fdollars-in-identifiers`

Accept '\$' in identifiers.

`-fextended-identifiers`

Accept universal character names and extended characters in identifiers. This option is enabled by default for C99 (and later C standard versions) and C++.

`-fno-canonical-system-headers`

When preprocessing, do not shorten system header paths with canonicalization.

`-fmax-include-depth=depth`

Set the maximum depth of the nested `#include`. The default is 200.

`-fsearch-include-path[=kind]`

Look for input files on the `#include` path, not just the current directory. This is particularly useful with C++20 modules, for which both header units and module interface units need to be compiled directly:

```
g++ -c -std=c++20 -fmodules -fsearch-include-path bits/stdc++.h bits/std.cc
```

kind defaults to 'user', which looks on the `#include "..."` search path; you can also explicitly specify 'system' for the `#include <...>` search path.

`-ftabstop=width`

Set the distance between tab stops. This helps the preprocessor report correct column numbers in warnings or errors, even if tabs appear on the line. If the value is less than 1 or greater than 100, the option is ignored. The default is 8.

`-ftrack-macro-expansion[=level]`

Track locations of tokens across macro expansions. This allows the compiler to emit diagnostic about the current macro expansion stack when a compilation error occurs in a macro expansion. Using this option makes the preprocessor and the compiler consume more memory. The *level* parameter can be used to choose the level of precision of token location tracking thus decreasing the memory consumption if necessary. Value '0' of *level* de-activates this option. Value '1' tracks tokens locations in a degraded mode for the sake of minimal memory overhead. In this mode all tokens resulting from the expansion of an argument of a function-like macro have the same location. Value '2' tracks tokens locations completely. This value is the most memory hungry. When this option is given no argument, the default parameter value is '2'.

Note that `-ftrack-macro-expansion=2` is activated by default.

`-fmacro-prefix-map=old=new`

When preprocessing files residing in directory *old*, expand the `__FILE__` and `__BASE_FILE__` macros as if the files resided in directory *new* instead. This can be used to change an absolute path to a relative path by using `.` for *new* which can result in more reproducible builds that are location independent. This option also affects `__builtin_FILE()` during compilation. See also `-ffile-prefix-map` and `-fcanon-prefix-map`.

-fexec-charset=charset

Set the execution character set, used for string and character constants. The default is UTF-8. *charset* can be any encoding supported by the system's `iconv` library routine.

-fwide-exec-charset=charset

Set the wide execution character set, used for wide string and character constants. The default is one of UTF-32BE, UTF-32LE, UTF-16BE, or UTF-16LE, whichever corresponds to the width of `wchar_t` and the big-endian or little-endian byte order being used for code generation. As with **-fexec-charset**, *charset* can be any encoding supported by the system's `iconv` library routine; however, you will have problems with encodings that do not fit exactly in `wchar_t`.

-finput-charset=charset

Set the input character set, used for translation from the character set of the input file to the source character set used by GCC. The default is UTF-8. *charset* can be any encoding supported by the system's `iconv` library routine. If the input character set is UTF-8, warnings about ill-formed code unit sequences are issued if **-Winvalid-utf8** is enabled. Otherwise no diagnostics are issued when the input character set matches the execution character set. If they are different, ill-formed code unit sequences result in an error during transcoding to the execution character set.

-fpch-deps

When using precompiled headers (see Section 3.22 [Precompiled Headers], page 555), this flag causes the dependency-output flags to also list the files from the precompiled header's dependencies. If not specified, only the precompiled header are listed and not the files that were used to create it, because those files are not consulted when a precompiled header is used.

-fpch-preprocess

This option allows use of a precompiled header (see Section 3.22 [Precompiled Headers], page 555) together with **-E**. It inserts a special `#pragma`, `#pragma GCC pch_preprocess "filename"` in the output to mark the place where the precompiled header was found, and its *filename*. When **-fpreprocessed** is in use, GCC recognizes this `#pragma` and loads the PCH.

This option is off by default, because the resulting preprocessed output is only really suitable as input to GCC. It is switched on by **-save-temps**.

You should not write this `#pragma` in your own code, but it is safe to edit the filename if the PCH file is available in a different location. The filename may be absolute or it may be relative to GCC's current directory.

-fworking-directory

Enable generation of linemarkers in the preprocessor output that let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor emits, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC uses this directory, when it's present in the preprocessed input, as the directory emitted as the current working directory in some debugging information

formats. This option is implicitly enabled if debugging information is enabled, but this can be inhibited with the negated form `-fno-working-directory`. If the `-P` flag is present in the command line, this option has no effect, since no `#line` directives are emitted whatsoever.

`-C`

`--comments`

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`

`--comments-in-macros`

Do not discard comments, including during macro expansion. This is like `-C`, except that comments contained within macros are also passed through to the output file where the macro is expanded.

In addition to the side effects of the `-C` option, the `-CC` option causes all C++-style comments inside a macro to be converted to C-style comments. This is to prevent later use of that macro from inadvertently commenting out the remainder of the source line.

The `-CC` option is generally used to support lint comments.

`-P`

`--no-line-commands`

Inhibit generation of linemarkers in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code, and will be sent to a program which might be confused by the linemarkers.

`-traditional`

`--traditional`

`-traditional-cpp`

`--traditional-cpp`

Try to imitate the behavior of pre-standard C preprocessors, as opposed to ISO C preprocessors. See the GNU CPP manual for details.

Note that GCC does not otherwise attempt to emulate a pre-standard C compiler, and these options are only supported with the `-E` switch, or when invoking CPP explicitly.

`-trigraphs`

`--trigraphs`

Support ISO C trigraphs. These are three-character sequences, all starting with `'??'`, that are defined by ISO C to stand for single characters. For example, `'??/'` stands for `'\'`, so `'??/n'` is a character constant for a newline.

The nine trigraphs and their replacements are

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

By default, GCC ignores trigraphs, but in standard-conforming modes it converts them. See the `-std` and `-ansi` options.

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

`-H`

`--trace-includes`

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 `...!`.

`-dletters`

`--dump=letters`

`--dump letters`

Says to make debugging dumps during compilation as specified by *letters*. The flags documented here are those relevant to the preprocessor. Other *letters* are interpreted by the compiler proper, or reserved for future versions of GCC, and so are silently ignored. If you specify *letters* whose behavior conflicts, the result is undefined. See Section 3.19 [Developer Options], page 297, for more information.

`-dM`

`--dump=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
```

shows all the predefined macros.

If you use `-dM` without the `-E` option, `-dM` is interpreted as a synonym for `-fdump-rtl-mach`. See Section “Developer Options” in `gcc`.

`-dD`

`--dump=D` Like `-dM` except that it outputs *both* the `#define` directives and the result of preprocessing. Both kinds of output go to the standard output file.

`-dN`

`--dump=N` Like `-dD`, but emit only the macro names, not their expansions.

`-dI`

`--dump=I` Output `#include` directives in addition to the result of preprocessing.

`-dU`

`--dump=U` Like `-dD` except that only macros that are expanded, or whose definedness is tested in preprocessor directives, are output; the output is delayed until the use or test of the macro; and `#undef` directives are also output for macros tested but undefined at the time.

`-fdebug-cpp`

This option is only useful for debugging GCC. When used from CPP or with `-E`, it dumps debugging information about location maps. Every token in the output is preceded by the dump of the map its location belongs to.

When used from GCC without `-E`, this option has no effect.

`-Wp,option`

You can use `-Wp,option` to bypass the compiler driver and pass *option* directly through to the preprocessor. If *option* contains commas, it is split into multiple options at the commas. However, many options are modified, translated or interpreted by the compiler driver before being passed to the preprocessor, and `-Wp` forcibly bypasses this phase. The preprocessor's direct interface is undocumented and subject to change, so whenever possible you should avoid using `-Wp` and let the driver handle the options instead.

`-Xpreprocessor option`

Pass *option* as an option to the preprocessor. You can use this to supply system-specific preprocessor options that GCC does not recognize.

If you want to pass an option that takes an argument, you must use `-Xpreprocessor` twice, once for the option and once for the argument.

`-no-integrated-cpp`

`--no-integrated-cpp`

Perform preprocessing as a separate pass before compilation. By default, GCC performs preprocessing as an integrated part of input tokenization and parsing. If this option is provided, the appropriate language front end (`cc1`, `cc1plus`, or `cc1obj` for C, C++, and Objective-C, respectively) is instead invoked twice, once for preprocessing only and once for actual compilation of the preprocessed input. This option may be useful in conjunction with the `-B` or `-wrapper` options to specify an alternate preprocessor or perform additional processing of the program source between normal preprocessing and compilation.

3.15 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`

`--for-assembler=option`

`--for-assembler option`

Pass *option* as an option to the assembler. You can use this to supply system-specific assembler options that GCC does not recognize.

If you want to pass an option that takes an argument, you must use `-Xassembler` twice, once for the option and once for the argument.

3.16 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 34.

`-flink-libatomic`

Enable linking of libatomic if it's supported by target, and is enabled by default. The negative form `-fno-link-libatomic` can be used to explicitly disable linking of libatomic.

`-flinker-output=type`

This option controls code generation of the link-time optimizer. By default the linker output is automatically determined by the linker plugin. For debugging the compiler and if incremental linking with a non-LTO object file is desired, it may be useful to control the type manually.

If *type* is `'exec'`, code generation produces a static binary. In this case `-fpic` and `-fpie` are both disabled.

If *type* is `'dyn'`, code generation produces a shared library. In this case `-fpic` or `-fPIC` is preserved, but not enabled automatically. This allows to build shared libraries without position-independent code on architectures where this is possible, i.e. on x86.

If *type* is `'pie'`, code generation produces an `-fpie` executable. This results in similar optimizations as `'exec'` except that `-fpie` is not disabled if specified at compilation time.

If *type* is `'rel'`, the compiler assumes that incremental linking is done. The sections containing intermediate code for link-time optimization are merged, pre-optimized, and output to the resulting object file. In addition, if `-ffat-lto-objects` is specified, binary code is produced for future non-LTO linking. The object file produced by incremental linking is smaller than a static library produced from the same object files. At link time the result of incremental linking also loads faster than a static library assuming that the majority of objects in the library are used.

Finally `'nolto-rel'` configures the compiler for incremental linking where code generation is forced, a final binary is produced, and the intermediate code for

later link-time optimization is stripped. When multiple object files are linked together the resulting code is better optimized than with link-time optimizations disabled (for example, cross-module inlining happens), but most of the benefits of whole-program optimizations are lost.

During the incremental link (by `-r`) the linker plugin defaults to `rel`. GNU Binutils 2.44 or later is needed to incrementally link LTO objects and non-LTO objects into a single mixed object file. If any of the object files in an incremental link cannot be used for link-time optimization, the linker plugin issues a warning and uses `'nolto-rel'`. To maintain whole-program optimization, link such objects into a static library instead.

`-fuse-ld=bfd`

Use the `bfd` linker instead of the default linker.

`-fuse-ld=gold`

Use the `gold` linker instead of the default linker.

`-fuse-ld=lld`

Use the LLVM `lld` linker instead of the default linker.

`-fuse-ld=mold`

Use the Modern Linker (`mold`) instead of the default linker.

`-fuse-ld=wild`

Use the Wild linker (`wild`) instead of the default linker.

`-llibrary`

`-l library`

Search the library named *library* when linking. (The second alternative with the library as a separate argument is only for POSIX compliance and is not recommended.)

The `-l` option is passed directly to the linker by GCC. Refer to your linker documentation for exact details. The general description below applies to the GNU linker.

The linker searches a standard list of directories for the library. The directories searched include several standard system directories plus any that you specify with `-L`.

Static libraries are archives of object files, and have file names like `liblibrary.a`. Some targets also support shared libraries, which typically have names like `liblibrary.so`. If both static and shared libraries are found, the linker gives preference to linking with the shared library unless the `-static` option is used.

It makes a difference where in the command you write this option; the linker searches and processes libraries and object files in the order they are specified. Thus, `'foo.o -lz bar.o'` searches library `'z'` after file `foo.o` but before `bar.o`. If `bar.o` refers to functions in `'z'`, those functions may not be loaded.

`-lobjc`

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

-nostartfiles

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

-nodefaultlibs

Do not use the standard system libraries when linking. Only the libraries you specify are passed to the linker, and options specifying linkage of the system libraries, such as **-static-libgcc** or **-shared-libgcc**, are ignored. The standard startup files are used normally, unless **-nostartfiles** is used.

The compiler may generate calls to `memcpy`, `memset`, `memcpy` and `memmove`. These entries are usually resolved by entries in `libc`. These entry points should be supplied through some other mechanism when this option is specified.

-nolibc

Do not use the C library or system libraries tightly coupled with it when linking. Still link with the startup files, `libgcc` or toolchain provided language support libraries such as `libgnat`, `libgfortran` or `libstdc++` unless options preventing their inclusion are used as well. This typically removes `-lc` from the link command line, as well as system libraries that normally go with it and become meaningless when absence of a C library is assumed, for example `-lpthread` or `-lm` in some configurations. This is intended for bare-board targets when there is indeed no C library available.

-nostdlib**--no-standard-libraries**

Do not use the standard system startup files or libraries when linking. No startup files and only the libraries you specify are passed to the linker, and options specifying linkage of the system libraries, such as **-static-libgcc** or **-shared-libgcc**, are ignored.

The compiler may generate calls to `memcpy`, `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. (An example of such an internal subroutine is `__main`, used to ensure C++ constructors are called; see Section “collect2” in *GNU Compiler Collection (GCC) Internals*.)

-nostdlib++

Do not implicitly link with standard C++ libraries.

- e *entry***
- entry=*entry***
- entry *entry***
Specify that the program entry point is *entry*. The argument is interpreted by the linker; the GNU linker accepts either a symbol name or an address.

- pie**
- pie** Produce a dynamically linked position independent executable on targets that support it. For predictable results, you must also specify the same set of options used for compilation (**-fpie**, **-fPIE**, or model suboptions) when you specify this linker option.

- no-pie** Don't produce a dynamically linked position independent executable.

- static-pie**
- static-pie**
Produce a static position independent executable on targets that support it. A static position independent executable is similar to a static executable, but can be loaded at any address without a dynamic linker. For predictable results, you must also specify the same set of options used for compilation (**-fpie**, **-fPIE**, or model suboptions) when you specify this linker option.

- pthread** Link with the POSIX threads library. This option is supported on GNU/Linux targets, most other Unix derivatives, and also on x86 Cygwin and MinGW targets. On some targets this option also sets flags for the preprocessor, so it should be used consistently for both compilation and linking.

- r** Produce a relocatable object as output. This is also known as partial linking.

- rdynamic**
Pass the flag **-export-dynamic** to the ELF linker, on targets that support it. This instructs the linker to add all symbols, not only used ones, to the dynamic symbol table. This option is needed for some uses of **dlopen** or to allow obtaining backtraces from within a program.

- s** Remove all symbol table and relocation information from the executable.

- static**
- static** On systems that support dynamic linking, this overrides **-pie** and prevents linking with the shared libraries. On other systems, this option has no effect.

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

¹ On some systems, '**gcc -shared**' needs to build supplementary stub code for constructors to work. On multi-libbed systems, '**gcc -shared**' must select the correct support libraries to link against. Failing to supply the correct flags may lead to subtle defects. Supplying them in cases where they are not necessary is innocuous. **-shared** suppresses the addition of startup code to alter the floating-point environment as done with **-ffast-math**, **-Ofast** or **-funsafe-math-optimizations** on some targets.

-shared-libgcc

-static-libgcc

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

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

Therefore, the G++ driver automatically adds **-shared-libgcc** whenever you build a shared library or a main executable, because C++ programs typically use exceptions, so this is the right thing to do.

If, instead, you use the GCC driver to create shared libraries, you may find that they are not always linked with the shared `libgcc`. If GCC finds, at its configuration time, that you have a non-GNU linker or a GNU linker that does not support option **--eh-frame-hdr**, it links the shared version of `libgcc` into shared libraries by default. Otherwise, it takes advantage of the linker and optimizes away the linking with the shared version of `libgcc`, linking with the static version of `libgcc` by default. This allows exceptions to propagate through such shared libraries, without incurring relocation costs at library load time.

However, if a library or main executable is supposed to throw or catch exceptions, you must link it using the G++ driver, or using the option **-shared-libgcc**, such that it is linked with the shared `libgcc`.

-static-libasan

When the **-fsanitize=address** option is used to link a program, the GCC driver automatically links against `libasan`. If `libasan` is available as a shared library, and the **-static** option is not used, then this links against the shared version of `libasan`. The **-static-libasan** option directs the GCC driver to link `libasan` statically, without necessarily linking other libraries statically.

-static-libhwasan

When the **-fsanitize=hwaddress** option is used to link a program, the GCC driver automatically links against `libhwasan`. If `libhwasan` is available as a shared library, and the **-static** option is not used, then this links against the shared version of `libhwasan`. The **-static-libhwasan** option directs the GCC driver to link `libhwasan` statically, without necessarily linking other libraries statically.

-static-liblsan

When the **-fsanitize=leak** option is used to link a program, the GCC driver automatically links against `liblsan`. If `liblsan` is available as a shared library, and the **-static** option is not used, then this links against the shared version of `liblsan`. The **-static-liblsan** option directs the GCC driver to link `liblsan` statically, without necessarily linking other libraries statically.

-static-libtsan

When the `-fsanitize=thread` option is used to link a program, the GCC driver automatically links against `libtsan`. If `libtsan` is available as a shared library, and the `-static` option is not used, then this links against the shared version of `libtsan`. The `-static-libtsan` option directs the GCC driver to link `libtsan` statically, without necessarily linking other libraries statically.

-static-libubsan

When the `-fsanitize=undefined` option is used to link a program, the GCC driver automatically links against `libubsan`. If `libubsan` is available as a shared library, and the `-static` option is not used, then this links against the shared version of `libubsan`. The `-static-libubsan` option directs the GCC driver to link `libubsan` statically, without necessarily linking other libraries statically.

-static-libstdc++

When the `g++` program is used to link a C++ program, it normally automatically links against `libstdc++`. If `libstdc++` is available as a shared library, and the `-static` option is not used, then this links against the shared version of `libstdc++`. That is normally fine. However, it is sometimes useful to freeze the version of `libstdc++` used by the program without going all the way to a fully static link. The `-static-libstdc++` option directs the `g++` driver to link `libstdc++` statically, without necessarily linking other libraries statically.

-symbolic**--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`
`--for-linker=option`
`--for-linker 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`
`--force-link=symbol`
`--force-link 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.

`-Tbss=addr`
`-Tdata=addr`
`-Ttext=addr`
`-N`
`-n`
`-t`
`-Z`
`-z keyword`
 These options are passed through to the linker without interpretation by GCC. Refer to your linker documentation for the meanings of these options.

3.17 Options for Directory Search

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

`-I dir`
`-iquote dir`
`-isystem dir`
`-idirafter dir`
`--include-directory-after=dir`
`--include-directory-after dir`
`--include-directory=dir`
`--include-directory dir`
 Add the directory *dir* to the list of directories to be searched for header files during preprocessing. `--include-directory` is an alias for `-I`, while `--include-directory-after` is an alias for `-idirafter`. If *dir* begins with `'=` or `$SYSROOT`, then the `'=` or `$SYSROOT` is replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.

Directories specified with `-iquote` apply only to the quote form of the directive, `#include "file"`. Directories specified with `-I`, `-isystem`, or `-idirafter` apply to lookup for both the `#include "file"` and `#include <file>` directives.

You can specify any number or combination of these options on the command line to search for header files in several directories. The lookup order is as follows:

1. For the quote form of the include directive, the directory of the current file is searched first.
2. For the quote form of the include directive, the directories specified by `-iquote` options are searched in left-to-right order, as they appear on the command line.
3. Directories specified with `-I` options are scanned in left-to-right order.
4. Directories specified with `-isystem` options are scanned in left-to-right order.
5. Standard system directories are scanned.
6. Directories specified with `-idirafter` options are scanned in left-to-right order.

You can use `-I` to override a system header file, substituting your own version, since these directories are searched before the standard system header file directories. However, you should not use this option to add directories that contain vendor-supplied system header files; use `-isystem` for that.

The `-isystem` and `-idirafter` options also mark the directory as a system directory, so that it gets the same special treatment that is applied to the standard system directories.

If a standard system include directory, or a directory specified with `-isystem`, is also specified with `-I`, the `-I` option is ignored. The directory is still searched but as a system directory at its normal position in the system include chain. This is to ensure that GCC's procedure to fix buggy system headers and the ordering for the `#include_next` directive are not inadvertently changed. If you really need to change the search order for system directories, use the `-nostdinc` and/or `-isystem` options.

`-I-`

`--include-barrier`

Split the include path. This option has been deprecated. Please use `-iquote` instead for `-I` directories before the `-I-` and remove the `-I-` option.

Any directories specified with `-I` options before `-I-` are searched only for headers requested with `#include "file"`; they are not searched for `#include <file>`. If additional directories are specified with `-I` options after the `-I-`, those directories are searched for all `'#include'` directives.

In addition, `-I-` inhibits the use of the directory of the current file directory as the first search directory for `#include "file"`. There is no way to override this effect of `-I-`.

`-iprefix prefix`

`--include-prefix=prefix`

`--include-prefix 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`
`--include-with-prefix=prefix`
`--include-with-prefix prefix`
`--include-with-prefix-after=prefix`
`--include-with-prefix-after prefix`
`--include-with-prefix-before=prefix`
`--include-with-prefix-before prefix`
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.
`--include-with-prefix` and `--include-with-prefix-after` are both aliases for `-iwithprefix`, while `--include-with-prefix-before` is an alias for `-iwithprefixbefore`.

`-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.

`-imultiarch dir`
Use *dir* as a subdirectory of the directory containing architecture-specific C++ headers.

`-nostdinc`
`--no-standard-includes`
Do not search the standard system directories for header files. Only the directories explicitly specified with `-I`, `-iquote`, `-isystem`, and/or `-idirafter` options (and the directory of the current file, if appropriate) are searched.

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

`--embed-dir=dir`
`--embed-directory=dir`
`--embed-directory dir`
Append *dir* directory to the list of searched directories for `#embed` preprocessing directive or `__has_embed` macro. There are no default directories for `#embed`.
If *dir* begins with `'='` or `$$SYSROOT`, then the `'='` or `$$SYSROOT` is replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.

`-iplugindir=dir`
Set the directory to search for plugins that are passed by `-fplugin=name` instead of `-fplugin=path/name.so`. This option is not meant to be used by the user, but only passed by the driver.

-Ldir

--library-directory=dir

--library-directory dir

Add directory *dir* to the list of directories to be searched for **-l**.

-Bprefix

--prefix=prefix

--prefix prefix

This option specifies where to find the executables, libraries, include files, and data files of the compiler itself.

The compiler driver program runs one or more of the subprograms **cpp**, **cc1**, **as** and **ld**. It tries *prefix* as a prefix for each program it tries to run, both with and without '*machine/version/*' for the corresponding target machine and compiler version.

For each subprogram to be run, the compiler driver first tries the **-B** prefix, if any. If that name is not found, or if **-B** is not specified, the driver tries two standard prefixes, */usr/lib/gcc/* and */usr/local/lib/gcc/*. If neither of those results in a file name that is found, the unmodified program name is searched for using the directories specified in your **PATH** environment variable.

The compiler checks to see if the path provided by **-B** refers to a directory, and if necessary it adds a directory separator character at the end of the path.

-B prefixes that effectively specify directory names also apply to libraries in the linker, because the compiler translates these options into **-L** options for the linker. They also apply to include files in the preprocessor, because the compiler translates these options into **-isystem** options for the preprocessor. In this case, the compiler appends '*include*' to the prefix.

The runtime support file *libgcc.a* can also be searched for using the **-B** prefix, if needed. If it is not found there, the two standard prefixes above are tried, and that is all. The file is left out of the link if it is not found by those means.

Another way to specify a prefix much like the **-B** prefix is to use the environment variable **GCC_EXEC_PREFIX**. See Section 3.21 [Environment Variables], page 552.

As a special kludge, if the path provided by **-B** is *[dir/]stageN/*, where *N* is a number in the range 0 to 9, then it is replaced by *[dir/]include*. This is to help with boot-strapping the compiler.

-no-canonical-prefixes

--no-canonical-prefixes

Do not expand any symbolic links, resolve references to '*/../*' or '*/./*', or make the path absolute when generating a relative prefix.

--sysroot=dir

--sysroot dir

Use *dir* as the logical root directory for headers and libraries. For example, if the compiler normally searches for headers in */usr/include* and libraries in */usr/lib*, it instead searches *dir/usr/include* and *dir/usr/lib*.

If you use both this option and the **-isysroot** option, then the **--sysroot** option applies to libraries, but the **-isysroot** option applies to header files.

The GNU linker (beginning with version 2.16) has the necessary support for this option. If your linker does not support this option, the header file aspect of `--sysroot` still works, but the library aspect does not.

`--no-sysroot-suffix`

For some targets, a suffix is added to the root directory specified with `--sysroot`, depending on the other options used, so that headers may for example be found in `dir/suffix/usr/include` instead of `dir/usr/include`. This option disables the addition of such a suffix.

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` is `-fno-foo`. In the table below, only one of the forms is listed—the one that is not the default. You can figure out the other form by either removing ‘no-’ or adding it.

`-fstack-reuse=reuse-level`

This option controls stack space reuse for user declared local/auto variables and compiler generated temporaries. *reuse_level* can be ‘all’, ‘named_vars’, or ‘none’. ‘all’ enables stack reuse for all local variables and temporaries, ‘named_vars’ enables the reuse only for user defined local variables with names, and ‘none’ disables stack reuse completely. The default value is ‘all’. The option is needed when the program extends the lifetime of a scoped local variable or a compiler generated temporary beyond the end point defined by the language. When a lifetime of a variable ends, and if the variable lives in memory, the optimizing compiler has the freedom to reuse its stack space with other temporaries or scoped local variables whose live range does not overlap with it. Legacy code extending local lifetime is likely to break with the stack reuse optimization.

For example,

```
int *p;
{
    int local1;

    p = &local1;
    local1 = 10;
    ....
}
{
    int local2;
    local2 = 20;
    ...
}

if (*p == 10) // out of scope use of local1
{
    ...
}
```

Another example:

```

struct A
{
    A(int k) : i(k), j(k) { }
    int i;
    int j;
};

A *ap;

void foo(const A& ar)
{
    ap = &ar;
}

void bar()
{
    foo(A(10)); // temp object's lifetime ends when foo returns

    {
        A a(20);
        ....
    }
    ap->i+= 10; // ap references out of scope temp whose space
               // is reused with a. What is the value of ap->i?
}

```

The lifetime of a compiler generated temporary is well defined by the C++ standard. When a lifetime of a temporary ends, and if the temporary lives in memory, the optimizing compiler has the freedom to reuse its stack space with other temporaries or scoped local variables whose live range does not overlap with it. However some of the legacy code relies on the behavior of older compilers in which temporaries' stack space is not reused, the aggressive stack reuse can lead to runtime errors. This option is used to control the temporary stack reuse optimization.

-ftrapv This option generates traps for signed overflow on addition, subtraction, multiplication operations. The options **-ftrapv** and **-fwrapv** override each other, so using **-ftrapv -fwrapv** on the command-line results in **-fwrapv** being effective. Note that only active options override, so using **-ftrapv -fwrapv -fno-wrapv** on the command-line results in **-ftrapv** being effective.

-fwrapv This option instructs the compiler to assume that signed arithmetic overflow of addition, subtraction and multiplication wraps around using twos-complement representation. This flag enables some optimizations and disables others. The options **-ftrapv** and **-fwrapv** override each other, so using **-ftrapv -fwrapv** on the command-line results in **-fwrapv** being effective. Note that only active options override, so using **-ftrapv -fwrapv -fno-wrapv** on the command-line results in **-ftrapv** being effective.

-fwrapv-pointer

This option instructs the compiler to assume that pointer arithmetic overflow on addition and subtraction wraps around using twos-complement representation. This flag disables some optimizations which assume pointer overflow is invalid.

-fstrict-overflow

This option implies **-fno-wrapv** **-fno-wrapv-pointer** and when negated implies **-fwrapv** **-fwrapv-pointer**.

-fexceptions

Enable exception handling. Generates extra code needed to propagate exceptions. For some targets, this implies GCC generates frame unwind information for all functions, which can produce significant data size overhead, although it does not affect execution. If you do not specify this option, GCC enables it by default for languages like C++ that normally require exception handling, and disables it for languages like C that do not normally require it. However, you may need to enable this option when compiling C code that needs to interoperate properly with exception handlers written in C++. You may also wish to disable this option if you are compiling older C++ programs that don't use exception handling.

-fnon-call-exceptions

Generate code that allows trapping instructions to throw exceptions. Note that this requires platform-specific runtime support that does not exist everywhere. Moreover, it only allows *trapping* instructions to throw exceptions, i.e. memory references or floating-point instructions. It does not allow exceptions to be thrown from arbitrary signal handlers such as **SIGALRM**. This enables **-fexceptions**.

-fdelete-dead-exceptions

Consider that instructions that may throw exceptions but don't otherwise contribute to the execution of the program can be optimized away. This does not affect calls to functions except those with the **pure** or **const** attributes. This option is enabled by default for the Ada and C++ compilers, as permitted by the language specifications. Optimization passes that cause dead exceptions to be removed are enabled independently at different optimization levels.

-funwind-tables

Similar to **-fexceptions**, except that it just generates any needed static data, but does not affect the generated code in any other way. You normally do not need to enable this option; instead, a language processor that needs this handling enables it on your behalf.

-fasynchronous-unwind-tables

Generate unwind table in DWARF format, if supported by target machine. The table is exact at each instruction boundary, so it can be used for stack unwinding from asynchronous events (such as debugger or garbage collector).

-fno-gnu-unique

On systems with recent GNU assembler and C library, the C++ compiler uses the **STB_GNU_UNIQUE** binding to make sure that definitions of template static data members and static local variables in inline functions are unique even in the presence of **RTLD_LOCAL**; this is necessary to avoid problems with a library used by two different **RTLD_LOCAL** plugins depending on a definition in one of them and therefore disagreeing with the other one about the binding of the

symbol. But this causes `dlclose` to be ignored for affected DSOs; if your program relies on reinitialization of a DSO via `dlclose` and `dlopen`, you can use `-fno-gnu-unique`.

`-fpcc-struct-return`

Return “short” `struct` and `union` values in memory like longer ones, rather than in registers. This convention is less efficient, but it has the advantage of allowing intercallability between GCC-compiled files and files compiled with other compilers, particularly the Portable C Compiler (`pcc`).

The precise convention for returning structures in memory depends on the target configuration macros.

Short structures and unions are those whose size and alignment match that of some integer type.

Warning: code compiled with the `-fpcc-struct-return` switch is not binary compatible with code compiled with the `-freg-struct-return` switch. Use it to conform to a non-default application binary interface.

`-freg-struct-return`

Return `struct` and `union` values in registers when possible. This is more efficient for small structures than `-fpcc-struct-return`.

If you specify neither `-fpcc-struct-return` nor `-freg-struct-return`, GCC defaults to whichever convention is standard for the target. If there is no standard convention, GCC defaults to `-fpcc-struct-return`, except on targets where GCC is the principal compiler. In those cases, we can choose the standard, and we chose the more efficient register return alternative.

Warning: code compiled with the `-freg-struct-return` switch is not binary compatible with code compiled with the `-fpcc-struct-return` switch. Use it to conform to a non-default application binary interface.

`-fshort-enums`

Allocate to an `enum` type only as many bytes as it needs for the declared range of possible values. Specifically, the `enum` type is equivalent to the smallest integer type that has enough room. This option has no effect for an enumeration type with a fixed underlying type.

Warning: the `-fshort-enums` switch causes GCC to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non-default application binary interface.

`-fshort-wchar`

Override the underlying type for `wchar_t` to be `short unsigned int` instead of the default for the target. This option is useful for building programs to run under WINE.

Warning: the `-fshort-wchar` switch causes GCC to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non-default application binary interface.

-fcommon In C code, this option controls the placement of global variables defined without an initializer, known as *tentative definitions* in the C standard. Tentative

definitions are distinct from declarations of a variable with the `extern` keyword, which do not allocate storage.

The default is `-fno-common`, which specifies that the compiler places uninitialized global variables in the BSS section of the object file. This inhibits the merging of tentative definitions by the linker so you get a multiple-definition error if the same variable is accidentally defined in more than one compilation unit.

The `-fcommon` places uninitialized global variables in a common block. This allows the linker to resolve all tentative definitions of the same variable in different compilation units to the same object, or to a non-tentative definition. This behavior is inconsistent with C++, and on many targets implies a speed and code size penalty on global variable references. It is mainly useful to enable legacy code to link without errors.

`-fno-ident`

`-Qy`

`-Qn` `-fno-ident` suppresses emission of `.ident` assembler directives and causes the `#ident` preprocessor directive to be ignored. `-Qy` and `-Qn` are obsolete synonyms for `-fident` and `-fno-ident`, respectively.

`-finhibit-size-directive`

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

`-fverbose-asm`

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

`-fno-verbose-asm`, the default, causes the extra information to be omitted and is useful when comparing two assembler files.

The added comments include:

- information on the compiler version and command-line options,
- the source code lines associated with the assembly instructions, in the form `FILENAME:LINENUMBER:CONTENT OF LINE`,
- hints on which high-level expressions correspond to the various assembly instruction operands.

For example, given this C source file:

```
int test (int n)
{
    int i;
    int total = 0;

    for (i = 0; i < n; i++)
        total += i * i;
```

```
    return total;
}
```

compiling to (x86_64) assembly via `-S` and emitting the result direct to stdout via `-o -`

```
gcc -S test.c -fverbose-asm -Os -o -
```

gives output similar to this:

```
.file "test.c"
# GNU C11 (GCC) version 7.0.0 20160809 (experimental) (x86_64-pc-linux-gnu)
[...snip...]
# options passed:
[...snip...]

.text
.globl test
.type test, @function
test:
.LFB0:
.cfi_startproc
# test.c:4:  int total = 0;
xorl %eax, %eax # <retval>
# test.c:6:  for (i = 0; i < n; i++)
xorl %edx, %edx # i
.L2:
# test.c:6:  for (i = 0; i < n; i++)
cmpl %edi, %edx # n, i
jge .L5 #,
# test.c:7:      total += i * i;
movl %edx, %ecx # i, tmp92
imull %edx, %ecx # i, tmp92
# test.c:6:  for (i = 0; i < n; i++)
incl %edx # i
# test.c:7:      total += i * i;
addl %ecx, %eax # tmp92, <retval>
jmp .L2 #
.L5:
# test.c:10: }
ret
.cfi_endproc
.LFE0:
.size test, .-test
.ident "GCC: (GNU) 7.0.0 20160809 (experimental)"
.section .note.GNU-stack,"",@progbits
```

The comments are intended for humans rather than machines and hence the precise format of the comments is subject to change.

-frecord-gcc-switches

This switch causes the command line used to invoke the compiler to be recorded into the object file that is being created. This switch is only implemented on some targets and the exact format of the recording is target and binary file format dependent, but it usually takes the form of a section containing ASCII text. This switch is related to the `-fverbose-asm` switch, but that switch only records information in the assembler output file as comments, so it never reaches the object file. See also `-grecord-gcc-switches` for another way of storing compiler options into the object file.

- fpic** Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine. Such code accesses all constant addresses through a global offset table (GOT). The dynamic loader resolves the GOT entries when the program starts (the dynamic loader is not part of GCC; it is part of the operating system). If the GOT size for the linked executable exceeds a machine-specific maximum size, you get an error message from the linker indicating that **-fpic** does not work; in that case, recompile with **-fPIC** instead. (These maximums are 8k on the SPARC, 28k on AArch64 and 32k on the m68k and RS/6000. The x86 has no such limit.)
- Position-independent code requires special support, and therefore works only on certain machines. For the x86, GCC supports PIC for System V but not for the Sun 386i. Code generated for the IBM RS/6000 is always position-independent. When this flag is set, the macros `__pic__` and `__PIC__` are defined to 1.
- fPIC** If supported for the target machine, emit position-independent code, suitable for dynamic linking and avoiding any limit on the size of the global offset table. This option makes a difference on AArch64, m68k, PowerPC and SPARC.
- Position-independent code requires special support, and therefore works only on certain machines.
- When this flag is set, the macros `__pic__` and `__PIC__` are defined to 2.
- fpie**
-fPIE These options are similar to **-fpic** and **-fPIC**, but the generated position-independent code can be only linked into executables. Usually these options are used to compile code that will be linked using the **-pie** GCC option.
- fpie** and **-fPIE** both define the macros `__pie__` and `__PIE__`. The macros have the value 1 for **-fpie** and 2 for **-fPIE**.
- fno-plt** Do not use the PLT for external function calls in position-independent code. Instead, load the callee address at call sites from the GOT and branch to it. This leads to more efficient code by eliminating PLT stubs and exposing GOT loads to optimizations. On architectures such as 32-bit x86 where PLT stubs expect the GOT pointer in a specific register, this gives more register allocation freedom to the compiler. Lazy binding requires use of the PLT; with **-fno-plt** all external symbols are resolved at load time.
- Alternatively, the function attribute `nopl` can be used to avoid calls through the PLT for specific external functions.
- In position-dependent code, a few targets also convert calls to functions that are marked to not use the PLT to use the GOT instead.
- fno-jump-tables** Do not use jump tables for switch statements even where it would be more efficient than other code generation strategies. This option is of use in conjunction with **-fpic** or **-fPIC** for building code that forms part of a dynamic linker and cannot reference the address of a jump table. On some targets, jump tables do not require a GOT and this option is not needed.

-fno-bit-tests

Do not use bit tests for switch statements even where it would be more efficient than other code generation strategies.

-ffixed-reg

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

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

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

-fcall-used-reg

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

It is an error to use this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model produces disastrous results.

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

-fcall-saved-reg

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

It is an error to use this flag with the frame pointer or stack pointer. Use of this flag for other registers that have fixed pervasive roles in the machine's execution model produces disastrous results.

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

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

-fpack-struct[=*n*]

Without a value specified, pack all structure members together without holes. When a value is specified (which must be a small power of two), pack structure members according to this value, representing the maximum alignment (that is, objects with default alignment requirements larger than this are output potentially unaligned at the next fitting location).

Warning: the `-fpack-struct` switch causes GCC to generate code that is not binary compatible with code generated without that switch. Additionally, it makes the code suboptimal. Use it to conform to a non-default application binary interface.

-fleading-underscore

This option and its counterpart, `-fno-leading-underscore`, forcibly change the way C symbols are represented in the object file. One use is to help link with legacy assembly code.

Warning: the `-fleading-underscore` switch causes GCC to generate code that is not binary compatible with code generated without that switch. Use it to conform to a non-default application binary interface. Not all targets provide complete support for this switch.

`-ftls-model=model`

Alter the thread-local storage model to be used (see Section 6.6 [Thread-Local], page 715). The *model* argument should be one of ‘global-dynamic’, ‘local-dynamic’, ‘initial-exec’ or ‘local-exec’. Note that the choice is subject to optimization: the compiler may use a more efficient model for symbols not visible outside of the translation unit, or if `-fpic` is not given on the command line.

The default without `-fpic` is ‘initial-exec’; with `-fpic` the default is ‘global-dynamic’.

`-ftrampolines`

For targets that normally need trampolines for nested functions, always generate them instead of using descriptors. Otherwise, for targets that do not need them, like for example HP-PA or IA-64, do nothing.

A trampoline is a small piece of code that is created at run time on the stack when the address of a nested function is taken, and is used to call the nested function indirectly. Therefore, it requires the stack to be made executable in order for the program to work properly.

`-fno-trampolines` is enabled by default on a language by language basis to let the compiler avoid generating them, if it computes that this is safe, and replace them with descriptors. Descriptors are made up of data only, but the generated code must be prepared to deal with them. As of this writing, `-fno-trampolines` is enabled by default only for Ada.

Moreover, code compiled with `-ftrampolines` and code compiled with `-fno-trampolines` are not binary compatible if nested functions are present. This option must therefore be used on a program-wide basis and be manipulated with extreme care.

For languages other than Ada, the `-ftrampolines` and `-fno-trampolines` options currently have no effect, and trampolines are always generated on platforms that need them for nested functions.

`-ftrampoline-impl=[stack|heap]`

By default, trampolines are generated on stack. However, certain platforms (such as the Apple M1) do not permit an executable stack. Compiling with `-ftrampoline-impl=heap` generate calls to `__gcc_nested_func_ptr_created` and `__gcc_nested_func_ptr_deleted` in order to allocate and deallocate trampoline space on the executable heap. These functions are implemented in `libgcc`, and will only be provided on specific targets: x86_64 Darwin, x86_64 and aarch64 Linux. *PLEASE NOTE:* Heap trampolines are *not* guaranteed to be correctly deallocated if you `setjmp`, instantiate nested functions, and then `longjmp` back to a state prior to having allocated those nested functions.

`-fvisibility=[default|internal|hidden|protected]`

Set the default ELF image symbol visibility to the specified option—all symbols are marked with this unless overridden within the code. Using this feature can very substantially improve linking and load times of shared object libraries, produce more optimized code, provide near-perfect API export and prevent symbol clashes. It is **strongly** recommended that you use this in any shared objects you distribute.

Despite the nomenclature, ‘**default**’ always means public; i.e., available to be linked against from outside the shared object. ‘**protected**’ and ‘**internal**’ are pretty useless in real-world usage so the only other commonly used option is ‘**hidden**’. The default if `-fvisibility` isn’t specified is ‘**default**’, i.e., make every symbol public.

A good explanation of the benefits offered by ensuring ELF symbols have the correct visibility is given by “How To Write Shared Libraries” by Ulrich Drepper (which can be found at <https://www.akkadia.org/drepper/>)—however a superior solution made possible by this option to marking things hidden when the default is public is to make the default hidden and mark things public. This is the norm with DLLs on Windows and with `-fvisibility=hidden` and `__attribute__((visibility("default")))` instead of `__declspec(dllexport)` you get almost identical semantics with identical syntax. This is a great boon to those working with cross-platform projects.

For those adding visibility support to existing code, you may find `#pragma GCC visibility` of use. This works by you enclosing the declarations you wish to set visibility for with (for example) `#pragma GCC visibility push(hidden)` and `#pragma GCC visibility pop`. Bear in mind that symbol visibility should be viewed **as part of the API interface contract** and thus all new code should always specify visibility when it is not the default; i.e., declarations only for use within the local DSO should **always** be marked explicitly as hidden as so to avoid PLT indirection overheads—making this abundantly clear also aids readability and self-documentation of the code. Note that due to ISO C++ specification requirements, `operator new` and `operator delete` must always be of default visibility.

Be aware that headers from outside your project, in particular system headers and headers from any other library you use, may not be expecting to be compiled with visibility other than the default. You may need to explicitly say `#pragma GCC visibility push(default)` before including any such headers.

`extern` declarations are not affected by `-fvisibility`, so a lot of code can be recompiled with `-fvisibility=hidden` with no modifications. However, this means that calls to `extern` functions with no explicit visibility use the PLT, so it is more effective to use `__attribute__((visibility))` and/or `#pragma GCC visibility` to tell the compiler which `extern` declarations should be treated as hidden.

Note that `-fvisibility` does affect C++ vague linkage entities. This means that, for instance, an exception class that is be thrown between DSOs must

be explicitly marked with default visibility so that the ‘`type_info`’ nodes are unified between the DSOs.

An overview of these techniques, their benefits and how to use them is at <https://gcc.gnu.org/wiki/Visibility>.

-fstrict-volatile-bitfields

This option should be used if accesses to volatile bit-fields (or other structure fields, although the compiler usually honors those types anyway) should use a single access of the width of the field’s type, aligned to a natural alignment if possible. For example, targets with memory-mapped peripheral registers might require all such accesses to be 16 bits wide; with this flag you can declare all peripheral bit-fields as **unsigned short** (assuming short is 16 bits on these targets) to force GCC to use 16-bit accesses instead of, perhaps, a more efficient 32-bit access.

If this option is disabled, the compiler uses the most efficient instruction. In the previous example, that might be a 32-bit load instruction, even though that accesses bytes that do not contain any portion of the bit-field, or memory-mapped registers unrelated to the one being updated.

In some cases, such as when the **packed** attribute is applied to a structure field, it may not be possible to access the field with a single read or write that is correctly aligned for the target machine. In this case GCC falls back to generating multiple accesses rather than code that will fault or truncate the result at run time.

Note: Due to restrictions of the C/C++11 memory model, write accesses are not allowed to touch non bit-field members. It is therefore recommended to define all bits of the field’s type as bit-field members.

The default value of this option is determined by the application binary interface for the target processor.

-fsync-libcalls

This option controls whether any out-of-line instance of the **__sync** family of functions may be used to implement the C++11 **__atomic** family of functions.

The default value of this option is enabled, thus the only useful form of the option is **-fno-sync-libcalls**. This option is used in the implementation of the **libatomic** runtime library.

-fzero-init-padding-bits=value

Guarantee zero initialization of padding bits in automatic variable initializers. Certain languages guarantee zero initialization of padding bits in certain cases, e.g. C23 when using empty initializers (**{}**), or C++ when using zero-initialization or C guarantees that fields not specified in an initializer have their padding bits zero initialized. This option allows to change when padding bits in initializers are guaranteed to be zero initialized. The default is **-fzero-init-padding-bits=standard**, which makes no further guarantees than the corresponding standard. E.g.

```
struct A { char a; unsigned long long b; char c; };
union B { char a; unsigned long long b; };
struct A a = {}; // C23 guarantees padding bits are zero.
```

```

struct A b = { 1, 2, 3 }; // No guarantees.
union B c = {}; // C23 guarantees padding bits are zero.
union B d = { 1 }; // No guarantees.

```

-fzero-init-padding-bits=unions guarantees zero initialization of padding bits in unions on top of what the standards guarantee, if the initializer of an union is empty (then all bits of the union are zero initialized) or if the initialized member of the union is smaller than the size of the union (in that case guarantees padding bits outside of the initialized member to be zero initialized). This was the GCC behavior before GCC 15 and in the above example guarantees zero initialization of last `sizeof (unsigned long long) - 1` bytes in the union.

-fzero-init-padding-bits=all guarantees additionally zero initialization of padding bits of other aggregates, so the padding in between `b.a` and `b.b` (if any) and tail padding in the structure (if any).

3.19 GCC Developer Options

This section describes command-line options that are primarily of interest to GCC developers, including options to support compiler testing and investigation of compiler bugs and compile-time performance problems. This includes options that produce debug dumps at various points in the compilation; that print statistics such as memory use and execution time; and that print information about GCC's configuration, such as where it searches for libraries. You should rarely need to use any of these options for ordinary compilation and linking tasks.

Many developer options that cause GCC to dump output to a file take an optional `'=filename'` suffix. You can specify `'stdout'` or `'-'` to dump to standard output, and `'stderr'` for standard error.

If `'=filename'` is omitted, a default dump file name is constructed by concatenating the base dump file name, a pass number, phase letter, and pass name. The base dump file name is the name of output file produced by the compiler if explicitly specified and not an executable; otherwise it is the source file name. The pass number is determined by the order passes are registered with the compiler's pass manager. This is generally the same as the order of execution, but passes registered by plugins, target-specific passes, or passes that are otherwise registered late are numbered higher than the pass named `'final'`, even if they are executed earlier. The phase letter is one of `'i'` (inter-procedural analysis), `'l'` (language-specific), `'r'` (RTL), or `'t'` (tree). The files are created in the directory of the output file.

-fcallgraph-info

-fcallgraph-info=MARKERS

Makes the compiler output callgraph information for the program, on a per-object-file basis. The information is generated in the common VCG format. It can be decorated with additional, per-node and/or per-edge information, if a list of comma-separated markers is additionally specified. When the `su` marker is specified, the callgraph is decorated with stack usage information; it is equivalent to **-fstack-usage**. When the `da` marker is specified, the callgraph is decorated with information about dynamically allocated objects.

When compiling with `-flto`, no callgraph information is output along with the object file. At LTO link time, `-fcallgraph-info` may generate multiple callgraph information files next to intermediate LTO output files.

```
-dletters
--dump=letters
--dump letters
-fdump-rtl-pass
-fdump-rtl-pass-options
-fdump-rtl-pass-options=filename
```

Says to make debugging dumps during compilation at times specified by *letters* when using `-d` or by *pass* when using `-fdump-rtl`. This is used for debugging the RTL-based passes of the compiler.

Some `-dletters` switches have different meaning when `-E` is used for preprocessing. See Section 3.14 [Preprocessor Options], page 267, for information about preprocessor-specific dump options.

The ‘*options*’ form allows greater control over the details of the dump. See `-fdump-tree`.

Here are actual instances of command-line options following these patterns 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-comp_gotos`
Dump after duplicating the computed gotos.

`-fdump-rtl-ce1`
`-fdump-rtl-ce2`
`-fdump-rtl-ce3`
`-fdump-rtl-ce1`, `-fdump-rtl-ce2`, and `-fdump-rtl-ce3` enable dumping after the three if conversion passes.

`-fdump-rtl-cprop_hardreg`
Dump after hard register copy propagation.

`-fdump-rtl-csa`
Dump after combining stack adjustments.

`-fdump-rtl-cse1`
`-fdump-rtl-cse2`
`-fdump-rtl-cse1` and `-fdump-rtl-cse2` enable dumping after the two common subexpression elimination passes.

`-fdump-rtl-dce`
Dump after the standalone dead code elimination passes.

`-fdump-rtl-dbr`
Dump after delayed branch scheduling.

`-fdump-rtl-dce1`
`-fdump-rtl-dce2`
`-fdump-rtl-dce1` and `-fdump-rtl-dce2` enable dumping after the two dead store elimination passes.

`-fdump-rtl-eh`
Dump after finalization of EH handling code.

`-fdump-rtl-eh_ranges`
Dump after conversion of EH handling range regions.

`-fdump-rtl-expand`
Dump after RTL generation.

`-fdump-rtl-fwprop1`
`-fdump-rtl-fwprop2`
`-fdump-rtl-fwprop1` and `-fdump-rtl-fwprop2` enable dumping after the two forward propagation passes.

`-fdump-rtl-gcse1`
`-fdump-rtl-gcse2`
`-fdump-rtl-gcse1` and `-fdump-rtl-gcse2` enable dumping after global common subexpression elimination.

`-fdump-rtl-init_regs`
Dump after the initialization of the registers.

`-fdump-rtl-initvals`
Dump after the computation of the initial value sets.

`-fdump-rtl-into_cfglayout`
Dump after converting to cfglayout mode.

`-fdump-rtl-ira`
Dump after iterated register allocation.

`-fdump-rtl-jump`
Dump after the second jump optimization.

`-fdump-rtl-loop2`
`-fdump-rtl-loop2` enables dumping after the rtl loop optimization passes.

`-fdump-rtl-mach`
Dump after performing the machine dependent reorganization pass, if that pass exists.

`-fdump-rtl-mode_sw`
Dump after removing redundant mode switches.

`-fdump-rtl-rnreg`
Dump after register renumbering.

`-fdump-rtl-outof_cfglayout`
Dump after converting from cfglayout mode.

`-fdump-rtl-peephole2`
Dump after the peephole pass.

`-fdump-rtl-postreload`
Dump after post-reload optimizations.

`-fdump-rtl-pro_and_epilogue`
Dump after generating the function prologues and epilogues.

`-fdump-rtl-sched1`
`-fdump-rtl-sched2`
`-fdump-rtl-sched1` and `-fdump-rtl-sched2` enable dumping after the basic block scheduling passes.

`-fdump-rtl-ree`
Dump after sign/zero extension elimination.

`-fdump-rtl-seqabstr`
Dump after common sequence discovery.

`-fdump-rtl-shorten`
Dump after shortening branches.

`-fdump-rtl-split1`
`-fdump-rtl-split2`
`-fdump-rtl-split3`
`-fdump-rtl-split4`
`-fdump-rtl-split5`
These options enable dumping after five rounds of instruction splitting.

- `-fdump-rtl-sms`
Dump after modulo scheduling. This pass is only run on some architectures.
- `-fdump-rtl-stack`
Dump after conversion from GCC's "flat register file" registers to the x87's stack-like registers. This pass is only run on x86 variants.
- `-fdump-rtl-subreg1`
- `-fdump-rtl-subreg2`
`-fdump-rtl-subreg1` and `-fdump-rtl-subreg2` enable dumping after the two subreg expansion passes.
- `-fdump-rtl-vartrack`
Dump after variable tracking.
- `-fdump-rtl-vregs`
Dump after converting virtual registers to hard registers.
- `-fdump-rtl-web`
Dump after live range splitting.
- `-fdump-rtl-regclass`
- `-fdump-rtl-subregs_of_mode_init`
- `-fdump-rtl-subregs_of_mode_finish`
- `-fdump-rtl-dfinit`
- `-fdump-rtl-dfinish`
These dumps are defined but always produce empty files.
- `-da`
- `--dump=a`
- `-fdump-rtl-all`
Produce all the dumps listed above.
- `-dA`
- `--dump=A` Annotate the assembler output with miscellaneous debugging information.
- `-dD`
- `--dump=D` Dump all macro definitions, at the end of preprocessing, in addition to normal output.
- `-dH`
- `--dump=H` Produce a core dump whenever an error occurs.
- `-dp`
- `--dump=p` Annotate the assembler output with a comment indicating which pattern and alternative is used. The length and cost of each instruction are also printed.
- `-dP`
- `--dump=P` Dump the RTL in the assembler output as a comment before each instruction. Also turns on `-dp` annotation.

- `-dx`
- `--dump=x` Just generate RTL for a function instead of compiling it. Usually used with `-fdump-rtl-expand`.
- `-fdump-debug`
Dump debugging information generated during the debug generation phase.
- `-fdump-earlydebug`
Dump debugging information generated during the early debug generation phase.
- `-fdump-noaddr`
When doing debugging dumps, suppress address output. This makes it more feasible to use `diff` on debugging dumps for compiler invocations with different compiler binaries and/or different text / bss / data / heap / stack / dso start locations.
- `-freport-bug`
Collect and dump debug information into a temporary file if an internal compiler error (ICE) occurs.
- `-fdump-unnumbered`
When doing debugging dumps, suppress instruction numbers and address output. This makes it more feasible to use `diff` on debugging dumps for compiler invocations with different options, in particular with and without `-g`.
- `-fdump-unnumbered-links`
When doing debugging dumps (see `-d` option above), suppress instruction numbers for the links to the previous and next instructions in a sequence.
- `-fdump-internal-locations`
Dump detailed information about GCC's internal representation of source code locations.
- `-fdump-ipa-switch`
- `-fdump-ipa-switch-options`
Control the dumping at various stages of inter-procedural analysis language tree to a file. The file name is generated by appending a switch specific suffix to the source file name, and the file is created in the same directory as the output file. The following dumps are possible:
 - `'all'` Enables all inter-procedural analysis dumps.
 - `'cgraph'` Dumps information about call-graph optimization, unused function removal, and inlining decisions.
 - `'inline'` Dump after function inlining.
 - `'strubm'` Dump after selecting `strub` modes, and recording the selections as function attributes.
 - `'strub'` Dump `strub` transformations: interface changes, function wrapping, and insertion of builtin calls for stack scrubbing and water-marking.

Additionally, the options `-optimized`, `-missed`, `-note`, and `-all` can be provided, with the same meaning as for `-fopt-info`, defaulting to `-optimized`.

For example, `-fdump-ipa-inline-optimized-missed` will emit information on callsites that were inlined, along with callsites that were not inlined.

By default, the dump will contain messages about successful optimizations (equivalent to `-optimized`) together with low-level details about the analysis.

`-fdump-ipa-clones`

Create a dump file containing information about creation of call graph node clones and removals of call graph nodes during inter-procedural optimizations and transformations. Its main intended use is that tools that create live-patches can determine the set of functions that need to be live-patched to completely replace a particular function (see `-flive-patching`). The file name is generated by appending suffix `ipa-clones` to the source file name, and the file is created in the same directory as the output file. Each entry in the file is on a separate line containing semicolon separated fields.

In the case of call graph clone creation, the individual fields are:

1. String **Callgraph clone**.
2. Name of the function being cloned as it is presented to the assembler.
3. A number that uniquely represents the function being cloned in the call graph. Note that the number is unique only within a compilation unit or within whole-program analysis but is likely to be different in the two phases.
4. The file name of the source file where the function is defined.
5. The line on which the function definition is located.
6. The column where the function definition is located.
7. Name of the new function clone as it is presented to the assembler.
8. A number that uniquely represents the new function clone in the call graph. Note that the number is unique only within a compilation unit or within whole-program analysis but is likely to be different in the two phases.
9. The file name of the source file where the source code location of the new clone points to.
10. The line to which the source code location of the new clone points to.
11. The column to which the source code location of the new clone points to.
12. A string that determines the reason for cloning.

In the case of call graph clone removal, the individual fields are:

1. String **Callgraph removal**.
2. Name of the function being removed as it would be presented to the assembler.
3. A number that uniquely represents the function being cloned in the call graph. Note that the number is unique only within a compilation unit or within whole-program analysis but is likely to be different in the two phases.

4. The file name of the source file where the function is defined.
5. The line on which the function definition is located.
6. The column where the function definition is located.

-fdump-lang

Dump language-specific information. The file name is made by appending *.lang* to the source file name.

-fdump-lang-all**-fdump-lang-switch****-fdump-lang-switch-options****-fdump-lang-switch-options=filename**

Control the dumping of language-specific information. The *options* and *filename* portions behave as described in the **-fdump-tree** option. **-fdump-tree-all** enables all language-specific dumps; other options vary with the language. For instance, see Section 3.5 [C++ Dialect Options], page 52, for the **-fdump-lang** flags supported by the C++ front-end.

-fdump-passes

Print on **stderr** the list of optimization passes that are turned on and off by the current command-line options.

-fdump-statistics-option

Enable and control dumping of pass statistics in a separate file. The file name is generated by appending a suffix ending in *‘.statistics’* to the source file name, and the file is created in the same directory as the output file. If the *‘-option’* form is used, *‘-stats’* causes counters to be summed over the whole compilation unit while *‘-details’* dumps every event as the passes generate them. The default with no option is to sum counters for each function compiled.

-fdump-tree-all**-fdump-tree-switch****-fdump-tree-switch-options****-fdump-tree-switch-options=filename**

Control the dumping at various stages of processing the intermediate language tree to a file. If the *‘-options’* form is used, *options* is a list of *‘-’* separated options which control the details of the dump. Not all options are applicable to all dumps; those that are not meaningful are ignored. The following options are available

‘address’ Print the address of each node. Usually this is not meaningful as it changes according to the environment and source file. Its primary use is for tying up a dump file with a debug environment.

‘asmname’ If **DECL_ASSEMBLER_NAME** has been set for a given decl, use that in the dump instead of **DECL_NAME**. Its primary use is ease of use working backward from mangled names in the assembly file.

‘slim’ When dumping front-end intermediate representations, inhibit dumping of members of a scope or body of a function merely

because that scope has been reached. Only dump such items when they are directly reachable by some other path.

When dumping pretty-printed trees, this option inhibits dumping the bodies of control structures.

When dumping RTL, print the RTL in slim (condensed) form instead of the default LISP-like representation.

<code>'raw'</code>	Print a raw representation of the tree. By default, trees are pretty-printed into a C-like representation.
<code>'details'</code>	Enable more detailed dumps (not honored by every dump option). Also include information from the optimization passes.
<code>'stats'</code>	Enable dumping various statistics about the pass (not honored by every dump option).
<code>'blocks'</code>	Enable showing basic block boundaries (disabled in raw dumps).
<code>'graph'</code>	For each of the other indicated dump files (<code>-fdump-rtl-pass</code>), dump a representation of the control flow graph suitable for viewing with GraphViz to <code>file.passid.pass.dot</code> . Each function in the file is pretty-printed as a subgraph, so that GraphViz can render them all in a single plot. RTL is always dumped in slim form.
<code>'vops'</code>	Enable showing virtual operands for every statement.
<code>'lineno'</code>	Enable showing line numbers for statements.
<code>'uid'</code>	Enable showing the unique ID (<code>DECL_UID</code>) for each variable.
<code>'verbose'</code>	Enable showing the tree dump for each statement.
<code>'eh'</code>	Enable showing the EH region number holding each statement.
<code>'scev'</code>	Enable showing scalar evolution analysis details.
<code>'optimized'</code>	Enable showing optimization information (only available in certain passes).
<code>'missed'</code>	Enable showing missed optimization information (only available in certain passes).
<code>'note'</code>	Enable other detailed optimization information (only available in certain passes).
<code>'folding'</code>	Enable dumping information about match-and-simplify (match.pd) patterns, when they are applied.
<code>'all'</code>	Turn on all options, except <code>raw</code> , <code>slim</code> , <code>verbose</code> and <code>lineno</code> .
<code>'optall'</code>	Turn on all optimization options, i.e., <code>optimized</code> , <code>missed</code> , and <code>note</code> .

To determine what tree dumps are available or find the dump for a pass of interest follow the steps below.

1. Invoke GCC with `-fdump-passes` and in the `stderr` output look for a code that corresponds to the pass you are interested in. For example, the codes `tree-evrp`, `tree-vrp1`, and `tree-vrp2` correspond to the three Value Range Propagation passes. The number at the end distinguishes distinct invocations of the same pass.
2. To enable the creation of the dump file, append the pass code to the `-fdump-` option prefix and invoke GCC with it. For example, to enable the dump from the Early Value Range Propagation pass, invoke GCC with the `-fdump-tree-evrp` option. Optionally, you may specify the name of the dump file. If you don't specify one, GCC creates as described below.
3. Find the pass dump in a file whose name is composed of three components separated by a period: the name of the source file GCC was invoked to compile, a numeric suffix indicating the pass number followed by the letter 't' for tree passes (and the letter 'r' for RTL passes), and finally the pass code. For example, the Early VRP pass dump might be in a file named `myfile.c.038t.evrp` in the current working directory. Note that the numeric codes are not stable and may change from one version of GCC to another.

`-fopt-info`

`-fopt-info-options`

`-fopt-info-options=filename`

Controls optimization dumps from various optimization passes. If the '`-options`' form is used, *options* is a list of '-' separated option keywords to select the dump details and optimizations.

The *options* can be divided into three groups:

1. options describing what kinds of messages should be emitted,
2. options describing the verbosity of the dump, and
3. options describing which optimizations should be included.

The options from each group can be freely mixed as they are non-overlapping. However, in case of any conflicts, the later options override the earlier options on the command line.

The following options control which kinds of messages should be emitted:

'`optimized`'

Print information when an optimization is successfully applied. It is up to a pass to decide which information is relevant. For example, the vectorizer passes print the source location of loops which are successfully vectorized.

'`missed`'

Print information about missed optimizations. Individual passes control which information to include in the output.

'`note`'

Print verbose information about optimizations, such as certain transformations, more detailed messages about decisions etc.

‘all’ Print detailed optimization information. This includes ‘optimized’, ‘missed’, and ‘note’.

The following option controls the dump verbosity:

‘internals’ By default, only “high-level” messages are emitted. This option enables additional, more detailed, messages, which are likely to only be of interest to GCC developers.

One or more of the following option keywords can be used to describe a group of optimizations:

‘ipa’ Enable dumps from all interprocedural optimizations.
 ‘loop’ Enable dumps from all loop optimizations.
 ‘inline’ Enable dumps from all inlining optimizations.
 ‘omp’ Enable dumps from all OMP (Offloading and Multi Processing) optimizations.
 ‘vec’ Enable dumps from all vectorization optimizations.
 ‘optall’ Enable dumps from all optimizations. This is a superset of the optimization groups listed above.

If *options* is omitted, it defaults to ‘optimized-optall’, which means to dump messages about successful optimizations from all the passes, omitting messages that are treated as “internals”.

If the *filename* is provided, then the dumps from all the applicable optimizations are concatenated into the *filename*. Otherwise the dump is output onto **stderr**. Though multiple **-fopt-info** options are accepted, only one of them can include a *filename*. If other filenames are provided then all but the first such option are ignored.

Note that the output *filename* is overwritten in case of multiple translation units. If a combined output from multiple translation units is desired, **stderr** should be used instead.

In the following example, the optimization info is output to **stderr**:

```
gcc -O3 -fopt-info
```

This example:

```
gcc -O3 -fopt-info-missed=missed.all
```

outputs missed optimization report from all the passes into **missed.all**, and this one:

```
gcc -O2 -ftree-vectorize -fopt-info-vec-missed
```

prints information about missed optimization opportunities from vectorization passes on **stderr**. Note that **-fopt-info-vec-missed** is equivalent to **-fopt-info-missed-vec**. The order of the optimization group names and message types listed after **-fopt-info** does not matter.

As another example,

```
gcc -O3 -fopt-info-inline-optimized-missed=inline.txt
```

outputs information about missed optimizations as well as optimized locations from all the inlining passes into `inline.txt`.

Finally, consider:

```
gcc -fopt-info-vec-missed=vec.miss -fopt-info-loop-optimized=loop.opt
```

Here the two output filenames `vec.miss` and `loop.opt` are in conflict since only one output file is allowed. In this case, only the first option takes effect and the subsequent options are ignored. Thus only `vec.miss` is produced which contains dumps from the vectorizer about missed opportunities.

-fsave-optimization-record

Write a `SRCFILE.opt-record.json.gz` file detailing what optimizations were performed, for those optimizations that support `-fopt-info`.

This option is experimental and the format of the data within the compressed JSON file is subject to change.

It is roughly equivalent to a machine-readable version of `-fopt-info-all`, as a collection of messages with source file, line number and column number, with the following additional data for each message:

- the execution count of the code being optimized, along with metadata about whether this was from actual profile data, or just an estimate, allowing consumers to prioritize messages by code hotness,
- the function name of the code being optimized, where applicable,
- the “inlining chain” for the code being optimized, so that when a function is inlined into several different places (which might themselves be inlined), the reader can distinguish between the copies,
- objects identifying those parts of the message that refer to expressions, statements or symbol-table nodes, which of these categories they are, and, when available, their source code location,
- the GCC pass that emitted the message, and
- the location in GCC’s own code from which the message was emitted

Additionally, some messages are logically nested within other messages, reflecting implementation details of the optimization passes.

-fsched-verbose=n

On targets that use instruction scheduling, this option controls the amount of debugging output the scheduler prints to the dump files.

For n greater than zero, `-fsched-verbose` outputs the same information as `-fdump-rtl-sched1` and `-fdump-rtl-sched2`. For n greater than one, it also output basic block probabilities, detailed ready list information and unit/insn info. For n greater than two, it includes RTL at abort point, control-flow and regions info. And for n over four, `-fsched-verbose` also includes dependence info.

-fenable-kind-pass

-fdisable-kind-pass=range-list

This is a set of options that are used to explicitly disable/enable optimization passes. These options are intended for use for debugging GCC. Compiler users should use regular options for enabling/disabling passes instead.

-fdisable-ipa-pass

Disable IPA pass *pass*. *pass* is the pass name. If the same pass is statically invoked in the compiler multiple times, the pass name should be appended with a sequential number starting from 1.

-fdisable-rtl-pass**-fdisable-rtl-pass=*range-list***

Disable RTL pass *pass*. *pass* is the pass name. If the same pass is statically invoked in the compiler multiple times, the pass name should be appended with a sequential number starting from 1. *range-list* is a comma-separated list of function ranges or assembler names. Each range is a number pair separated by a colon. The range is inclusive in both ends. If the range is trivial, the number pair can be simplified as a single number. If the function's call graph node's *uid* falls within one of the specified ranges, the *pass* is disabled for that function. The *uid* is shown in the function header of a dump file, and the pass names can be dumped by using option **-fdump-passes**.

-fdisable-tree-pass**-fdisable-tree-pass=*range-list***

Disable tree pass *pass*. See **-fdisable-rtl** for the description of option arguments.

-fenable-ipa-pass

Enable IPA pass *pass*. *pass* is the pass name. If the same pass is statically invoked in the compiler multiple times, the pass name should be appended with a sequential number starting from 1.

-fenable-rtl-pass**-fenable-rtl-pass=*range-list***

Enable RTL pass *pass*. See **-fdisable-rtl** for option argument description and examples.

-fenable-tree-pass**-fenable-tree-pass=*range-list***

Enable tree pass *pass*. See **-fdisable-rtl** for the description of option arguments.

Here are some examples showing uses of these options.

```
# disable ccp1 for all functions
-fdisable-tree-ccp1
# disable complete unroll for function whose cgraph node uid is 1
-fenable-tree-cunroll=1
# disable gcse2 for functions at the following ranges [1,1],
# [300,400], and [400,1000]
# disable gcse2 for functions foo and foo2
-fdisable-rtl-gcse2=foo,foo2
# disable early inlining
-fdisable-tree-einline
# disable ipa inlining
-fdisable-ipa-inline
```

```
# enable tree full unroll
-fenable-tree-unroll
```

-fchecking

-fchecking=*n*

Enable internal consistency checking. The default depends on the compiler configuration. **-fchecking=2** enables further internal consistency checking that might affect code generation.

-frandom-seed=*string*

This option provides a seed that GCC uses in place of random numbers in generating certain symbol names that have to be different in every compiled file. It is also used to place unique stamps in coverage data files and the object files that produce them. You can use the **-frandom-seed** option to produce reproducibly identical object files.

The *string* can either be a number (decimal, octal or hex) or an arbitrary string (in which case it's converted to a number by computing CRC32).

The *string* should be different for every file you compile.

-save-temps

--save-temps

Store the usual “temporary” intermediate files permanently; name them as auxiliary output files, as specified described under **-dumpbase** and **-dumpdir**.

When used in combination with the **-x** command-line option, **-save-temps** is sensible enough to avoid overwriting an input source file with the same extension as an intermediate file. The corresponding intermediate file may be obtained by renaming the source file before using **-save-temps**.

-save-temps=cwd

Equivalent to **-save-temps -dumpdir ./**.

-save-temps=obj

Equivalent to **-save-temps -dumpdir outdir/**, where *outdir/* is the directory of the output file specified after the **-o** option, including any directory separators. If the **-o** option is not used, the **-save-temps=obj** switch behaves like **-save-temps=cwd**.

-specs=*file*

--specs=*file*

--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. See Section “Specifying Subprocesses and the Switches to Pass to Them” in *GNU Compiler Collection (GCC) Internals*, for information about the format of the *file*.

-time[=file]

Report the CPU time taken by each subprocess in the compilation sequence. For C source files, this is the compiler proper and assembler (plus the linker if linking is done).

Without the specification of an output file, the output looks like this:

```
# cc1 0.12 0.01
# as 0.00 0.01
```

The first number on each line is the “user time”, that is time spent executing the program itself. The second number is “system time”, time spent executing operating system routines on behalf of the program. Both numbers are in seconds.

With the specification of an output file, the output is appended to the named file, and it looks like this:

```
0.12 0.01 cc1 options
0.00 0.01 as options
```

The “user time” and the “system time” are moved before the program name, and the options passed to the program are displayed, so that one can later tell what file was being compiled, and with which options.

-fdump-final-insns[=file]

Dump the final internal representation (RTL) to *file*. If the optional argument is omitted (or if *file* is *.*), the name of the dump file is determined by appending *.gkd* to the dump base name, see **-dumpbase**.

-fcompare-debug[=opts]

If no error occurs during compilation, run the compiler a second time, adding *opts* and **-fcompare-debug-second** to the arguments passed to the second compilation. Dump the final internal representation in both compilations, and print an error if they differ.

If the equal sign is omitted, the default **-gtoggle** is used.

The environment variable `GCC_COMPARE_DEBUG`, if defined, non-empty and nonzero, implicitly enables **-fcompare-debug**. If `GCC_COMPARE_DEBUG` is defined to a string starting with a dash, then it is used for *opts*, otherwise the default **-gtoggle** is used.

-fcompare-debug=, with the equal sign but without *opts*, is equivalent to **-fno-compare-debug**, which disables the dumping of the final representation and the second compilation, preventing even `GCC_COMPARE_DEBUG` from taking effect.

To verify full coverage during **-fcompare-debug** testing, set `GCC_COMPARE_DEBUG` to say **-fcompare-debug-not-overridden**, which GCC rejects as an invalid option in any actual compilation (rather than preprocessing, assembly or linking). To get just a warning, setting `GCC_COMPARE_DEBUG` to `‘-w%n-fcompare-debug not overridden’` will do.

-fcompare-debug-second

This option is implicitly passed to the compiler for the second compilation requested by **-fcompare-debug**, along with options to silence warnings, and omitting other options that would cause the compiler to produce output to files

or to standard output as a side effect. Dump files and preserved temporary files are renamed so as to contain the `.gk` additional extension during the second compilation, to avoid overwriting those generated by the first.

When this option is passed to the compiler driver, it causes the *first* compilation to be skipped, which makes it useful for little other than debugging the compiler proper.

-gtoggle Turn off generation of debug info, if leaving out this option generates it, or turn it on at level 2 otherwise. The position of this argument in the command line does not matter; it takes effect after all other options are processed, and it does so only once, no matter how many times it is given. This is mainly intended to be used with `-fcompare-debug`.

-fvar-tracking-assignments-toggle
Toggle `-fvar-tracking-assignments`, in the same way that `-gtoggle` toggles `-g`.

-Q When used on the command line prior to `--help=`, `-Q` acts as a modifier to the help output. See Section 3.2 [Overall Options], page 34, for details about `--help=`.

Otherwise, this option makes the compiler print out each function name as it is compiled, and print some statistics about each pass when it finishes.

-ftime-report
Makes the compiler print some statistics to stderr about the time consumed by each pass when it finishes.

If SARIF output of diagnostics was requested via `-fdiagnostics-format=sarif-file` or `-fdiagnostics-format=sarif-stderr` then the `-ftime-report` information is instead emitted in JSON form as part of SARIF output. The precise format of this JSON data is subject to change, and the values may not exactly match those emitted to stderr due to being written out at a slightly different place within the compiler.

-ftime-report-details
Record the time consumed by infrastructure parts separately for each pass.

-fira-verbose=*n*
Control the verbosity of the dump file for the integrated register allocator. The default value is 5. If the value *n* is greater or equal to 10, the dump output is sent to stderr using the same format as *n* minus 10.

-flto-report
Prints a report with internal details on the workings of the link-time optimizer. The contents of this report vary from version to version. It is meant to be useful to GCC developers when processing object files in LTO mode (via `-flto`).
Disabled by default.

-flto-report-wpa
Like `-flto-report`, but only print for the WPA phase of link-time optimization.

-fmem-report

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

-fmem-report-wpa

Makes the compiler print some statistics about permanent memory allocation for the WPA phase only.

-fpre-ipa-mem-report**-fpost-ipa-mem-report**

Makes the compiler print some statistics about permanent memory allocation before or after interprocedural optimization.

-fmultiflags

This option enables multilib-aware **TFLAGS** to be used to build target libraries with options different from those the compiler is configured to use by default, through the use of specs (see Section “Specifying Subprocesses and the Switches to Pass to Them” in *GNU Compiler Collection (GCC) Internals*) set up by compiler internals, by the target, or by builders at configure time.

Like **TFLAGS**, this allows the target libraries to be built for portable baseline environments, while the compiler defaults to more demanding ones. That’s useful because users can easily override the defaults the compiler is configured to use to build their own programs, if the defaults are not ideal for their target environment, whereas rebuilding the runtime libraries is usually not as easy or desirable.

Unlike **TFLAGS**, the use of specs enables different flags to be selected for different multilibs. The way to accomplish that is to build with ‘make **TFLAGS**=-fmultiflags’, after configuring ‘--with-specs=%{fmultiflags:...}’.

This option is discarded by the driver once it’s done processing driver self spec. It is also useful to check that **TFLAGS** are being used to build all target libraries, by configuring a non-bootstrap compiler ‘--with-specs='%{!fmultiflags:%emissing **TFLAGS**}’’ and building the compiler and target libraries.

-fprofile-report

Makes the compiler print some statistics about consistency of the (estimated) profile and effect of individual passes.

-fstack-usage

Makes the compiler output stack usage information for the program, on a per-function basis. The filename for the dump is made by appending **.su** to the *auxname*. *auxname* is generated from the name of the output file, if explicitly specified and it is not an executable, otherwise it is the basename of the source file. An entry is made up of four fields separated by tabulation characters:

- The name of the function preceded by its source location
- The mangled name of the function
- A number of bytes

- One or more qualifiers: `static`, `dynamic`, `bounded`

The qualifier `static` means that the function manipulates the stack statically: a fixed number of bytes are allocated for the frame on function entry and released on function exit; no stack adjustments are otherwise made in the function. The second field is this fixed number of bytes.

The qualifier `dynamic` means that the function manipulates the stack dynamically: in addition to the static allocation described above, stack adjustments are made in the body of the function, for example to push/pop arguments around function calls. If the qualifier `bounded` is also present, the amount of these adjustments is bounded at compile time and the second field is an upper bound of the total amount of stack used by the function. If it is not present, the amount of these adjustments is not bounded at compile time and the second field only represents the bounded part.

-fstats Emit statistics about front-end processing at the end of the compilation. This option is supported only by the C++ front end, and the information is generally only useful to the G++ development team.

-fdbg-cnt-list

Print the name and the counter upper bound for all debug counters.

-fdbg-cnt=counter-value-list

Set the internal debug counter lower and upper bound. *counter-value-list* is a comma-separated list of *name:lower_bound1-upper_bound1* [*:lower_bound2-upper_bound2...*] tuples which sets the name of the counter and list of closed intervals. The *lower_bound* is optional and is zero initialized if not set. For example, with `-fdbg-cnt=dce:2-4:10-11,tail_call:10`, `dbg_cnt(dce)` returns true only for second, third, fourth, tenth and eleventh invocation. For `dbg_cnt(tail_call)` true is returned for first 10 invocations.

-print-autofdo-gcov-version

--print-autofdo-gcov-version

Print the current version of GCOV being used by the AutoFDO infrastructure.

-print-file-name=library

--print-file-name=library

--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-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-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-multi-os-directory`

Print the path to OS libraries for the selected multilib, relative to some `lib` subdirectory. If OS libraries are present in the `lib` subdirectory and no multilibs are used, this is usually just `.`, if OS libraries are present in `libsuffix` sibling directories this prints e.g. `../lib64`, `../lib` or `../lib32`, or if OS libraries are present in `lib/subdir` subdirectories it prints e.g. `amd64`, `sparcv9` or `ev6`.

`-print-multiarch`

`--print-multiarch`

Print the path to OS libraries for the selected multiarch, relative to some `lib` subdirectory.

`-print-prog-name=program`

`--print-prog-name=program`

`--print-prog-name program`

Like `-print-file-name`, but searches for a program such as `cpp`.

`-print-libgcc-file-name`

`--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-search-dirs`

Print the name of the configured installation directory and a list of program and library directories `gcc` searches—and don’t do anything else.

This is useful when `gcc` prints the error message ‘**installation problem, cannot exec cpp0: No such file or directory**’. To resolve this you either need to put `cpp0` and the other compiler components where `gcc` expects to find them, or you can set the environment variable `GCC_EXEC_PREFIX` to the directory where you installed them. Don’t forget the trailing ‘/’. See Section 3.21 [Environment Variables], page 552.

`-print-sysroot`

`--print-sysroot`

Print the target sysroot directory that is used during compilation. This is the target sysroot specified either at configure time or using the `--sysroot` option, possibly with an extra suffix that depends on compilation options. If no target sysroot is specified, the option prints nothing.

`-print-sysroot-headers-suffix`

`--print-sysroot-headers-suffix`

Print the suffix added to the target sysroot when searching for headers, or give an error if the compiler is not configured with such a suffix—and don’t do anything else.

-dumpmachine

Print the compiler’s target machine (for example, ‘i686-pc-linux-gnu’)—and don’t do anything else.

-dumpversion

Print the compiler version (for example, 3.0, 6.3.0 or 7)—and don’t do anything else. This is the compiler version used in filesystem paths and specs. Depending on how the compiler has been configured it can be just a single number (major version), two numbers separated by a dot (major and minor version) or three numbers separated by dots (major, minor and patchlevel version).

-dumpfullversion

Print the full compiler version—and don’t do anything else. The output is always three numbers separated by dots, major, minor and patchlevel version.

-dumpspecs

Print the compiler’s built-in specs—and don’t do anything else. (This is used when GCC itself is being built.) See Section “Specifying Subprocesses and the Switches to Pass to Them” in *GNU Compiler Collection (GCC) Internals*.

--param name=value**--param=name=value**

GCC by convention uses parameters that can be specified on the command line instead of hard-wired constants to represent arbitrary compiler limits or heuristics. Many parameters are related to optimization; for example, GCC does not inline functions that contain more than a certain number of instructions. The static analyzer similarly uses parameters to limit complexity, link-time optimization uses parameters to control partitioning, and so on. Other parameters control aspects of GCC that are completely internal, such as its memory allocation and garbage collection strategy. Still others control target-specific behavior.

The **--param** option provides a uniform interface for specifying values for these compiler parameters. However, 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. You should not depend on parameter settings for correct compilation of your program. They are exposed via the command line for the convenience of developers in debugging compilation problems or, in some cases, to provide workarounds for compiler bugs.

See Section “Parameters” in *GNU Compiler Collection (GCC) Internals*, for documentation of these internal parameters.

3.20 Target-Specific Options

Each target machine supported by GCC can have its own options—for example, to allow you to compile for a particular processor variant or ABI, or to control optimizations specific to that machine. Similarly, GCC also has options that are specific to particular operating systems or runtime environments on the target.

By convention, the names of machine-specific options start with ‘-m’. Some configurations of the compiler also support additional target-specific options, usually for compatibility with other compilers on the same platform.

3.20.1 AArch64 Options

These options are defined for AArch64 implementations:

-mabi=name

Generate code for the specified data model. Permissible values are ‘ilp32’ for SysV-like data model where int, long int and pointers are 32 bits, and ‘lp64’ for SysV-like data model where int is 32 bits, but long int and pointers are 64 bits.

The default depends on the specific target configuration. Note that the LP64 and ILP32 ABIs are not link-compatible; you must compile your entire program with the same ABI, and link with a compatible set of libraries.

The ‘ilp32’ model is deprecated.

-mbig-endian

Generate big-endian code. This is the default when GCC is configured for an ‘aarch64_be-*-’ target.

-mlittle-endian

Generate little-endian code. This is the default when GCC is configured for an ‘aarch64-*-’ but not an ‘aarch64_be-*-’ target.

-menable-sysreg-checking

Generates an error message if an attempt is made to access a system register which is not available on the target architecture.

-mgeneral-regs-only

Generate code that uses only the general-purpose registers. This prevents the compiler from using floating-point and Advanced SIMD registers but does not impose any restrictions on the assembler.

-mcmodel=tiny

Generate code for the tiny code model. The program and its statically defined symbols must be within 1MB of each other. Programs can be statically or dynamically linked.

-mcmodel=small

Generate code for the small code model. The program and its statically defined symbols must be within 4GB of each other. Programs can be statically or dynamically linked. This is the default code model.

-mcmodel=large

Generate code for the large code model. This makes no assumptions about addresses and sizes of sections. Programs can be statically linked only. The **-mcmodel=large** option is incompatible with **-mabi=ilp32**, **-fpic** and **-fPIC**.

-mtp=name

Specify the system register to use as a thread pointer. The valid values are ‘tpidr_el0’, ‘tpidrro_el0’, ‘tpidr_el1’, ‘tpidr_el2’, ‘tpidr_el3’. For back-

wards compatibility the aliases ‘e10’, ‘e11’, ‘e12’, ‘e13’ are also accepted. The default setting is ‘tpidr_e10’. It is recommended to compile all code intended to interoperate with the same value of this option to avoid accessing a different thread pointer from the wrong exception level.

-mstrict-align

-mno-strict-align

Avoid or allow generating memory accesses that may not be aligned on a natural object boundary as described in the architecture specification.

-momit-leaf-frame-pointer

-mno-omit-leaf-frame-pointer

Omit or keep the frame pointer in leaf functions. The former behavior is the default.

-mstack-protector-guard=guard

-mstack-protector-guard-reg=reg

-mstack-protector-guard-offset=offset

Generate stack protection code using canary at *guard*. Supported locations are ‘global’ for a global canary or ‘sysreg’ for a canary in an appropriate system register.

With the latter choice the options **-mstack-protector-guard-reg=reg** and **-mstack-protector-guard-offset=offset** furthermore specify which system register to use as base register for reading the canary, and from what offset from that base register. There is no default register or offset as this is entirely for use within the Linux kernel.

-mtls-dialect=desc

Use TLS descriptors as the thread-local storage mechanism for dynamic accesses of TLS variables. This is the default.

-mtls-dialect=traditional

Use traditional TLS as the thread-local storage mechanism for dynamic accesses of TLS variables.

-mtls-size=size

Specify bit size of immediate TLS offsets. Valid values are 12, 24, 32, 48. This option requires binutils 2.26 or newer.

-mfix-cortex-a53-835769

-mno-fix-cortex-a53-835769

Enable or disable the workaround for the ARM Cortex-A53 erratum number 835769. This involves inserting a NOP instruction between memory instructions and 64-bit integer multiply-accumulate instructions. This flag will be ignored if an architecture or cpu is specified on the command line which does not need the workaround.

-mfix-cortex-a53-843419

-mno-fix-cortex-a53-843419

Enable or disable the workaround for the ARM Cortex-A53 erratum number 843419. This erratum workaround is made at link time and this will only pass

the corresponding flag to the linker. This flag will be ignored if an architecture or cpu is specified on the command line which does not need the workaround.

-m~~low~~-precision-~~recip~~-sqrt

-mno-~~low~~-precision-~~recip~~-sqrt

Enable or disable the reciprocal square root approximation. This option only has an effect if **-ffast-math** or **-funsafe-math-optimizations** is used as well. Enabling this reduces precision of reciprocal square root results to about 16 bits for single precision and to 32 bits for double precision.

-m~~low~~-precision-sqrt

-mno-~~low~~-precision-sqrt

Enable or disable the square root approximation. This option only has an effect if **-ffast-math** or **-funsafe-math-optimizations** is used as well. Enabling this reduces precision of square root results to about 16 bits for single precision and to 32 bits for double precision. If enabled, it implies **-m~~low~~-precision-~~recip~~-sqrt**.

-m~~low~~-precision-div

-mno-~~low~~-precision-div

Enable or disable the division approximation. This option only has an effect if **-ffast-math** or **-funsafe-math-optimizations** is used as well. Enabling this reduces precision of division results to about 16 bits for single precision and to 32 bits for double precision.

-mtrack-speculation

-mno-track-speculation

Enable or disable generation of additional code to track speculative execution through conditional branches. The tracking state can then be used by the compiler when expanding calls to `__builtin_speculation_safe_value` to permit a more efficient code sequence to be generated.

-moutline-atomics

-mno-outline-atomics

Enable or disable calls to out-of-line helpers to implement atomic operations. These helpers will, at runtime, determine if the LSE instructions from ARMv8.1-A can be used; if not, they will use the load/store-exclusive instructions that are present in the base ARMv8.0 ISA.

This option is only applicable when compiling for the base ARMv8.0 instruction set. If using a later revision, e.g. **-march=armv8.1-a** or **-march=armv8-a+lse**, the ARMv8.1-Atomics instructions will be used directly. The same applies when using **-mcpu=** when the selected cpu supports the 'lse' feature. This option is on by default.

-mmax-vectorization

-mno-max-vectorization

Enable or disable an override to vectorizer cost model making vectorization always appear profitable. This option can be combined with **-mautovec-preference** allowing precise control over which ISA will be used for auto-vectorization. Unlike **-fno-vect-cost-model** or **-fvect-cost-**

`model=unlimited` this option does not turn off cost comparison between different vector modes.

-mautovec-preference=name

Force an ISA selection strategy for auto-vectorization. The possible values of *name* are:

‘default’ Use the default heuristics.

‘asimd-only’

Use only Advanced SIMD for auto-vectorization.

‘sve-only’

Use only SVE for auto-vectorization.

‘prefer-asimd’

Use both Advanced SIMD and SVE. Prefer Advanced SIMD when the costs are deemed equal.

‘prefer-sve’

Use both Advanced SIMD and SVE. Prefer SVE when the costs are deemed equal.

For best performance it is highly recommended to use `-mcpu` or `-mtune` instead. This parameter should only be used for code exploration.

-march=name

Specify the name of the target architecture and, optionally, one or more feature modifiers. This option has the form `-march=arch{+[no]feature}*.`

The table below summarizes the permissible values for *arch* and the features that they enable by default:

<i>arch</i> value	Architecture	Includes by default
‘armv8-a’	Armv8-A	‘+fp’, ‘+simd’
‘armv8.1-a’	Armv8.1-A	‘armv8-a’, ‘+crc’, ‘+lse’, ‘+rdma’
‘armv8.2-a’	Armv8.2-A	‘armv8.1-a’
‘armv8.3-a’	Armv8.3-A	‘armv8.2-a’, ‘+pauth’, ‘+fcma’, ‘+jscvt’
‘armv8.4-a’	Armv8.4-A	‘armv8.3-a’, ‘+flagm’, ‘+fp16fml’, ‘+dotprod’, ‘+rcpc2’
‘armv8.5-a’	Armv8.5-A	‘armv8.4-a’, ‘+sb’, ‘+ssbs’, ‘+predres’, ‘+frintts’, ‘+flagm2’
‘armv8.6-a’	Armv8.6-A	‘armv8.5-a’, ‘+bf16’, ‘+i8mm’
‘armv8.7-a’	Armv8.7-A	‘armv8.6-a’, ‘+wfxt’, ‘+xs’
‘armv8.8-a’	Armv8.8-a	‘armv8.7-a’, ‘+mops’
‘armv8.9-a’	Armv8.9-a	‘armv8.8-a’
‘armv9-a’	Armv9-A	‘armv8.5-a’, ‘+sve’, ‘+sve2’
‘armv9.1-a’	Armv9.1-A	‘armv9-a’, ‘+bf16’, ‘+i8mm’
‘armv9.2-a’	Armv9.2-A	‘armv9.1-a’, ‘+wfxt’, ‘+xs’
‘armv9.3-a’	Armv9.3-A	‘armv9.2-a’, ‘+mops’
‘armv9.4-a’	Armv9.4-A	‘armv9.3-a’, ‘+sve2p1’
‘armv9.5-a’	Armv9.5-A	‘armv9.4-a’, ‘cpa’, ‘+faminmax’, ‘+lut’
‘armv8-r’	Armv8-R	‘armv8-r’

The value `'native'` is available on native AArch64 GNU/Linux and causes the compiler to pick the architecture of the host system. This option has no effect if the compiler is unable to recognize the architecture of the host system. When `-march=native` is given and no other `-mcpu` or `-mtune` is given then GCC will pick the host CPU as the CPU to tune for as well as select the architecture features from. That is, `-march=native` is treated as `-mcpu=native`.

The permissible values for *feature* are listed in the sub-section on `[-march and -mcpu Feature Modifiers]`, page 324. Where conflicting feature modifiers are specified, the right-most feature is used.

GCC uses *name* to determine what kind of instructions it can emit when generating assembly code. If `-march` is specified without either of `-mtune` or `-mcpu` also being specified, the code is tuned to perform well across a range of target processors implementing the target architecture.

`-mtune=name`

Specify the name of the target processor for which GCC should tune the performance of the code. Permissible values for this option are: `'generic'`, `'cortex-a35'`, `'cortex-a53'`, `'cortex-a55'`, `'cortex-a57'`, `'cortex-a72'`, `'cortex-a73'`, `'cortex-a75'`, `'cortex-a76'`, `'cortex-a76ae'`, `'cortex-a77'`, `'cortex-a65'`, `'cortex-a65ae'`, `'cortex-a34'`, `'cortex-a78'`, `'cortex-a78ae'`, `'cortex-a78c'`, `'ares'`, `'exynos-m1'`, `'emag'`, `'falkor'`, `'oryon-1'`, `'neoverse-512tvb'`, `'neoverse-e1'`, `'neoverse-n1'`, `'neoverse-n2'`, `'neoverse-v1'`, `'neoverse-v2'`, `'grace'`, `'neoverse-v3'`, `'neoverse-v3ae'`, `'armagicpu'`, `'neoverse-n3'`, `'olympus'`, `'cortex-a725'`, `'cortex-x925'`, `'qdf24xx'`, `'saphira'`, `'phedca'`, `'xgene1'`, `'vulcan'`, `'octeonx'`, `'octeonx81'`, `'octeonx83'`, `'octeonx2'`, `'octeonx2t98'`, `'octeonx2t96'`, `'octeonx2t93'`, `'octeonx2f95'`, `'octeonx2f95n'`, `'octeonx2f95mm'`, `'a64fx'`, `'fujitsu-monaka'`, `'thunderx'`, `'thunderxt88'`, `'thunderxt88p1'`, `'thunderxt81'`, `'tsv110'`, `'hip12'`, `'thunderxt83'`, `'thunderx2t99'`, `'thunderx3t110'`, `'zeus'`, `'cortex-a57.cortex-a53'`, `'cortex-a72.cortex-a53'`, `'cortex-a73.cortex-a35'`, `'cortex-a73.cortex-a53'`, `'cortex-a75.cortex-a55'`, `'cortex-a76.cortex-a55'`, `'cortex-r82'`, `'cortex-r82ae'`, `'cortex-x1'`, `'cortex-x1c'`, `'cortex-x2'`, `'cortex-x3'`, `'cortex-x4'`, `'cortex-a510'`, `'cortex-a520'`, `'cortex-a520ae'`, `'cortex-a710'`, `'cortex-a715'`, `'cortex-a720'`, `'cortex-a720ae'`, `'ampere1'`, `'ampere1a'`, `'ampere1b'`, `'ampere1c'`, `'cobalt-100'`, `'apple-m1'`, `'apple-m2'`, `'apple-m3'`, `'apple-m4'`, `'apple-m5'`, `'c1-nano'`, `'c1-pro'`, `'c1-premium'`, `'c1-ultra'` and `'native'`.

The values `'cortex-a57.cortex-a53'`, `'cortex-a72.cortex-a53'`, `'cortex-a73.cortex-a35'`, `'cortex-a73.cortex-a53'`, `'cortex-a75.cortex-a55'`, `'cortex-a76.cortex-a55'`, `'apple-m1'`, `'apple-m2'`, `'apple-m3'`, `'gb10'` specify that GCC should tune for a big.LITTLE system.

The value `'neoverse-512tvb'` specifies that GCC should tune for Neoverse cores that (a) implement SVE and (b) have a total vector bandwidth of 512 bits per cycle. In other words, the option tells GCC to tune for Neoverse cores that can execute 4 128-bit Advanced SIMD arithmetic instructions a cycle and that can execute an equivalent number of SVE arithmetic instructions per cycle (2 for

256-bit SVE, 4 for 128-bit SVE). This is more general than tuning for a specific core like Neoverse V1 but is more specific than the default tuning described below.

Additionally on native AArch64 GNU/Linux systems the value ‘**native**’ tunes performance to the host system. This option has no effect if the compiler is unable to recognize the processor of the host system.

Where none of `-mtune=`, `-mcpu=` or `-march=` are specified, the code is tuned to perform well across a range of target processors.

This option cannot be suffixed by feature modifiers.

`-mcpu=name`

Specify the name of the target processor, optionally suffixed by one or more feature modifiers. This option has the form `-mcpu=cpu{+[no]feature}*1`, where the permissible values for *cpu* are the same as those available for `-mtune`. The permissible values for *feature* are documented in the sub-section on [`-march` and `-mcpu` Feature Modifiers], page 324. Where conflicting feature modifiers are specified, the right-most feature is used.

GCC uses *name* to determine what kind of instructions it can emit when generating assembly code (as if by `-march`) and to determine the target processor for which to tune for performance (as if by `-mtune`). Where this option is used in conjunction with `-march` or `-mtune`, those options take precedence over the appropriate part of this option.

`-mcpu=neoverse-512tvb` is special in that it does not refer to a specific core, but instead refers to all Neoverse cores that (a) implement SVE and (b) have a total vector bandwidth of 512 bits a cycle. Unless overridden by `-march`, `-mcpu=neoverse-512tvb` generates code that can run on a Neoverse V1 core, since Neoverse V1 is the first Neoverse core with these properties. Unless overridden by `-mtune`, `-mcpu=neoverse-512tvb` tunes code in the same way as for `-mtune=neoverse-512tvb`.

`-moverride=string`

Override tuning decisions made by the back-end in response to a `-mtune=` switch. The syntax, semantics, and accepted values for *string* in this option are not guaranteed to be consistent across releases.

This option is only intended to be useful when developing GCC.

`-mpc-relative-literal-loads`

`-mno-pc-relative-literal-loads`

Enable or disable PC-relative literal loads. With this option literal pools are accessed using a single instruction and emitted after each function. This limits the maximum size of functions to 1MB. This is enabled by default for `-mmodel=tiny`.

The `-mpc-relative-literal-loads` is deprecated.

`-msign-return-address=scope`

Select the function scope on which return address signing will be applied. Permissible values are ‘**none**’, which disables return address signing, ‘**non-leaf**’, which enables pointer signing for functions which are not leaf functions, and

‘all’, which enables pointer signing for all functions. The default value is ‘none’. This option has been deprecated by `-mbranch-protection`.

`-mbranch-protection=features`

Select the branch protection features to use. *features* can have one of the following forms:

‘none’ is the default and turns off all types of branch protection.

‘standard’ turns on all types of branch protection features. If a feature has additional tuning options, then ‘standard’ sets it to its standard level.

‘pac-ret’ turns on return address signing to its standard level: signing functions that save the return address to memory (non-leaf functions practically always do this) using the A-key.

‘pac-ret+leaf’ extends the ‘pac-ret’ signing to include leaf functions.

‘pac-ret+b-key’ or ‘pac-ret+leaf+b-key’ can be used to sign the functions with the B-key instead of the A-key.

‘bti’ turns on branch target identification mechanism.

‘gcs’ turns on guarded control stack compatible code generation.

`-mharden-sls=opts`

Enable compiler hardening against straight line speculation (SLS). *opts* is a comma-separated list of the following options:

‘retbr’

‘blr’

In addition, ‘-mharden-sls=all’ enables all SLS hardening while ‘-mharden-sls=none’ disables all SLS hardening.

`-mearly-ra=scope`

Determine when to enable an early register allocation pass. This pass runs before instruction scheduling and tries to find a spill-free allocation of floating-point and vector code. It also tries to make use of strided multi-register instructions, such as SME2’s strided LD1 and ST1.

The possible values of *scope* are: *all*, which runs the pass on all functions; *strided*, which runs the pass on functions that have access to strided multi-register instructions; and *none*, which disables the pass.

`-mearly-ra=all` is the default for `-O2` and above, and for `-Os`. `-mearly-ra=none` is the default otherwise.

`-mearly-ldp-fusion`

`-mno-early-ldp-fusion`

Enable the copy of the AArch64 load/store pair fusion pass that runs before register allocation. Enabled by default at ‘-O’ and above.

`-mlate-ldp-fusion`

`-mno-late-ldp-fusion`

Enable the copy of the AArch64 load/store pair fusion pass that runs after register allocation. Enabled by default at ‘-O’ and above.

-mnarrow-gp-writes

Enable conversion of 64-bit general purpose register writes to equivalent 32-bit operations when the upper 32 bits are known to be zero. This pass can be controlled with `-mnarrow-gp-writes` and is active at `-O2` and above, but not enabled by default, except for `-mcpu=olympus`.

-msve-vector-bits=bits

Specify the number of bits in an SVE vector register. This option only has an effect when SVE is enabled.

GCC supports two forms of SVE code generation: “vector-length agnostic” output that works with any size of vector register and “vector-length specific” output that allows GCC to make assumptions about the vector length when it is useful for optimization reasons. The possible values of ‘bits’ are: ‘scalable’, ‘128’, ‘256’, ‘512’, ‘1024’ and ‘2048’. Specifying ‘scalable’ selects vector-length agnostic output. At present ‘-msve-vector-bits=128’ also generates vector-length agnostic output for big-endian targets. All other values generate vector-length specific code. The behavior of these values may change in future releases and no value except ‘scalable’ should be relied on for producing code that is portable across different hardware SVE vector lengths.

The default is ‘-msve-vector-bits=scalable’, which produces vector-length agnostic code.

3.20.1.1 -march and -mcpu Feature Modifiers

Feature modifiers used with `-march` and `-mcpu` can be any of the following and their inverses `nofeature`:

‘crc’	Enable CRC extension. This is on by default for <code>-march=armv8.1-a</code> .
‘crypto’	Enable Crypto extension. This also enables Advanced SIMD and floating-point instructions.
‘fp’	Enable floating-point instructions. This is on by default for all possible values for options <code>-march</code> and <code>-mcpu</code> .
‘simd’	Enable Advanced SIMD instructions. This also enables floating-point instructions. This is on by default for all possible values for options <code>-march</code> and <code>-mcpu</code> .
‘sve’	Enable Scalable Vector Extension instructions. This also enables Advanced SIMD and floating-point instructions.
‘lse’	Enable Large System Extension instructions. This is on by default for <code>-march=armv8.1-a</code> .
‘rdma’	Enable Round Double Multiply Accumulate instructions. This is on by default for <code>-march=armv8.1-a</code> .
‘fp16’	Enable FP16 extension. This also enables floating-point instructions.
‘fp16fml’	Enable FP16 fmla extension. This also enables FP16 extensions and floating-point instructions. This option is enabled by default for <code>-march=armv8.4-a</code> . Use of this option with architectures prior to Armv8.2-A is not supported.

<code>'rcpc'</code>	Enable the RCpc extension. This enables the use of the LDAPR instructions for load-acquire atomic semantics, and passes it on to the assembler, enabling inline asm statements to use instructions from the RCpc extension.
<code>'dotprod'</code>	Enable the Dot Product extension. This also enables Advanced SIMD instructions.
<code>'aes'</code>	Enable the Armv8-a aes and pmull crypto extension. This also enables Advanced SIMD instructions.
<code>'sha2'</code>	Enable the Armv8-a sha2 crypto extension. This also enables Advanced SIMD instructions.
<code>'sha3'</code>	Enable the sha512 and sha3 crypto extension. This also enables Advanced SIMD instructions. Use of this option with architectures prior to Armv8.2-A is not supported.
<code>'sm4'</code>	Enable the sm3 and sm4 crypto extension. This also enables Advanced SIMD instructions. Use of this option with architectures prior to Armv8.2-A is not supported.
<code>'profile'</code>	Enable the Statistical Profiling extension. This option is only to enable the extension at the assembler level and does not affect code generation.
<code>'rng'</code>	Enable the Armv8.5-a Random Number instructions. This option is only to enable the extension at the assembler level and does not affect code generation.
<code>'memtag'</code>	Enable the Armv8.5-a Memory Tagging Extensions. Use of this option with architectures prior to Armv8.5-A is not supported.
<code>'sb'</code>	Enable the Armv8-a Speculation Barrier instruction. This option is only to enable the extension at the assembler level and does not affect code generation. This option is enabled by default for <code>-march=armv8.5-a</code> .
<code>'ssbs'</code>	Enable the Armv8-a Speculative Store Bypass Safe instruction. This option is only to enable the extension at the assembler level and does not affect code generation. This option is enabled by default for <code>-march=armv8.5-a</code> .
<code>'predres'</code>	Enable the Armv8-a Execution and Data Prediction Restriction instructions. This option is only to enable the extension at the assembler level and does not affect code generation. This option is enabled by default for <code>-march=armv8.5-a</code> .
<code>'sve2'</code>	Enable the Armv8-a Scalable Vector Extension 2. This also enables SVE instructions.
<code>'sve2-bitperm'</code>	Enable SVE2 bitperm instructions. This also enables SVE2 instructions.
<code>'sve2-sm4'</code>	Enable SVE2 sm4 instructions. This also enables SVE2 instructions.
<code>'sve2-aes'</code>	Enable SVE2 aes instructions. This also enables SVE2 instructions.
<code>'sve2-sha3'</code>	Enable SVE2 sha3 instructions. This also enables SVE2 instructions.

<code>'sve2p1'</code>	Enable SVE2.1 instructions. This also enables SVE2 instructions.
<code>'sve2p2'</code>	Enable SVE2.2 instructions. This also enables SVE2 and SVE2.1 instructions.
<code>'tme'</code>	Enable the Transactional Memory Extension.
<code>'i8mm'</code>	Enable 8-bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating-point instructions. This option is enabled by default for <code>-march=armv8.6-a</code> . Use of this option with architectures prior to Armv8.2-A is not supported.
<code>'f32mm'</code>	Enable 32-bit Floating point Matrix Multiply instructions. This also enables SVE instructions. Use of this option with architectures prior to Armv8.2-A is not supported.
<code>'f64mm'</code>	Enable 64-bit Floating point Matrix Multiply instructions. This also enables SVE instructions. Use of this option with architectures prior to Armv8.2-A is not supported.
<code>'bf16'</code>	Enable brain half-precision floating-point instructions. This also enables Advanced SIMD and floating-point instructions. This option is enabled by default for <code>-march=armv8.6-a</code> . Use of this option with architectures prior to Armv8.2-A is not supported.
<code>'ls64'</code>	Enable the 64-byte atomic load and store instructions for accelerators.
<code>'mops'</code>	Enable the instructions to accelerate memory operations like <code>memcpy</code> , <code>memmove</code> , <code>memset</code> . This option is enabled by default for <code>-march=armv8.8-a</code>
<code>'flagm'</code>	Enable the Flag Manipulation instructions Extension.
<code>'flagm2'</code>	Enable the FlagM2 flag conversion instructions.
<code>'pauth'</code>	Enable the Pointer Authentication Extension.
<code>'cssc'</code>	Enable the Common Short Sequence Compression instructions.
<code>'cmpbr'</code>	Enable the shorter compare and branch instructions, <code>cbb</code> , <code>cbh</code> and <code>cb</code> .
<code>'sme'</code>	Enable the Scalable Matrix Extension.
<code>'sme-i16i64'</code>	Enable the FEAT_SME_I16I64 extension to SME. This also enables SME instructions.
<code>'sme-f64f64'</code>	Enable the FEAT_SME_F64F64 extension to SME. This also enables SME instructions.
<code>'sme2'</code>	Enable the Scalable Matrix Extension 2. This also enables SME instructions.
<code>'sme-f8f16'</code>	Enable the FEAT_SME_F8F16 extension to SME. This also enables SME2 and FP8 instructions.
<code>'sme-f8f32'</code>	Enable the FEAT_SME_F8F32 extension to SME. This also enables SME2 and FP8 instructions.

<code>'sme-b16b16'</code>	Enable the FEAT_SME_B16B16 extension to SME. This also enables SME2 and SVE_B16B16 instructions.
<code>'sme-f16f16'</code>	Enable the FEAT_SME_F16F16 extension to SME. This also enables SME2 instructions.
<code>'sme2p1'</code>	Enable the Scalable Matrix Extension version 2.1. This also enables SME2 instructions.
<code>'sme2p2'</code>	Enable the Scalable Matrix Extension version 2.2. This also enables SME2 and SME2.1 instructions.
<code>'fcma'</code>	Enable the complex number SIMD extensions.
<code>'jscvt'</code>	Enable the <code>fjcvttzs</code> JavaScript conversion instruction.
<code>'frintts'</code>	Enable floating-point round to integral value instructions.
<code>'wfxt'</code>	Enable <code>wfet</code> and <code>wfit</code> instructions.
<code>'xs'</code>	Enable the XS memory attribute extension.
<code>'lse128'</code>	Enable the LSE128 128-bit atomic instructions extension. This also enables LSE instructions.
<code>'d128'</code>	Enable support for 128-bit system register read/write instructions. This also enables the LSE128 extension.
<code>'gcs'</code>	Enable support for Armv9.4-a Guarded Control Stack extension.
<code>'the'</code>	Enable support for Armv8.9-a/9.4-a translation hardening extension.
<code>'rcpc2'</code>	Enable the RCpc2 extension.
<code>'rcpc3'</code>	Enable the RCpc3 (Release Consistency) extension.
<code>'fp8'</code>	Enable the fp8 (8-bit floating point) extension.
<code>'fp8fma'</code>	Enable the fp8 (8-bit floating point) multiply accumulate extension.
<code>'ssve-fp8fma'</code>	Enable the fp8 (8-bit floating point) multiply accumulate extension in streaming mode.
<code>'fp8dot4'</code>	Enable the fp8 (8-bit floating point) to single-precision 4-way dot product extension.
<code>'ssve-fp8dot4'</code>	Enable the fp8 (8-bit floating point) to single-precision 4-way dot product extension in streaming mode.
<code>'fp8dot2'</code>	Enable the fp8 (8-bit floating point) to half-precision 2-way dot product extension.
<code>'ssve-fp8dot2'</code>	Enable the fp8 (8-bit floating point) to half-precision 2-way dot product extension in streaming mode.

<code>'faminmax'</code>	Enable the Floating Point Absolute Maximum/Minimum extension.
<code>'lut'</code>	Enable the Lookup Table extension.
<code>'sme-lutv2'</code>	Enable the SME Lookup Table v2 (LUTv2) extension.
<code>'cpa'</code>	Enable the Checked Pointer Arithmetic instructions.
<code>'sve-b16b16'</code>	Enable the SVE non-widening brain floating-point (bf16) extension. This only has an effect when sve2 or sme2 are also enabled.
<code>'sve-bfscale'</code>	Enable the SVE_BFSCALE extension.
<code>'poe2'</code>	Enable the Permission Overlays Extension 2.
<code>'tev'</code>	Enable the TIndex Exception-like Vector Extension.
<code>'tlbid'</code>	Enable the TLBI Domains Extension.
<code>'gcie'</code>	Enable the GICv5 (Generic Interrupt Controller) CPU Interface Extension.
<code>'mpamv2'</code>	Enable MPAMv2 system registers.
<code>'lscp'</code>	Enable the load acquire and store release pair extension.
<code>'mops-go'</code>	Enable tag only variants of MOPS instructions. This also enables the instructions to accelerate memory operations and Armv8.5-a Memory Tagging Extensions.
<code>'sve2p3'</code>	Enable SVE2.3. This also enables SVE2.2 instructions.
<code>'sme2p3'</code>	Enable SME2.3. This also enables SME2.2 instructions.
<code>'f16f32dot'</code>	Enable Armv9.7-a f16f32dot instructions. This also enables Advanced SIMD, floating-point instructions and Armv8.2-a FP16 instructions.
<code>'sve-b16mm'</code>	Enable the SVE B16MM Extension. This also enables SVE instructions, Advanced SIMD and floating-point instructions.
<code>'mtetc'</code>	Enable Data cache tag block operations. This also enables Armv8.5-a Memory Tagging Extensions.
<code>'f16f32mm'</code>	Enable Armv9.7-a f16f32mm instructions. This also enables Advanced SIMD, floating-point instructions and Armv8.2-a FP16 instructions.
<code>'f16mm'</code>	Enable f16mm instructions. This also enables Advanced SIMD, floating-point instructions and Armv8.2-a FP16 instructions.

Feature **crypto** implies **aes**, **sha2**, and **simd**, which implies **fp**. Conversely, **nofp** implies **nosimd**, which implies **nocrypto**, **noaes** and **nosha2**.

3.20.2 Adapteva Epiphany Options

These ‘-m’ options are defined for Adapteva Epiphany:

-mhalf-reg-file

-mno-half-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

-mno-prefer-short-insn-regs

Preferentially allocate registers that allow short instruction generation. This can result in increased instruction count, so this may either reduce or increase overall code size.

-mbranch-cost=num

Set the cost of branches to roughly *num* “simple” instructions. This cost is only a heuristic and is not guaranteed to produce consistent results across releases.

-mcmove

-mno-cmove

Enable the generation of conditional moves.

-mnops=num

Emit *num* NOPs before every other generated instruction.

-mno-soft-cmpsf

-msoft-cmpsf

For single-precision floating-point comparisons, emit an `fsub` instruction and test the flags. This is faster than a software comparison, but can get incorrect results in the presence of NaNs, or when two different small numbers are compared such that their difference is calculated as zero. The default is **-msoft-cmpsf**, which uses slower, but IEEE-compliant, software comparisons.

-mstack-offset=num

Set the offset between the top of the stack and the stack pointer. E.g., a value of 8 means that the eight bytes in the range `sp+0...sp+7` can be used by leaf functions without stack allocation. Values other than ‘8’ or ‘16’ are untested and unlikely to work. Note also that this option changes the ABI; compiling a program with a different stack offset than the libraries have been compiled with generally does not work. This option can be useful if you want to evaluate if a different stack offset would give you better code, but to actually use a different stack offset to build working programs, it is recommended to configure the toolchain with the appropriate **--with-stack-offset=num** option.

-mno-round-nearest

-mround-nearest

-mno-round-nearest makes 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**. Note that **-mlong-calls** is equivalent to **-mno-short-calls**, and similarly **-mno-long-calls** is equivalent to **-mshort-calls**.

-msmall16**-mno-small16**

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**

-mmay-round-for-trunc**-mno-may-round-for-trunc**

This option allows floating point to integer truncation to be replaced with rounding to save mode switching. It's disabled by default.

`-mfp-iarith`

`-mno-fp-iarith`

This option enables use of the floating-point unit for integer add and subtract. It's disabled by default.

`-msplit-lohi`

`-mno-split-lohi`

`-mpost-inc`

`-mno-post-inc`

`-mpost-modify`

`-mno-post-modify`

Code generation tweaks that control, 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`.

`-mno-vect-double`

Change the preferred SIMD mode to SImode. The default is `mvect-double`, which uses DImode as preferred SIMD mode.

`-max-vect-align=num`

The maximum alignment for SIMD vector mode types. *num* may be 4 or 8. The default is 8. Note that this is an ABI change, even though many library function interfaces are unaffected if they don't use SIMD vector modes in places that affect size and/or alignment of relevant types.

`-msplit-vecmove-early`

`-mno-split-vecmove-early`

Split vector moves into single word moves before reload. In theory this can give better register allocation, but so far the reverse seems to be generally the case.

`-m1reg-reg`

Specify a register to hold the constant `-1`, which makes loading small negative constants and certain bitmasks faster. Allowable values for *reg* are `'r43'` and `'r63'`, which specify use of that register as a fixed register, and `'none'`, which means that no register is used for this purpose. The default is `m1reg-none`.

3.20.3 AMD GCN Options

These options are defined specifically for the AMD GCN port.

`-march=gpu`

`-mtune=gpu`

Set architecture type or tuning for *gpu*. Supported values for *gpu* are

`'gfx900'` Compile for GCN5 Vega 10 devices (gfx900).

`'gfx902'` Compile for GCN5 Vega gfx902 devices. (Experimental)

`'gfx904'` Compile for GCN5 Vega gfx904 devices. (Experimental)

`'gfx906'` Compile for GCN5 Vega 20 devices (gfx906).

`'gfx908'` Compile for CDNA1 Instinct MI100 series devices (gfx908).

`'gfx909'` Compile for GCN5 Vega gfx909 devices. (Experimental)

<code>'gfx90a'</code>	Compile for CDNA2 Instinct MI200 series devices (gfx90a).
<code>'gfx90c'</code>	Compile for GCN5 Vega 7 devices (gfx90c).
<code>'gfx942'</code>	Compile for CDNA3 Instinct MI300 series devices (gfx942). (Experimental)
<code>'gfx950'</code>	Compile for the CDNA3 gfx950 devices. (Experimental)
<code>'gfx9-generic'</code>	Compile generic code for Vega devices, executable on the following subset of GFX9 devices: gfx900, gfx902, gfx904, gfx906, gfx909 and gfx90c.
<code>'gfx9-4-generic'</code>	Compile generic code for CDNA3 devices, executable on the following subset of GFX9 devices: gfx942 and gfx950. (Experimental)
<code>'gfx1030'</code>	Compile for RDNA2 gfx1030 devices (GFX10 series).
<code>'gfx1031'</code>	Compile for RDNA2 gfx1031 devices (GFX10 series). (Experimental)
<code>'gfx1032'</code>	Compile for RDNA2 gfx1032 devices (GFX10 series). (Experimental)
<code>'gfx1033'</code>	Compile for RDNA2 gfx1033 devices (GFX10 series). (Experimental)
<code>'gfx1034'</code>	Compile for RDNA2 gfx1034 devices (GFX10 series). (Experimental)
<code>'gfx1035'</code>	Compile for RDNA2 gfx1035 devices (GFX10 series). (Experimental)
<code>'gfx1036'</code>	Compile for RDNA2 gfx1036 devices (GFX10 series).
<code>'gfx10-3-generic'</code>	Compile generic code for GFX10-3 devices, executable on gfx1030, gfx1031, gfx1032, gfx1033, gfx1034, gfx1035, and gfx1036.
<code>'gfx1100'</code>	Compile for RDNA3 gfx1100 devices (GFX11 series).
<code>'gfx1101'</code>	Compile for RDNA3 gfx1101 devices (GFX11 series). (Experimental)
<code>'gfx1102'</code>	Compile for RDNA3 gfx1102 devices (GFX11 series). (Experimental)
<code>'gfx1103'</code>	Compile for RDNA3 gfx1103 devices (GFX11 series).
<code>'gfx1150'</code>	Compile for RDNA3 gfx1150 devices (GFX11 series). (Experimental)
<code>'gfx1151'</code>	Compile for RDNA3 gfx1151 devices (GFX11 series). (Experimental)
<code>'gfx1152'</code>	Compile for RDNA3 gfx1152 devices (GFX11 series). (Experimental)

‘gfx1153’ Compile for RDNA3 gfx1153 devices (GFX11 series). (Experimental)

‘gfx11-generic’
Compile generic code for GFX11 devices, executable on gfx1100, gfx1101, gfx1102, gfx1103, gfx1150, gfx1151, gfx1152, and gfx1153.

-mgang-private-size=bytes
Set the amount of local data-share (LDS) memory to reserve for gang-private variables. The default is 512.

-msram-ecc=on
-msram-ecc=off
-msram-ecc=any
Compile binaries suitable for devices with the SRAM-ECC feature enabled, disabled, or either mode. This feature can be enabled per-process on some devices. The compiled code must match the device mode. The default is ‘any’, for devices that support it.

-mxnack=on
-mxnack=off
-mxnack=any
Compile binaries suitable for devices with the XNACK feature enabled, disabled, or either mode. Some devices always require XNACK and some allow the user to configure XNACK. The compiled code must match the device mode. The default is ‘-mxnack=any’ on devices that support Unified Shared Memory, and ‘-mxnack=no’ otherwise.

-Wopenacc-dims
-Wno-openacc-dims
Control warnings about invalid OpenACC dimensions.

3.20.4 ARC Options

The following options control the architecture variant for which code is being compiled:

-mbarrel-shifter
Generate instructions supported by barrel shifter. This is the default unless **-mcpu=ARC601** or **‘-mcpu=ARCEM’** is in effect.

-mjli-always
Force to call a function using `jli.s` instruction. This option is valid only for ARCV2 architecture.

-mcpu=cpu
Set architecture type, register usage, and instruction scheduling parameters for *cpu*. There are also shortcut alias options available for backward compatibility and convenience. Supported values for *cpu* are

‘arc600’ Compile for ARC600. Aliases: **-mA6**, **-mARC600**.

‘arc601’ Compile for ARC601. Alias: **-mARC601**.

‘arc700’ Compile for ARC700. Aliases: **-mA7**, **-mARC700**. This is the default when configured with **--with-cpu=arc700**.

<code>'arcem'</code>	Compile for ARC EM.
<code>'archs'</code>	Compile for ARC HS.
<code>'em'</code>	Compile for ARC EM CPU with no hardware extensions.
<code>'em4'</code>	Compile for ARC EM4 CPU.
<code>'em4_dmips'</code>	Compile for ARC EM4 DMIPS CPU.
<code>'em4_fpus'</code>	Compile for ARC EM4 DMIPS CPU with the single-precision floating-point extension.
<code>'em4_fpuda'</code>	Compile for ARC EM4 DMIPS CPU with single-precision floating-point and double assist instructions.
<code>'hs'</code>	Compile for ARC HS CPU with no hardware extensions except the atomic instructions.
<code>'hs34'</code>	Compile for ARC HS34 CPU.
<code>'hs38'</code>	Compile for ARC HS38 CPU.
<code>'hs38_linux'</code>	Compile for ARC HS38 CPU with all hardware extensions on.
<code>'hs4x'</code>	Compile for ARC HS4x CPU.
<code>'hs4xd'</code>	Compile for ARC HS4xD CPU.
<code>'hs4x_rel31'</code>	Compile for ARC HS4x CPU release 3.10a.
<code>'arc600_norm'</code>	Compile for ARC 600 CPU with norm instructions enabled.
<code>'arc600_mul32x16'</code>	Compile for ARC 600 CPU with norm and 32x16-bit multiply instructions enabled.
<code>'arc600_mul64'</code>	Compile for ARC 600 CPU with norm and mul64 -family instructions enabled.
<code>'arc601_norm'</code>	Compile for ARC 601 CPU with norm instructions enabled.
<code>'arc601_mul32x16'</code>	Compile for ARC 601 CPU with norm and 32x16-bit multiply instructions enabled.
<code>'arc601_mul64'</code>	Compile for ARC 601 CPU with norm and mul64 -family instructions enabled.

- `'nps400'` Compile for ARC 700 on NPS400 chip.
 - `'em_mini'` Compile for ARC EM minimalist configuration featuring reduced register set.
 - `-mdpfp`
 - `-mdpfp-compact`
Generate double-precision FPX instructions, tuned for the compact implementation.
- `-mdpfp-fast`
Generate double-precision FPX instructions, tuned for the fast implementation.
- `-mno-dpfp-lrsr`
- `-mdpfp-lrsr`
Control whether `lr` and `sr` instructions use FPX extension aux registers. This is enabled by default.
- `-mea`
Generate extended arithmetic instructions. Currently only `divaw`, `adds`, `subs`, and `sat16` are supported. Only valid for `-mcpu=ARC700`.
- `-mmul32x16`
Generate 32x16-bit multiply and multiply-accumulate instructions.
- `-mmul64`
Generate `mul64` and `mulu64` instructions. Only valid for `-mcpu=ARC600`.
- `-mnorm`
Generate `norm` instructions. This is the default if `-mcpu=ARC700` is in effect.
- `-mspfp`
- `-mspfp-compact`
Generate single-precision FPX instructions, tuned for the compact implementation.
- `-mspfp-fast`
Generate single-precision FPX instructions, tuned for the fast implementation.
- `-msimd`
Enable generation of ARC SIMD instructions via target-specific builtins. Only valid for `-mcpu=ARC700`.
- `-msoft-float`
This option ignored; it is provided for compatibility purposes only. Software floating-point code is emitted by default, and this default can be overridden by FPX options; `-mspfp`, `-mspfp-compact`, or `-mspfp-fast` for single precision, and `-mdpfp`, `-mdpfp-compact`, or `-mdpfp-fast` for double precision.
- `-mswap`
Generate `swap` instructions.
- `-matomic`
This enables use of the locked load/store conditional extension to implement atomic memory built-in functions. Not available for ARC 6xx or ARC EM cores.
- `-mdiv-rem`
Enable `div` and `rem` instructions for ARCV2 cores.
- `-mcode-density`
- `-mno-code-density`
Enable code density instructions for ARC EM. This option is on by default for ARC HS.

- `-mll64` Enable double load/store operations for ARC HS cores.
- `-mtp-regno=regno`
Specify thread pointer register number.
- `-mbitops` Enable use of NPS400 bit operations.
- `-mcmem` Enable use of NPS400 xld/xst extension.
- `-mmpy-option=multo`
Compile ARCV2 code with a multiplier design option. You can specify the option using either a string or numeric value for *multo*. 'w1h1' is the default value. The recognized values are:
 - '0'
 - 'none' No multiplier available.
 - '1'
 - 'w' 16x16 multiplier, fully pipelined. The following instructions are enabled: `mpyw` and `mpyuw`.
 - '2'
 - 'w1h1' 32x32 multiplier, fully pipelined (1 stage). The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.
 - '3'
 - 'w1h2' 32x32 multiplier, fully pipelined (2 stages). The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.
 - '4'
 - 'w1h3' Two 16x16 multipliers, blocking, sequential. The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.
 - '5'
 - 'w1h4' One 16x16 multiplier, blocking, sequential. The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.
 - '6'
 - 'w1h5' One 32x4 multiplier, blocking, sequential. The following instructions are additionally enabled: `mpy`, `mpyu`, `mpym`, `mpymu`, and `mpy_s`.
 - '7'
 - 'plus_dmpy' ARC HS SIMD support.
 - '8'
 - 'plus_macd' ARC HS SIMD support.
 - '9'
 - 'plus_qmacw' ARC HS SIMD support.

This option is only available for ARCV2 cores.

-mfpu=*fpu*

Enables support for specific floating-point hardware extensions for ARCV2 cores. Supported values for *fpu* are:

'fpus' Enables support for single-precision floating-point hardware extensions.

'fpud' Enables support for double-precision floating-point hardware extensions. The single-precision floating-point extension is also enabled. Not available for ARC EM.

'fpuda' Enables support for double-precision floating-point hardware extensions using double-precision assist instructions. The single-precision floating-point extension is also enabled. This option is only available for ARC EM.

'fpuda_div' Enables support for double-precision floating-point hardware extensions using double-precision assist instructions. The single-precision floating-point, square-root, and divide extensions are also enabled. This option is only available for ARC EM.

'fpuda_fma' Enables support for double-precision floating-point hardware extensions using double-precision assist instructions. The single-precision floating-point and fused multiply and add hardware extensions are also enabled. This option is only available for ARC EM.

'fpuda_all' Enables support for double-precision floating-point hardware extensions using double-precision assist instructions. All single-precision floating-point hardware extensions are also enabled. This option is only available for ARC EM.

'fpus_div' Enables support for single-precision floating-point, square-root and divide hardware extensions.

'fpud_div' Enables support for double-precision floating-point, square-root and divide hardware extensions. This option includes option **'fpus_div'**. Not available for ARC EM.

'fpus_fma' Enables support for single-precision floating-point and fused multiply and add hardware extensions.

'fpud_fma' Enables support for double-precision floating-point and fused multiply and add hardware extensions. This option includes option **'fpus_fma'**. Not available for ARC EM.

‘fpus_all’
Enables support for all single-precision floating-point hardware extensions.

‘fpud_all’
Enables support for all single- and double-precision floating-point hardware extensions. Not available for ARC EM.

-mirq-ctrl-saved=register-range, blink, lp_count
Specifies general-purposes registers that the processor automatically saves/restores on interrupt entry and exit. *register-range* is specified as two registers separated by a dash. The register range always starts with **r0**, the upper limit is **fp** register. *blink* and *lp_count* are optional. This option is only valid for ARC EM and ARC HS cores.

-mrgf-banked-regs=number
Specifies the number of registers replicated in second register bank on entry to fast interrupt. Fast interrupts are interrupts with the highest priority level P0. These interrupts save only PC and STATUS32 registers to avoid memory transactions during interrupt entry and exit sequences. Use this option when you are using fast interrupts in an ARC V2 family processor. Permitted values are 4, 8, 16, and 32.

-mlpc-width=width
Specify the width of the **lp_count** register. Valid values for *width* are 8, 16, 20, 24, 28 and 32 bits. The default width is fixed to 32 bits. If the width is less than 32, the compiler does not attempt to transform loops in your program to use the zero-delay loop mechanism unless it is known that the **lp_count** register can hold the required loop-counter value. Depending on the width specified, the compiler and run-time library might continue to use the loop mechanism for various needs. This option defines macro **__ARC_LPC_WIDTH__** with the value of *width*.

-mrf16 This option instructs the compiler to generate code for a 16-entry register file. This option defines the **__ARC_RF16__** preprocessor macro.

-mbranch-index
Enable use of **bi** or **bih** instructions to implement jump tables.

The following options are passed through to the assembler, and also define preprocessor macro symbols.

-mlock Passed down to the assembler to enable the locked load/store conditional extension. Also sets the preprocessor symbol **__Xlock**.

-mswape Passed down to the assembler to enable the swap byte ordering extension instruction. Also sets the preprocessor symbol **__Xswape**.

-mxy Passed down to the assembler to enable the XY memory extension. Also sets the preprocessor symbol **__Xxy**.

The following options control how the assembly code is annotated:

-misize Annotate assembler instructions with estimated addresses.

The following options are passed through to the linker:

`-marclinux`

`-mno-arclinux`

Passed through to the linker, to specify use of the `arclinux` emulation. This option is enabled by default in tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets when profiling is not requested.

`-marclinux_prof`

`-mno-arclinux_prof`

Passed through to the linker, to specify use of the `arclinux_prof` emulation. This option is enabled by default in tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets when profiling is requested.

The following options control the semantics of generated code:

`-mlong-calls`

Generate calls as register indirect calls, thus providing access to the full 32-bit address range.

`-mmmedium-calls`

`-mno-medium-calls`

Don't use less than 25-bit addressing range for calls, which is the offset available for an unconditional branch-and-link instruction. Conditional execution of function calls is suppressed, to allow use of the 25-bit range, rather than the 21-bit range with conditional branch-and-link. This is the default for tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets.

`-G num`

Put definitions of externally-visible data in a small data section if that data is no bigger than *num* bytes. The default value of *num* is 4 for any ARC configuration, or 8 when we have double load/store operations.

`-mno-sdata`

Do not generate sdata references. This is the default for tool chains built for `arc-linux-uclibc` and `arceb-linux-uclibc` targets.

`-mvolatile-cache`

`-mno-volatile-cache`

Control how volatile references are accessed. The default is `-mvolatile-cache`, which uses ordinary cached memory accesses for volatile references. Use `-mno-volatile-cache` to enable cache bypass for volatile references.

The following options fine tune code generation:

`-mauto-modify-reg`

Enable the use of pre/post modify with register displacement.

`-mno-brcc`

`-mbrcc`

This option controls a target-specific pass in `arc_reorg` to generate compare-and-branch (`brcc`) instructions, which is enabled by default. It has no effect on generation of these instructions driven by the combiner pass.

`-mcase-vector-pcrel`

Use PC-relative switch case tables to enable case table shortening. This is the default for `-Os`.

-mno-cond-exec

Disable the ARCompact-specific pass to generate conditional execution instructions.

Due to delay slot scheduling and interactions between operand numbers, literal sizes, instruction lengths, and the support for conditional execution, the target-independent pass to generate conditional execution is often lacking, so the ARC port has kept a special pass around that tries to find more conditional execution generation opportunities after register allocation, branch shortening, and delay slot scheduling have been done. This pass generally, but not always, improves performance and code size, at the cost of extra compilation time, which is why there is an option to switch it off. If you have a problem with call instructions exceeding their allowable offset range because they are conditionalized, you should consider using **-mmedium-calls** instead.

-mearly-cbranchsi

Enable pre-reload use of the **cbranchsi** pattern.

-mindexed-loads

Enable the use of indexed loads. This can be problematic because some optimizers then assume that indexed stores exist, which is not the case.

-mlra-priority-none

Don't indicate any priority for target registers.

-mlra-priority-compact

Indicate target register priority for r0..r3 / r12..r15.

-mlra-priority-noncompact

Reduce target register priority for r0..r3 / r12..r15.

-mmillicode

When optimizing for size (using **-Os**), prologues and epilogues that have to save or restore a large number of registers are often shortened by using call to a special function in libgcc; this is referred to as a *millicode* call. As these calls can pose performance issues, and/or cause linking issues when linking in a nonstandard way, this option is provided to turn on or off millicode call generation.

-mcode-density-frame

This option enable the compiler to emit **enter** and **leave** instructions. These instructions are only valid for CPUs with code-density feature.

-msize-level=level

Fine-tune size optimization with regards to instruction lengths and alignment. The recognized values for *level* are:

- '0' No size optimization. This level is deprecated and treated like '1'.
- '1' Short instructions are used opportunistically.
- '2' In addition, alignment of loops and of code after barriers are dropped.

‘3’ In addition, optional data alignment is dropped, and the option `Os` is enabled.

This defaults to ‘3’ when `-Os` is in effect. Otherwise, the behavior when this is not set is equivalent to level ‘1’.

`-mtune=cpu`

Set instruction scheduling parameters for *cpu*, overriding any implied by `-mcpu=`.

Supported values for *cpu* are

‘ARC600’ Tune for ARC600 CPU.

‘ARC601’ Tune for ARC601 CPU.

‘ARC700’ Tune for ARC700 CPU with standard multiplier block.

‘ARC700-xmac’

Tune for ARC700 CPU with XMAC block.

‘ARC725D’ Tune for ARC725D CPU.

‘ARC750D’ Tune for ARC750D CPU.

‘core3’ Tune for ARCV2 core3 type CPU. This option enable usage of `dbnz` instruction.

‘release31a’

Tune for ARC4x release 3.10a.

`-mmultcost=num`

Cost to assume for a multiply instruction, with ‘4’ being equal to a normal instruction.

3.20.5 ARM Options

These ‘-m’ options are defined for the ARM port:

`-mabi=name`

Generate code for the specified ABI. Permissible values are: ‘`apcs-gnu`’, ‘`atpcs`’, ‘`aapcs`’ and ‘`aapcs-linux`’.

`-mapcs-frame`

Generate a stack frame that is compliant with the ARM Procedure Call Standard for all functions, even if this is not strictly necessary for correct execution of the code. Specifying `-fomit-frame-pointer` with this option causes the stack frames not to be generated for leaf functions. The default is `-mno-apcs-frame`. This option is deprecated.

`-mapcs` This is a synonym for `-mapcs-frame` and is deprecated.

`-mthumb-interwork`

Generate code that supports calling between the ARM and Thumb instruction sets. Without this option, on pre-v5 architectures, the two instruction sets cannot be reliably used inside one program. The default is `-mno-thumb-interwork`, since slightly larger code is generated when `-mthumb-interwork` is specified. In AAPCS configurations this option is meaningless.

-msched-prolog

-mno-sched-prolog

Allow or prevent the reordering of instructions in the function prologue, or the merging of those instruction with the instructions in the function's body. With **-mno-sched-prolog**, this means that all functions start with a recognizable set of instructions (or in fact one of a choice from a small set of different function prologues), and this information can be used to locate the start of functions inside an executable piece of code. The default is **-msched-prolog**.

-mfloat-abi=name

Specifies which floating-point ABI to use. Permissible values are: **'soft'**, **'softfp'** and **'hard'**.

Specifying **'soft'** causes GCC to generate output containing library calls for floating-point operations. **'softfp'** allows the generation of code using hardware floating-point instructions, but still uses the soft-float calling conventions. **'hard'** allows generation of floating-point instructions and uses FPU-specific calling conventions.

The default depends on the specific target configuration. Note that the hard-float and soft-float ABIs are not link-compatible; you must compile your entire program with the same ABI, and link with a compatible set of libraries.

-mgeneral-regs-only

Generate code which uses only the general-purpose registers. This will prevent the compiler from using floating-point and Advanced SIMD registers but will not impose any restrictions on the assembler.

-mlittle-endian

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

-mbig-endian

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

-mbe8

-mbe32

When linking a big-endian image select between BE8 and BE32 formats. The option has no effect for little-endian images and is ignored. The default is dependent on the selected target architecture. For ARMv6 and later architectures the default is BE8, for older architectures the default is BE32. BE32 format has been deprecated by ARM.

-march=name[+extension...]

This specifies the name of the target ARM architecture. GCC uses this name to determine what kind of instructions it can emit when generating assembly code. This option can be used in conjunction with or instead of the **-mcpu=** option.

Permissible names are: **'armv4t'**, **'armv5t'**, **'armv5te'**, **'armv6'**, **'armv6j'**, **'armv6k'**, **'armv6kz'**, **'armv6t2'**, **'armv6z'**, **'armv6zk'**, **'armv7'**, **'armv7-a'**, **'armv7ve'**, **'armv8-a'**, **'armv8.1-a'**, **'armv8.2-a'**, **'armv8.3-a'**, **'armv8.4-a'**, **'armv8.5-a'**, **'armv8.6-a'**, **'armv9-a'**, **'armv7-r'**, **'armv8-r'**, **'armv6-m'**,

`'armv6s-m'`, `'armv7-m'`, `'armv7e-m'`, `'armv8-m.base'`, `'armv8-m.main'`, `'armv8.1-m.main'`, `'iwmmxt'` and `'iwmmxt2'`.

Additionally, the following architectures, which lack support for the Thumb execution state, are recognized but support is deprecated: `'armv4'`.

Many of the architectures support extensions. These can be added by appending `'+extension'` to the architecture name. Extension options are processed in order and capabilities accumulate. An extension will also enable any necessary base extensions upon which it depends. For example, the `'+crypto'` extension will always enable the `'+simd'` extension. The exception to the additive construction is for extensions that are prefixed with `'+no...'`: these extensions disable the specified option and any other extensions that may depend on the presence of that extension.

For example, `'-march=armv7-a+simd+nofp+vfpv4'` is equivalent to writing `'-march=armv7-a+vfpv4'` since the `'+simd'` option is entirely disabled by the `'+nofp'` option that follows it.

Most extension names are generically named, but have an effect that is dependent upon the architecture to which it is applied. For example, the `'+simd'` option can be applied to both `'armv7-a'` and `'armv8-a'` architectures, but will enable the original ARMv7-A Advanced SIMD (Neon) extensions for `'armv7-a'` and the ARMv8-A variant for `'armv8-a'`.

The table below lists the supported extensions for each architecture. Architectures not mentioned do not support any extensions.

<code>'armv5te'</code>	
<code>'armv6'</code>	
<code>'armv6j'</code>	
<code>'armv6k'</code>	
<code>'armv6kz'</code>	
<code>'armv6t2'</code>	
<code>'armv6z'</code>	
<code>'armv6zk'</code>	
<code>'+fp'</code>	The VFPv2 floating-point instructions. The extension <code>'+vfpv2'</code> can be used as an alias for this extension.
<code>'+nofp'</code>	Disable the floating-point instructions.
<code>'armv7'</code>	The common subset of the ARMv7-A, ARMv7-R and ARMv7-M architectures.
<code>'+fp'</code>	The VFPv3 floating-point instructions, with 16 double-precision registers. The extension <code>'+vfpv3-d16'</code> can be used as an alias for this extension. Note that floating-point is not supported by the base ARMv7-M architecture, but is compatible with both the ARMv7-A and ARMv7-R architectures.
<code>'+nofp'</code>	Disable the floating-point instructions.

'armv7-a'	
'+mp'	The multiprocessing extension.
'+sec'	The security extension.
'+fp'	The VFPv3 floating-point instructions, with 16 double-precision registers. The extension '+vfpv3-d16' can be used as an alias for this extension.
'+simd'	The Advanced SIMD (Neon) v1 and the VFPv3 floating-point instructions. The extensions '+neon' and '+neon-vfpv3' can be used as aliases for this extension.
'+vfpv3'	The VFPv3 floating-point instructions, with 32 double-precision registers.
'+vfpv3-d16-fp16'	The VFPv3 floating-point instructions, with 16 double-precision registers and the half-precision floating-point conversion operations.
'+vfpv3-fp16'	The VFPv3 floating-point instructions, with 32 double-precision registers and the half-precision floating-point conversion operations.
'+vfpv4-d16'	The VFPv4 floating-point instructions, with 16 double-precision registers.
'+vfpv4'	The VFPv4 floating-point instructions, with 32 double-precision registers.
'+neon-fp16'	The Advanced SIMD (Neon) v1 and the VFPv3 floating-point instructions, with the half-precision floating-point conversion operations.
'+neon-vfpv4'	The Advanced SIMD (Neon) v2 and the VFPv4 floating-point instructions.
'+nosimd'	Disable the Advanced SIMD instructions (does not disable floating point).
'+nofp'	Disable the floating-point and Advanced SIMD instructions.
'armv7ve'	The extended version of the ARMv7-A architecture with support for virtualization.
'+fp'	The VFPv4 floating-point instructions, with 16 double-precision registers. The extension '+vfpv4-d16' can be used as an alias for this extension.

<code>+simd</code>	The Advanced SIMD (Neon) v2 and the VFPv4 floating-point instructions. The extension <code>+neon-vfpv4</code> can be used as an alias for this extension.
<code>+vfpv3-d16</code>	The VFPv3 floating-point instructions, with 16 double-precision registers.
<code>+vfpv3</code>	The VFPv3 floating-point instructions, with 32 double-precision registers.
<code>+vfpv3-d16-fp16</code>	The VFPv3 floating-point instructions, with 16 double-precision registers and the half-precision floating-point conversion operations.
<code>+vfpv3-fp16</code>	The VFPv3 floating-point instructions, with 32 double-precision registers and the half-precision floating-point conversion operations.
<code>+vfpv4-d16</code>	The VFPv4 floating-point instructions, with 16 double-precision registers.
<code>+vfpv4</code>	The VFPv4 floating-point instructions, with 32 double-precision registers.
<code>+neon</code>	The Advanced SIMD (Neon) v1 and the VFPv3 floating-point instructions. The extension <code>+neon-vfpv3</code> can be used as an alias for this extension.
<code>+neon-fp16</code>	The Advanced SIMD (Neon) v1 and the VFPv3 floating-point instructions, with the half-precision floating-point conversion operations.
<code>+nosimd</code>	Disable the Advanced SIMD instructions (does not disable floating point).
<code>+nofp</code>	Disable the floating-point and Advanced SIMD instructions.
<code>armv8-a</code>	
<code>+crc</code>	The Cyclic Redundancy Check (CRC) instructions.
<code>+simd</code>	The ARMv8-A Advanced SIMD and floating-point instructions.
<code>+crypto</code>	The cryptographic instructions.
<code>+nocrypto</code>	Disable the cryptographic instructions.

	<code>'+nofp'</code>	Disable the floating-point, Advanced SIMD and cryptographic instructions.
	<code>'+sb'</code>	Speculation Barrier Instruction.
	<code>'+predres'</code>	Execution and Data Prediction Restriction Instructions.
<code>'armv8.1-a'</code>		
	<code>'+simd'</code>	The ARMv8.1-A Advanced SIMD and floating-point instructions.
	<code>'+crypto'</code>	The cryptographic instructions. This also enables the Advanced SIMD and floating-point instructions.
	<code>'+nocrypto'</code>	Disable the cryptographic instructions.
	<code>'+nofp'</code>	Disable the floating-point, Advanced SIMD and cryptographic instructions.
	<code>'+sb'</code>	Speculation Barrier Instruction.
	<code>'+predres'</code>	Execution and Data Prediction Restriction Instructions.
<code>'armv8.2-a'</code>		
<code>'armv8.3-a'</code>		
	<code>'+fp16'</code>	The half-precision floating-point data processing instructions. This also enables the Advanced SIMD and floating-point instructions.
	<code>'+fp16fml'</code>	The half-precision floating-point fmla extension. This also enables the half-precision floating-point extension and Advanced SIMD and floating-point instructions.
	<code>'+simd'</code>	The ARMv8.1-A Advanced SIMD and floating-point instructions.
	<code>'+crypto'</code>	The cryptographic instructions. This also enables the Advanced SIMD and floating-point instructions.
	<code>'+dotprod'</code>	Enable the Dot Product extension. This also enables Advanced SIMD instructions.
	<code>'+nocrypto'</code>	Disable the cryptographic extension.
	<code>'+nofp'</code>	Disable the floating-point, Advanced SIMD and cryptographic instructions.
	<code>'+sb'</code>	Speculation Barrier Instruction.

‘+predres’	Execution and Data Prediction Restriction Instructions.
‘+i8mm’	8-bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating-point instructions.
‘+bf16’	Brain half-precision floating-point instructions. This also enables Advanced SIMD and floating-point instructions.
‘armv8.4-a’	
‘+fp16’	The half-precision floating-point data processing instructions. This also enables the Advanced SIMD and floating-point instructions as well as the Dot Product extension and the half-precision floating-point fmla extension.
‘+simd’	The ARMv8.3-A Advanced SIMD and floating-point instructions as well as the Dot Product extension.
‘+crypto’	The cryptographic instructions. This also enables the Advanced SIMD and floating-point instructions as well as the Dot Product extension.
‘+nocrypto’	Disable the cryptographic extension.
‘+nofp’	Disable the floating-point, Advanced SIMD and cryptographic instructions.
‘+sb’	Speculation Barrier Instruction.
‘+predres’	Execution and Data Prediction Restriction Instructions.
‘+i8mm’	8-bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating-point instructions.
‘+bf16’	Brain half-precision floating-point instructions. This also enables Advanced SIMD and floating-point instructions.
‘armv8.5-a’	
‘+fp16’	The half-precision floating-point data processing instructions. This also enables the Advanced SIMD and floating-point instructions as well as the Dot Product extension and the half-precision floating-point fmla extension.

‘+simd’	The ARMv8.3-A Advanced SIMD and floating-point instructions as well as the Dot Product extension.
‘+crypto’	The cryptographic instructions. This also enables the Advanced SIMD and floating-point instructions as well as the Dot Product extension.
‘+nocrypto’	Disable the cryptographic extension.
‘+nofp’	Disable the floating-point, Advanced SIMD and cryptographic instructions.
‘+i8mm’	8-bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating-point instructions.
‘+bf16’	Brain half-precision floating-point instructions. This also enables Advanced SIMD and floating-point instructions.
‘armv8.6-a’	
‘+fp16’	The half-precision floating-point data processing instructions. This also enables the Advanced SIMD and floating-point instructions as well as the Dot Product extension and the half-precision floating-point fmla extension.
‘+simd’	The ARMv8.3-A Advanced SIMD and floating-point instructions as well as the Dot Product extension.
‘+crypto’	The cryptographic instructions. This also enables the Advanced SIMD and floating-point instructions as well as the Dot Product extension.
‘+nocrypto’	Disable the cryptographic extension.
‘+nofp’	Disable the floating-point, Advanced SIMD and cryptographic instructions.
‘+i8mm’	8-bit Integer Matrix Multiply instructions. This also enables Advanced SIMD and floating-point instructions.
‘+bf16’	Brain half-precision floating-point instructions. This also enables Advanced SIMD and floating-point instructions.
‘armv7-r’	
‘+fp.sp’	The single-precision VFPv3 floating-point instructions. The extension ‘+vfpv3xd’ can be used as an alias for this extension.

<code>‘+fp’</code>	The VFPv3 floating-point instructions with 16 double-precision registers. The extension <code>+vfpv3-d16</code> can be used as an alias for this extension.
<code>‘+vfpv3xd-d16-fp16’</code>	The single-precision VFPv3 floating-point instructions with 16 double-precision registers and the half-precision floating-point conversion operations.
<code>‘+vfpv3-d16-fp16’</code>	The VFPv3 floating-point instructions with 16 double-precision registers and the half-precision floating-point conversion operations.
<code>‘+nofp’</code>	Disable the floating-point extension.
<code>‘+idiv’</code>	The ARM-state integer division instructions.
<code>‘+noidiv’</code>	Disable the ARM-state integer division extension.
<code>‘armv7e-m’</code>	
<code>‘+fp’</code>	The single-precision VFPv4 floating-point instructions.
<code>‘+fpv5’</code>	The single-precision FPv5 floating-point instructions.
<code>‘+fp.dp’</code>	The single- and double-precision FPv5 floating-point instructions.
<code>‘+nofp’</code>	Disable the floating-point extensions.
<code>‘armv8.1-m.main’</code>	
<code>‘+dsp’</code>	The DSP instructions.
<code>‘+mve’</code>	The M-Profile Vector Extension (MVE) integer instructions.
<code>‘+mve.fp’</code>	The M-Profile Vector Extension (MVE) integer and single precision floating-point instructions.
<code>‘+fp’</code>	The single-precision floating-point instructions.
<code>‘+fp.dp’</code>	The single- and double-precision floating-point instructions.
<code>‘+nofp’</code>	Disable the floating-point extension.
<code>‘+cdec0, +cdec1, . . . , +cdec7’</code>	Enable the Custom Datapath Extension (CDE) on selected coprocessors according to the numbers given in the options in the range 0 to 7.
<code>‘+pacbti’</code>	Enable the Pointer Authentication and Branch Target Identification Extension.
<code>‘armv8-m.main’</code>	
<code>‘+dsp’</code>	The DSP instructions.

<code>'+nodsp'</code>	Disable the DSP extension.
<code>'+fp'</code>	The single-precision floating-point instructions.
<code>'+fp.dp'</code>	The single- and double-precision floating-point instructions.
<code>'+nofp'</code>	Disable the floating-point extension.
<code>'+cdec0, +cdec1, ... , +cdec7'</code>	Enable the Custom Datapath Extension (CDE) on selected coprocessors according to the numbers given in the options in the range 0 to 7.
<code>'armv8-r'</code>	
<code>'+crc'</code>	The Cyclic Redundancy Check (CRC) instructions.
<code>'+fp.sp'</code>	The single-precision Fpv5 floating-point instructions.
<code>'+simd'</code>	The ARMv8-A Advanced SIMD and floating-point instructions.
<code>'+crypto'</code>	The cryptographic instructions.
<code>'+nocrypto'</code>	Disable the cryptographic instructions.
<code>'+nofp'</code>	Disable the floating-point, Advanced SIMD and cryptographic instructions.

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

`-march=unset` causes the compiler to ignore any `-march=...` options that appear earlier on the command line and behave as if the option was never passed. This is useful to avoid warnings about conflicting CPU and architecture options when the two produce different architecture specifications.

`-mtune=name`

This option specifies the name of the target ARM processor for which GCC should tune the performance of the code. For some ARM implementations better performance can be obtained by using this option. Permissible names are: `'arm7tdmi'`, `'arm7tdmi-s'`, `'arm710t'`, `'arm720t'`, `'arm740t'`, `'strongarm'`, `'strongarm110'`, `'strongarm1100'`, `'strongarm1110'`, `'arm8'`, `'arm810'`, `'arm9'`, `'arm9e'`, `'arm920'`, `'arm920t'`, `'arm922t'`, `'arm946e-s'`, `'arm966e-s'`, `'arm968e-s'`, `'arm926ej-s'`, `'arm940t'`, `'arm9tdmi'`, `'arm10tdmi'`, `'arm1020t'`, `'arm1026ej-s'`, `'arm10e'`, `'arm1020e'`, `'arm1022e'`, `'arm1136j-s'`, `'arm1136jf-s'`, `'mpcore'`, `'mpcorenovfp'`, `'arm1156t2-s'`, `'arm1156t2f-s'`, `'arm1176jz-s'`, `'arm1176jzf-s'`, `'generic-armv7-a'`, `'cortex-a5'`, `'cortex-a7'`, `'cortex-a8'`, `'cortex-a9'`, `'cortex-a12'`, `'cortex-a15'`, `'cortex-a17'`, `'cortex-a32'`, `'cortex-a35'`, `'cortex-a53'`, `'cortex-a55'`, `'cortex-a57'`, `'cortex-a72'`, `'cortex-a73'`, `'cortex-a75'`, `'cortex-a76'`,

'cortex-a76ae', 'cortex-a77', 'cortex-a78', 'cortex-a78ae', 'cortex-a78c', 'cortex-a710', 'ares', 'cortex-r4', 'cortex-r4f', 'cortex-r5', 'cortex-r7', 'cortex-r8', 'cortex-r52', 'cortex-r52plus', 'cortex-m0', 'cortex-m0plus', 'cortex-m1', 'cortex-m3', 'cortex-m4', 'cortex-m7', 'cortex-m23', 'cortex-m33', 'cortex-m35p', 'cortex-m52', 'cortex-m55', 'cortex-m85', 'cortex-x1', 'cortex-x1c', 'cortex-m1.small-multiply', 'cortex-m0.small-multiply', 'cortex-m0plus.small-multiply', 'exynos-m1', 'marvell-pj4', 'neoverse-n1', 'neoverse-n2', 'neoverse-v1', 'xscale', 'iwmmxt', 'iwmmxt2', 'ep9312', 'fa526', 'fa626', 'fa606te', 'fa626te', 'fmp626', 'fa726te', 'star-mc1', 'star-mc3', 'xgene1'.

Additionally, this option can specify that GCC should tune the performance of the code for a big.LITTLE system. Permissible names are: 'cortex-a15.cortex-a7', 'cortex-a17.cortex-a7', 'cortex-a57.cortex-a53', 'cortex-a72.cortex-a53', 'cortex-a72.cortex-a35', 'cortex-a73.cortex-a53', 'cortex-a75.cortex-a55', 'cortex-a76.cortex-a55'.

-mtune=generic-arch specifies that GCC should tune the performance for a blend of processors within architecture *arch*. The aim is to generate code that run well on the current most popular processors, balancing between optimizations that benefit some CPUs in the range, and avoiding performance pitfalls of other CPUs. The effects of this option may change in future GCC versions as CPU models come and go.

-mtune permits the same extension options as **-mcpu**, but the extension options do not affect the tuning of the generated code.

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

-mcpu=name[+extension...]

This specifies the name of the target ARM processor. GCC uses this name to derive the name of the target ARM architecture (as if specified by **-march**) and the ARM processor type for which to tune for performance (as if specified by **-mtune**). Where this option is used in conjunction with **-march** or **-mtune**, those options take precedence over the appropriate part of this option.

Many of the supported CPUs implement optional architectural extensions. Where this is so the architectural extensions are normally enabled by default. If implementations that lack the extension exist, then the extension syntax can be used to disable those extensions that have been omitted. For floating-point and Advanced SIMD (Neon) instructions, the settings of the options **-mfloat-abi** and **-mfpu** must also be considered: floating-point and Advanced SIMD instructions will only be used if **-mfloat-abi** is not set to 'soft'; and any setting of **-mfpu** other than 'auto' will override the available floating-point and SIMD extension instructions.

For example, 'cortex-a9' can be found in three major configurations: integer only, with just a floating-point unit or with floating-point and Advanced SIMD. The default is to enable all the instructions, but the extensions '+nosimd' and

`+nofp` can be used to disable just the SIMD or both the SIMD and floating-point instructions respectively.

Permissible names for this option are the same as those for `-mtune`.

The following extension options are common to the listed CPUs:

- `+nodsp` Disable the DSP instructions on `'cortex-m33'`, `'cortex-m35p'`, `'cortex-m52'`, `'cortex-m55'`, `'cortex-m85'` and `'star-mc3'`. Also disable the M-Profile Vector Extension (MVE) integer and single precision floating-point instructions on `'cortex-m52'`, `'cortex-m55'`, `'cortex-m85'` and `'star-mc3'`.
- `+nopacbti` Disable the Pointer Authentication and Branch Target Identification Extension on `'cortex-m52'`, `'cortex-m85'` and `'star-mc3'`.
- `+nomve` Disable the M-Profile Vector Extension (MVE) integer and single precision floating-point instructions on `'cortex-m52'`, `'cortex-m55'`, `'cortex-m85'` and `'star-mc3'`.
- `+nomve.fp` Disable the M-Profile Vector Extension (MVE) single precision floating-point instructions on `'cortex-m52'`, `'cortex-m55'`, `'cortex-m85'` and `'star-mc3'`.
- `+cdecap0, +cdecap1, ... , +cdecap7` Enable the Custom Datapath Extension (CDE) on selected coprocessors according to the numbers given in the options in the range 0 to 7 on `'cortex-m52'`, `'cortex-m55'`, `'cortex-m85'`, `'star-mc1'` and `'star-mc3'`.
- `+nofp` Disables the floating-point instructions on `'arm9e'`, `'arm946e-s'`, `'arm966e-s'`, `'arm968e-s'`, `'arm10e'`, `'arm1020e'`, `'arm1022e'`, `'arm926ej-s'`, `'arm1026ej-s'`, `'cortex-r5'`, `'cortex-r7'`, `'cortex-r8'`, `'cortex-m4'`, `'cortex-m7'`, `'cortex-m33'`, `'cortex-m35p'`, `'cortex-m52'`, `'cortex-m55'`, `'cortex-m85'` and `'star-mc3'`. Disables the floating-point and SIMD instructions on `'generic-armv7-a'`, `'cortex-a5'`, `'cortex-a7'`, `'cortex-a8'`, `'cortex-a9'`, `'cortex-a12'`, `'cortex-a15'`, `'cortex-a17'`, `'cortex-a15.cortex-a7'`, `'cortex-a17.cortex-a7'`, `'cortex-a32'`, `'cortex-a35'`, `'cortex-a53'` and `'cortex-a55'`.
- `+nofp.dp` Disables the double-precision component of the floating-point instructions on `'cortex-r5'`, `'cortex-r7'`, `'cortex-r8'`, `'cortex-r52'`, `'cortex-r52plus'` and `'cortex-m7'`.
- `+nosimd` Disables the SIMD (but not floating-point) instructions on `'generic-armv7-a'`, `'cortex-a5'`, `'cortex-a7'` and `'cortex-a9'`.
- `+crypto` Enables the cryptographic instructions on `'cortex-a32'`, `'cortex-a35'`, `'cortex-a53'`, `'cortex-a55'`, `'cortex-a57'`,

`'cortex-a72'`, `'cortex-a73'`, `'cortex-a75'`, `'exynos-m1'`,
`'xgene1'`, `'cortex-a57.cortex-a53'`, `'cortex-a72.cortex-a53'`,
`'cortex-a73.cortex-a35'`, `'cortex-a73.cortex-a53'` and
`'cortex-a75.cortex-a55'`.

Additionally the `'generic-armv7-a'` pseudo target defaults to VFPv3 with 16 double-precision registers. It supports the following extension options: `'mp'`, `'sec'`, `'vfpv3-d16'`, `'vfpv3'`, `'vfpv3-d16-fp16'`, `'vfpv3-fp16'`, `'vfpv4-d16'`, `'vfpv4'`, `'neon'`, `'neon-vfpv3'`, `'neon-fp16'`, `'neon-vfpv4'`. The meanings are the same as for the extensions to `-march=armv7-a`.

`-mcpu=generic-arch` is also permissible, and is equivalent to `-march=arch -mtune=generic-arch`. See `-mtune` for more information.

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

`-mcpu=unset` causes the compiler to ignore any `-mcpu=...` options that appear earlier on the command line and behave as if the option was never passed. This is useful to avoid warnings about conflicting CPU and architecture options when the two produce different architecture specifications.

`-mfpu=name`

This specifies what floating-point hardware (or hardware emulation) is available on the target. Permissible names are: `'auto'`, `'vfpv2'`, `'vfpv3'`, `'vfpv3-fp16'`, `'vfpv3-d16'`, `'vfpv3-d16-fp16'`, `'vfpv3xd'`, `'vfpv3xd-fp16'`, `'neon-vfpv3'`, `'neon-fp16'`, `'vfpv4'`, `'vfpv4-d16'`, `'fpv4-sp-d16'`, `'neon-vfpv4'`, `'fpv5-d16'`, `'fpv5-sp-d16'`, `'fp-armv8'`, `'neon-fp-armv8'` and `'crypto-neon-fp-armv8'`. Note that `'neon'` is an alias for `'neon-vfpv3'` and `'vfp'` is an alias for `'vfpv2'`.

The setting `'auto'` is the default and is special. It causes the compiler to select the floating-point and Advanced SIMD instructions based on the settings of `-mcpu` and `-march`.

If the selected floating-point hardware includes the NEON extension (e.g. `-mfpu=neon`), note that floating-point operations are not generated by GCC's auto-vectorization pass unless `-funsafe-math-optimizations` is also specified. This is because NEON hardware does not fully implement the IEEE 754 standard for floating-point arithmetic (in particular denormal values are treated as zero), so the use of NEON instructions may lead to a loss of precision.

You can also set the fpu name at function level by using the `target("fpu=")` function attributes (see Section 6.4.2.4 [ARM Attributes], page 655) or pragmas (see Section 6.5.15 [Function Specific Option Pragmas], page 713).

`-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 6.1.5 [Half-Precision], page 578, for more information.

-mstructure-size-boundary=n

The sizes of all structures and unions are rounded up to a multiple of the number of bits set by this option. Permissible values are 8, 32 and 64. The default value varies for different toolchains. For the COFF targeted toolchain the default value is 8. A value of 64 is only allowed if the underlying ABI supports it.

Specifying a larger number can produce faster, more efficient code, but can also increase the size of the program. Different values are potentially incompatible. Code compiled with one value cannot necessarily expect to work with code or libraries compiled with another value, if they exchange information using structures or unions.

This option is deprecated.

-mabort-on-noreturn

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

-mlong-calls**-mno-long-calls**

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

Even if this switch is enabled, not all function calls are turned into long calls. The heuristic is that static functions, functions that have the `short_call` attribute, functions that are inside the scope of a `#pragma no_long_calls` directive, and functions whose definitions have already been compiled within the current compilation unit are not turned into long calls. The exceptions to this rule are that weak function definitions, functions with the `long_call` attribute or the `section` attribute, and functions that are within the scope of a `#pragma long_calls` directive are always turned into long calls.

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

-msingle-pic-base

Treat the register used for PIC addressing as read-only, rather than loading it in the prologue for each function. The runtime system is responsible for initializing this register with an appropriate value before execution begins.

-mpic-register=reg

Specify the register to be used for PIC addressing. For standard PIC base case, the default is any suitable register determined by compiler. For single PIC base case, the default is 'R9' if target is EABI based or stack-checking is enabled, otherwise the default is 'R10'.

-mpic-data-is-text-relative

Assume that the displacement between the text and data segments is fixed at static link time. This permits using PC-relative addressing operations to access data known to be in the data segment. For non-VxWorks RTP targets, this option is enabled by default. When disabled on such targets, it will enable **-msingle-pic-base** by default.

-mpoke-function-name

Write the name of each function into the text section, directly preceding the function prologue. The generated code is similar to this:

```

t0
    .ascii "arm_poke_function_name", 0
    .align
t1
    .word 0xff000000 + (t1 - t0)
arm_poke_function_name
    mov     ip, sp
    stmfd   sp!, {fp, ip, lr, pc}
    sub     fp, ip, #4

```

When performing a stack backtrace, code can inspect the value of `pc` stored at `fp + 0`. If the trace function then looks at location `pc - 12` and the top 8 bits are set, then we know that there is a function name embedded immediately preceding this location and has length `((pc[-3]) & 0xff000000)`.

-mthumb**-marm**

Select between generating code that executes in ARM and Thumb states. The default for most configurations is to generate code that executes in ARM state, but the default can be changed by configuring GCC with the **--with-mode=state** configure option.

You can also override the ARM and Thumb mode for each function by using the **target("thumb")** and **target("arm")** function attributes (see Section 6.4.2.4 [ARM Attributes], page 655) or pragmas (see Section 6.5.15 [Function Specific Option Pragmas], page 713).

-mtpcs-frame

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all non-leaf functions. (A leaf function is one that does not call any other functions.) The default is **-mno-tpcs-frame**.

-mtpcs-leaf-frame

Generate a stack frame that is compliant with the Thumb Procedure Call Standard for all leaf functions. (A leaf function is one that does not call any other functions.) The default is **-mno-apcs-leaf-frame**.

-mcallee-super-interworking

Gives all externally visible functions in the file being compiled an ARM instruction set header which switches to Thumb mode before executing the rest of the function. This allows these functions to be called from non-interworking code. This option is not valid in AAPCS configurations because interworking is enabled by default.

-mcaller-super-interworking

Allows calls via function pointers (including virtual functions) to execute correctly regardless of whether the target code has been compiled for interworking or not. There is a small overhead in the cost of executing a function pointer if this option is enabled. This option is not valid in AAPCS configurations because interworking is enabled by default.

-mtp=name

Specify the access model for the thread local storage pointer. The model ‘**soft**’ generates calls to `__aeabi_read_tp`. Other accepted models are ‘**tpidrurw**’, ‘**tpidruro**’ and ‘**tpidrprw**’ which fetch the thread pointer from the corresponding system register directly (supported from the arm6k architecture and later). These system registers are accessed through the CP15 co-processor interface and the argument ‘**cp15**’ is also accepted as a convenience alias of ‘**tpidruro**’. The argument ‘**auto**’ uses the best available method for the selected processor. The default setting is ‘**auto**’.

-mtls-dialect=diect

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. This option conflicts with **-mslow-flash-data**.

-mfix-cortex-m3-ldrd

Some Cortex-M3 cores can cause data corruption when **ldrd** instructions with overlapping destination and base registers are used. This option avoids generating these instructions. This option is enabled by default when **-mcpu=cortex-m3** is specified.

-mfix-cortex-a57-aes-1742098**-mno-fix-cortex-a57-aes-1742098****-mfix-cortex-a72-aes-1655431****-mno-fix-cortex-a72-aes-1655431**

Enable (disable) mitigation for an erratum on Cortex-A57 and Cortex-A72 that affects the AES cryptographic instructions. This option is enabled by default when either **-mcpu=cortex-a57** or **-mcpu=cortex-a72** is specified.

-munaligned-access**-mno-unaligned-access**

Enables (or disables) reading and writing of 16- and 32- bit values from addresses that are not 16- or 32- bit aligned. By default unaligned access is

disabled for all pre-ARMv6, all ARMv6-M and for ARMv8-M Baseline architectures, and enabled for all other architectures. If unaligned access is not enabled then words in packed data structures are accessed a byte at a time.

The ARM attribute `Tag_CPU_unaligned_access` is set in the generated object file to either true or false, depending upon the setting of this option. If unaligned access is enabled then the preprocessor symbol `__ARM_FEATURE_UNALIGNED` is also defined.

`-mslow-flash-data`

Assume loading data from flash is slower than fetching instruction. Therefore literal load is minimized for better performance. This option is only supported when compiling for ARMv7 M-profile and off by default. It conflicts with `-mword-relocations`.

`-masm-syntax-unified`

Assume inline assembler is using unified asm syntax. The default is currently off which implies divided syntax. This option has no impact on Thumb2. However, this may change in future releases of GCC. Divided syntax should be considered deprecated.

`-mrestrict-it`

Restricts generation of IT blocks to conform to the rules of ARMv8-A. IT blocks can only contain a single 16-bit instruction from a select set of instructions. This option is on by default for ARMv8-A Thumb mode.

`-mpure-code`

Do not allow constant data to be placed in code sections. Additionally, when compiling for ELF object format give all text sections the ELF processor-specific section attribute `SHF_ARM_PURECODE`. This option is only available when generating non-pic code for M-profile targets.

`-mcmse`

Generate secure code as per the "ARMv8-M Security Extensions: Requirements on Development Tools Engineering Specification", which can be found on <https://developer.arm.com/documentation/ecm0359818/latest/>.

`-mfix-cmse-cve-2021-35465`

Mitigate against a potential security issue with the VLLDM instruction in some M-profile devices when using CMSE (CVE-2021-35465). This option is enabled by default when the option `-mcpu=` is used with `cortex-m33`, `cortex-m35p`, `cortex-m55`, `cortex-m85` or `star-mc1`. The option `-mno-fix-cmse-cve-2021-35465` can be used to disable the mitigation.

`-mstack-protector-guard=guard`

`-mstack-protector-guard-offset=offset`

Generate stack protection code using canary at *guard*. Supported locations are 'global' for a global canary or 'tls' for a canary accessible via the TLS register. The option `-mstack-protector-guard-offset=` is for use with `-fstack-protector-guard=tls` and not for use in user-land code.

-mfdpic

-mno-fdpic

Select the FDPIC ABI, which uses 64-bit function descriptors to represent pointers to functions. When the compiler is configured for **arm*-uclinuxfdpiceabi** targets, this option is on by default and implies **-fPIE** if none of the PIC/PIE-related options is provided. On other targets, it only enables the FDPIC-specific code generation features, and the user should explicitly provide the PIC/PIE-related options as needed.

Note that static linking is not supported because it would still involve the dynamic linker when the program self-relocates. If such behavior is acceptable, use **-static** and **-Wl,-dynamic-linker** options.

The opposite **-mno-fdpic** option is useful (and required) to build the Linux kernel using the same (**arm*-uclinuxfdpiceabi**) toolchain as the one used to build the userland programs.

-mbranch-protection=features

Enable branch protection features (armv8.1-m.main only).

features can have one of the following forms:

‘**none**’ generates code without branch protection or return address signing.

‘**standard**’ generates code with all branch protection features enabled at their standard level.

‘**pac-ret**’ generates code with return address signing set to its standard level, which is to sign all functions that save the return address to memory.

‘**pac-ret+leaf**’ extends the ‘**pac-ret**’ signing to include leaf functions even if they do not write the return address to memory.

‘**bti**’ adds landing-pad instructions at the permitted targets of indirect branch instructions.

If support for the ‘**+pacbti**’ architecture extension is not enabled (e.g. via **-march=**), then all branch protection and return address signing operations are constrained to use only the instructions defined in the architectural-NOP space. The generated code remains backwards-compatible with earlier versions of the architecture, but the additional security can be enabled at run time on processors that support the ‘**PACBTI**’ extension.

Branch target enforcement using BTI can only be enabled at runtime if all code in the application has been compiled with at least ‘**-mbranch-protection=bti**’.

Any setting other than ‘**none**’ is supported only on armv8-m.main or later.

The default is to generate code without branch protection or return address signing.

-mvectorize-with-neon-quad

-mvectorize-with-neon-double

Control whether vectorization uses NEON quad-word or double-word registers. The default is **-mvectorize-with-neon-quad**.

3.20.6 AVR Options

These options are defined for AVR implementations:

`-mmcu=mcu`

Specify the AVR instruction set architecture (ISA) or device type. The default for this option is `avr2`.

The following AVR devices and ISAs are supported. *Note:* A complete device support consists of startup code `crtmcu.o`, a device header `avr/io*.h`, a device library `libmcu.a` and a device-specs file `specs-mcu`. Only the latter is provided by the compiler according the supported *mcus* below. The rest is supported by AVR-LibC, or by means of `atpack` files from the hardware manufacturer.

- `avr2` “Classic” devices with up to 8 KiB of program memory.
`mcu = attiny22, attiny26, at90s2313, at90s2323, at90s2333, at90s2343, at90s4414, at90s4433, at90s4434, at90c8534, at90s8515, at90s8535.`
- `avr25` “Classic” devices with up to 8 KiB of program memory and with the MOVW instruction.
`mcu = attiny13, attiny13a, attiny24, attiny24a, attiny25, attiny261, attiny261a, attiny2313, attiny2313a, attiny43u, attiny44, attiny44a, attiny45, attiny48, attiny441, attiny461, attiny461a, attiny4313, attiny84, attiny84a, attiny85, attiny87, attiny88, attiny828, attiny841, attiny861, attiny861a, ata5272, ata6616c, at86rf401.`
- `avr3` “Classic” devices with 16 KiB up to 64 KiB of program memory.
`mcu = at76c711, at43usb355.`
- `avr31` “Classic” devices with 128 KiB of program memory.
`mcu = atmega103, at43usb320.`
- `avr35` “Classic” devices with 16 KiB up to 64 KiB of program memory and with the MOVW instruction.
`mcu = attiny167, attiny1634, atmega8u2, atmega16u2, atmega32u2, ata5505, ata6617c, ata664251, at90usb82, at90usb162.`
- `avr4` “Enhanced” devices with up to 8 KiB of program memory.
`mcu = atmega48, atmega48a, atmega48p, atmega48pa, atmega48pb, atmega8, atmega8a, atmega8hva, atmega88, atmega88a, atmega88p, atmega88pa, atmega88pb, atmega8515, atmega8535, ata5795, ata6285, ata6286, ata6289, ata6612c, at90pwm1, at90pwm2, at90pwm2b, at90pwm3, at90pwm3b, at90pwm81.`
- `avr5` “Enhanced” devices with 16 KiB up to 64 KiB of program memory.
`mcu = atmega16, atmega16a, atmega16hva, atmega16hva2, atmega16hvb, atmega16hvbrevb, atmega16m1, atmega16u4, atmega161, atmega162, atmega163, atmega164a, atmega164p,`

```

atmega164pa,    atmega165,    atmega165a,    atmega165p,
atmega165pa,    atmega168,    atmega168a,    atmega168p,
atmega168pa,    atmega168pb,    atmega169,    atmega169a,
atmega169p, atmega169pa, atmega32, atmega32a, atmega32c1,
atmega32hvb, atmega32hvbrevb, atmega32m1, atmega32u4,
atmega32u6, atmega323, atmega324a, atmega324p, atmega324pa,
atmega324pb,    atmega325,    atmega325a,    atmega325p,
atmega325pa,    atmega328,    atmega328p,    atmega328pb,
atmega329, atmega329a, atmega329p, atmega329pa, atmega3250,
atmega3250a,    atmega3250p,    atmega3250pa,    atmega3290,
atmega3290a,    atmega3290p,    atmega3290pa,    atmega406,
atmega64, atmega64a, atmega64c1, atmega64hve, atmega64hve2,
atmega64m1, atmega64rfr2, atmega640, atmega644, atmega644a,
atmega644p,    atmega644pa,    atmega644rfr2,    atmega645,
atmega645a, atmega645p, atmega649, atmega649a, atmega649p,
atmega6450,    atmega6450a,    atmega6450p,    atmega6490,
atmega6490a, atmega6490p, ata5790, ata5790n, ata5791,
ata6613c, ata6614q, ata5782, ata5831, ata8210, ata8510,
ata5787, ata5835, ata5700m322, ata5702m322, at90pwm161,
at90pwm216, at90pwm316, at90can32, at90can64, at90scr100,
at90usb646, at90usb647, at94k, m3000.

```

avr51 “Enhanced” devices with 128 KiB of program memory.
mcu = atmega128, atmega128a, atmega128rfa1, atmega128rfr2,
atmega1280, atmega1281, atmega1284, atmega1284p,
atmega1284rfr2, at90can128, at90usb1286, at90usb1287.

avr6 “Enhanced” devices with 3-byte PC, i.e. with more than 128 KiB
of program memory.
mcu = atmega256rfr2, atmega2560, atmega2561,
atmega2564rfr2.

avrxmega2 “XMEGA” devices with more than 8 KiB and up to 64 KiB of
program memory.
mcu = atxmega8e5, atxmega16a4, atxmega16a4u, atxmega16c4,
atxmega16d4, atxmega16e5, atxmega32a4, atxmega32a4u,
atxmega32c3, atxmega32c4, atxmega32d3, atxmega32d4,
atxmega32e5, avr64da28, avr64da28s, avr64da32, avr64da32s,
avr64da48, avr64da48s, avr64da64, avr64da64s, avr64db28,
avr64db32, avr64db48, avr64db64, avr64dd14, avr64dd20,
avr64dd28, avr64dd32, avr64du28, avr64du32, avr64ea28,
avr64ea32, avr64ea48, avr64sd28, avr64sd32, avr64sd48.

avrxmega3 “XMEGA” devices with up to 64 KiB of combined program
memory and RAM, and with program memory visible in the RAM
address space.
mcu = attiny202, attiny204, attiny212, attiny214,

attiny402, attiny404, attiny406, attiny412, attiny414,
 attiny416, attiny416auto, attiny417, attiny424, attiny426,
 attiny427, attiny804, attiny806, attiny807, attiny814,
 attiny816, attiny817, attiny824, attiny826, attiny827,
 attiny1604, attiny1606, attiny1607, attiny1614, attiny1616,
 attiny1617, attiny1624, attiny1626, attiny1627, attiny3214,
 attiny3216, attiny3217, attiny3224, attiny3226, attiny3227,
 atmega808, atmega809, atmega1608, atmega1609, atmega3208,
 atmega3209, atmega4808, atmega4809, avr16dd14, avr16dd20,
 avr16dd28, avr16dd32, avr16du14, avr16du20, avr16du28,
 avr16du32, avr16ea28, avr16ea32, avr16ea48, avr16eb14,
 avr16eb20, avr16eb28, avr16eb32, avr16la14, avr16la20,
 avr16la28, avr16la32, avr32da28, avr32da28s, avr32da32,
 avr32da32s, avr32da48, avr32da48s, avr32db28, avr32db32,
 avr32db48, avr32dd14, avr32dd20, avr32dd28, avr32dd32,
 avr32du14, avr32du20, avr32du28, avr32du32, avr32ea28,
 avr32ea32, avr32ea48, avr32eb14, avr32eb20, avr32eb28,
 avr32eb32, avr32la14, avr32la20, avr32la28, avr32la32,
 avr32sd20, avr32sd28, avr32sd32.

avrxmega4

“XMEGA” devices with more than 64 KiB and up to 128 KiB of program memory.

mcu = atxmega64a3, atxmega64a3u, atxmega64a4u,
 atxmega64b1, atxmega64b3, atxmega64c3, atxmega64d3,
 atxmega64d4, avr128da28, avr128da28s, avr128da32,
 avr128da32s, avr128da48, avr128da48s, avr128da64,
 avr128da64s, avr128db28, avr128db32, avr128db48,
 avr128db64.

avrxmega5

“XMEGA” devices with more than 64 KiB and up to 128 KiB of program memory and more than 64 KiB of RAM.

mcu = atxmega64a1, atxmega64a1u.

avrxmega6

“XMEGA” devices with more than 128 KiB of program memory.

mcu = atxmega128a3, atxmega128a3u, atxmega128b1,
 atxmega128b3, atxmega128c3, atxmega128d3, atxmega128d4,
 atxmega192a3, atxmega192a3u, atxmega192c3, atxmega192d3,
 atxmega256a3, atxmega256a3b, atxmega256a3bu,
 atxmega256a3u, atxmega256c3, atxmega256d3, atxmega384c3,
 atxmega384d3.

avrxmega7

“XMEGA” devices with more than 128 KiB of program memory and more than 64 KiB of RAM.

mcu = atxmega128a1, atxmega128a1u, atxmega128a4u.

avrtiny “Reduced Tiny” Tiny core devices with only 16 general purpose registers and 512 B up to 4 KiB of program memory.
`mcu = attiny4, attiny5, attiny9, attiny10, attiny102, attiny104, attiny20, attiny40.`

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

-mabsdata

Assume that all data in static storage can be accessed by LDS / STS instructions. This option has only an effect on reduced Tiny devices like ATtiny40. See also the `absdata` Section 6.4.2.5 [AVR Attributes], page 657.

-masm-len-notes

-mno-asm-len-notes

Recognize resp. ignore `[[len=spec]]` notes that are placed in (comments in) inline assembly code templates. They allow to specify the exact Section 6.11.7 [size of `asm` constructs], page 779. This option is on per default.

-mcv Use a *compact vector table*. Some devices support a CVT with only four entries: 0=Reset, 1=NMI, 2=Prio1 IRQ, 3=Prio0 IRQs. This option will link startup code from `crtmcu-cvt.o` instead of the usual `crtmcu.o`. Apart from providing a compact vector table, the startup code will set bit `CPUINT_CTRLA.CPUINT_CVT` which enables the CVT on the device.

When you do not want the startup code to set `CPUINT_CTRLA.CPUINT_CVT`, then you can satisfy symbol `__init_cvt` so that the respective code is no more pulled in from `libmcu.a`. For example, you can link with `-Wl,--defsym,__init_cvt=0`.

The CVT startup code is available since AVR-LibC v2.3.

-mdouble=bits

-mlong-double=bits

Set the size (in bits) of the `double` or `long double` type, respectively. Possible values for *bits* are 32 and 64. Whether or not a specific value for *bits* is allowed depends on the `--with-double=` and `--with-long-double=` configure options, and the same applies for the default values of the options.

-mgas-isr-prologues

Interrupt service routines (ISRs) may use the `__gcc_isr` pseudo instruction supported by GNU Binutils. If this option is on, the feature can still be disabled for individual ISRs by means of the Section 6.4.2.5 `[no_gcc_isr]`, page 657, function attribute. This feature is activated per default if optimization is on (but not with `-Og`, see Section 3.12 [Optimize Options], page 196), and if GNU Binutils support PR21683.

-mint8 Assume `int` to be 8-bit integer. This affects the sizes of all types: a `char` is 1 byte, an `int` is 1 byte, a `long` is 2 bytes, and `long long` is 4 bytes. Please note that this option does not conform to the C standards, but it results in smaller code size.

-mmain-is-OS_task

Do not save registers in `main`. The effect is the same like attaching attribute `__attribute__((__no_saveregs__))` to `main`. It is activated per default if optimization is on.

-mno-call-main

Don't run `main` by means of

```
XCALL  main
XJMP   exit
```

Instead, put `main` in section `.init9` so that no call is required. By setting this option the user asserts that `main` will not return.

This option can be used for devices with very limited resources in order to save a few bytes of code and stack space. It will work as expected since AVR-LibC v2.3. With older versions, there will be no performance gain.

-mno-interrupts

Generated code is not compatible with hardware interrupts. Code size is smaller.

-mrelax Try to replace `CALL` resp. `JMP` instruction by the shorter `RCALL` resp. `RJMP` instruction if applicable. Setting `-mrelax` just adds the `--mlink-relax` option to the assembler's command line and the `--relax` option to the linker's command line.

Jump relaxing is performed by the linker because jump offsets are not known before code is located. Therefore, the assembler code generated by the compiler is the same, but the instructions in the executable may differ from instructions in the assembler code.

Relaxing must be turned on if linker stubs are needed, see the section on `EIND` and linker stubs below.

-mpmem-wrap-around

Enable program counter wrap-around in linker relaxation.

-mrodata-in-ram**-mno-rodata-in-ram**

Locate the `.rodata` sections for read-only data in RAM resp. in program memory. For most devices, there is no choice and this option acts rather like an assertion.

Since v14 and for the AVR64* and AVR128* devices, `.rodata` is located in flash memory per default, provided the required GNU Binutils support (PR31124) is available. In that case, `-mrodata-in-ram` can be used to return to the old layout with `.rodata` in RAM.

-mtiny-stack

Only change the lower 8 bits of the stack pointer.

-mfractional-convert-truncate

Allow to use truncation instead of rounding towards zero for fractional fixed-point types.

-nodevicelib

Don't link against AVR-LibC's device specific library `libmcu.a`.

Notice that since AVR-LibC v2.3, that library contains code that is essential for the correct functioning of a program. In particular, it contains parts of the startup code like: `__init_sp` to initialize the stack pointer with symbol `__stack`, `__init_cvt` to set up the hardware to use a compact vector table with `-mcv`, `__call_main` to call `main` and `exit`, and `__do_flmmap_init` to set up FLMAP according to symbol `__flmap`.

-nodespecs

Don't add `-specs=device-specs/specs-mcu` to the compiler driver's command line. The user takes responsibility for supplying the sub-processes like compiler proper, assembler and linker with appropriate command-line options. This means that the user has to supply her private device specs file by means of `-specs=path-to-specs-file`. There is no more need for option `-mmcu=mcu`.

This option can also serve as a replacement for the older way of specifying custom device-specs files that needed `-B some-path` to point to a directory which contains a folder named `device-specs` which contains a specs file named `specs-mcu`, where `mcu` was specified by `-mmcu=mcu`.

-Waddr-space-convert

Warn about conversions between address spaces in the case where the resulting address space is not contained in the incoming address space.

-Wasm-len-notes

Warn about unrecognized `[[len=spec]]` Section 6.11.7 [length], page 779, annotations in inline assembly code templates. This Section 3.9 [warning], page 101, is on per default. Notice that in LTO mode, the code must be compiled with option `-ffat-lto-objects` in order for these diagnoses to appear.

-Wmisspelled-isr

Warn if the ISR is misspelled, i.e. without `__vector` prefix. Enabled by default.

3.20.6.1 AVR Optimization Options

The following options are pure optimization options. Options `-mgas-isr-prologues`, `-mmain-is-OS_task`, `-mno-call-main` and `-mrelax` from above are only *almost* optimization options, since there are rare occasions where their different code generation matters.

-maccumulate-args

Accumulate outgoing function arguments and acquire/release the needed stack space for outgoing function arguments once in function prologue/epilogue. Without this option, outgoing arguments are pushed before calling a function and popped afterwards. See also the `-fdefer-pop` Section 3.12 [optimization option], page 196.

Popping the arguments after the function call can be expensive on AVR so that accumulating the stack space might lead to smaller executables because arguments need not be removed from the stack after such a function call.

This option can lead to reduced code size for functions that perform several calls to functions that get their arguments on the stack like calls to printf-like functions.

-mbranch-cost=*cost*

Set the branch costs for conditional branch instructions to *cost*. Reasonable values for *cost* are small, non-negative integers. The default branch cost is 0.

-mcall-prologues

Functions prologues/epilogues are expanded as calls to appropriate subroutines. Code size is smaller.

-mfuse-add

-mno-fuse-add

-mfuse-add=*level*

Optimize indirect memory accesses on reduced Tiny devices. The default uses *level*=1 for optimizations *-Og* and *-O1*, and *level*=2 for higher optimizations. Valid values for *level* are 0, 1 and 2.

-mfuse-move

-mno-fuse-move

-mfuse-move=*level*

Run a post reload optimization pass that tries to fuse move instructions and to split multi-byte instructions into 8-bit operations. The default uses *level*=3 for optimization *-O1*, and *level*=23 for higher optimizations. Valid values for *level* are in the range 0 . . . 23 which is a 3:2:2:2 mixed radix value. Each digit controls some aspect of the optimization.

-mfuse-move2

Run a post combine optimization pass that tries to fuse move instructions.

-mstrict-X

Use address register *X* in a way proposed by the hardware. This means that *X* is only used in indirect, post-increment or pre-decrement addressing.

Without this option, the *X* register may be used in the same way as *Y* or *Z* which then is emulated by additional instructions. For example, loading a value with *X+const* addressing with a small non-negative *const* < 64 to a register *Rn* is performed as

```
    adiw r26, const    ; X += const
    ld   Rn, X         ; Rn = *X
    sbiw r26, const    ; X -= const
```

-msplit-bit-shift

Split multi-byte shifts with a constant offset into a shift with a byte offset and a residual shift with a non-byte offset. This optimization is turned on per default for *-O2* and higher, including *-Os* but excluding *-Oz*. Splitting of shifts with a constant offset that is a multiple of 8 is controlled by **-mfuse-move**.

-msplit-ldst

Split multi-byte loads and stores into several byte loads and stores. This optimization is turned on per default for *-O2* and higher.

-muse-nonzero-bits

Enable optimizations that are only possible when some bits in a register are always zero. This optimization is turned on per default for -O2 and higher.

3.20.6.2 EIND and Devices with More Than 128 Ki Bytes of Flash

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

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

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

- The compiler never sets **EIND**.
- The compiler uses **EIND** implicitly in **EICALL**/**EIJMP** instructions or might read **EIND** directly in order to emulate an indirect call/jump by means of a **RET** instruction.
- The compiler assumes that **EIND** never changes during the startup code or during the application. In particular, **EIND** is not saved/restored in function or interrupt service routine prologue/epilogue.
- For indirect calls to functions and computed goto, the linker generates *stubs*. Stubs are jump pads sometimes also called *trampolines*. Thus, the indirect call/jump jumps to such a stub. The stub contains a direct jump to the desired address.
- Linker relaxation must be turned on so that the linker generates the stubs correctly in all situations. See the compiler option **-mrelax** and the linker option **--relax**. There are corner cases where the linker is supposed to generate stubs but aborts without relaxation and without a helpful error message.
- The default linker script is arranged for code with **EIND** = 0. If code is supposed to work for a setup with **EIND** != 0, a custom linker script has to be used in order to place the sections whose name start with **.trampolines** into the segment where **EIND** points to.
- The startup code from libgcc never sets **EIND**. Notice that startup code is a blend of code from libgcc and AVR-LibC. For the impact of AVR-LibC on **EIND**, see the AVR-LibC user manual.
- 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.

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

Some AVR devices support memories larger than the 64 KiB range that can be accessed with 16-bit pointers. To access memory locations outside this 64 KiB range, the content of a RAMP register is used as high part of the address: The X, Y, Z address register is concatenated with the RAMPX, RAMPY, RAMPZ special function register, respectively, to get a wide address. Similarly, RAMPD is used together with direct addressing.

- The startup code initializes the RAMP special function registers with zero.

- If a [named address space], page 589, 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.

3.20.6.4 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 589, and Section 7.13.8 [AVR Built-in Functions], page 852.

`__AVR_ARCH__`

Build-in macro that resolves to a decimal number that identifies the architecture and depends on the `-mmcu=mcu` option. Possible values are:

2, 25, 3, 31, 35, 4, 5, 51, 6

for `mcu=avr2`, `avr25`, `avr3`, `avr31`, `avr35`, `avr4`, `avr5`, `avr51`, `avr6`, respectively and

100, 102, 103, 104, 105, 106, 107

for `mcu=avrtiny`, `avrxmega2`, `avrxmega3`, `avrxmega4`, `avrxmega5`, `avrxmega6`, `avrxmega7`, respectively. If `mcu` specifies a device, this built-in macro is set accordingly. For example, with `-mmcu=atmega8` the macro is defined to 4.

`__AVR_Device__`

Setting `-mmcu=device` defines this built-in macro which reflects the device's name. For example, `-mmcu=atmega8` defines the built-in macro `__AVR_ATmega8__`, `-mmcu=attiny261a` defines `__AVR_ATtiny261A__`, etc.

The built-in macros' names follow the scheme `__AVR_Device__` where *Device* is the device name as from the AVR user manual. The difference between *Device* in the built-in macro and *device* in `-mmcu=device` is that the latter is always lowercase.

If *device* is not a device but only a core architecture like 'avr51', this macro is not defined.

`__AVR_DEVICE_NAME__`

Setting `-mmcu=device` defines this built-in macro to the device's name. For example, with `-mmcu=atmega8` the macro is defined to `atmega8`.

If *device* is not a device but only a core architecture like 'avr51', this macro is not defined.

`__AVR_CVT__`
 The code is being compiled with option `-mcvt` to use a *compact vector table*.

`__AVR_XMEGA__`
 The device / architecture belongs to the XMEGA family of devices.

`__AVR_HAVE_ADIW__`
 The device has the ADIW and SBIW instructions.

`__AVR_HAVE_ELPM__`
 The device has the ELPM instruction.

`__AVR_HAVE_ELPMX__`
 The device has the ELPM *Rn*, *Z* and ELPM *Rn*, *Z*+ instructions.

`__AVR_HAVE_LPMX__`
 The device has the LPM *Rn*, *Z* and LPM *Rn*, *Z*+ instructions.

`__AVR_HAVE_MOVW__`
 The device has the MOVW instruction to perform 16-bit register-register moves.

`__AVR_HAVE_MUL__`
 The device has a hardware multiplier.

`__AVR_HAVE_JMP_CALL__`
 The device has the JMP and CALL instructions. This is the case for devices with more than 8 KiB of program memory.

`__AVR_HAVE_EIJMP_EICALL__`
`__AVR_3_BYTE_PC__`
 The device has the EIJP and EICALL instructions. This is the case for devices with more than 128 KiB of program memory. This also means that the program counter (PC) is 3 bytes wide.

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

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

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

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

`__NO_INTERRUPTS__`

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

`__AVR_ERRATA_SKIP__`

`__AVR_ERRATA_SKIP_JMP_CALL__`

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

`__AVR_ISA_RMW__`

The device has Read-Modify-Write instructions (`XCH`, `LAC`, `LAS` and `LAT`).

`__AVR_SFR_OFFSET__=offset`

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

`__AVR_SHORT_CALLS__`

The `-mshort-calls` command-line option is set.

`__AVR_PM_BASE_ADDRESS__=addr`

Some devices support reading from flash memory by means of `LD*` instructions. The flash memory is seen in the data address space at an offset of `__AVR_PM_BASE_ADDRESS__`. If this macro is not defined, this feature is not available. If defined, the address space is linear and there is no need to put `.rodata` into RAM. This is handled by the default linker description file, and is currently available for `avrtiny` and `avrxcmega3`. Even more convenient, there is no need to use address spaces like `__flash` or features like attribute `progmem` and `pgm_read*`.

`__AVR_HAVE_FLMAP__`

This macro is defined provided the following conditions are met:

- The device has the `NVMCTRL_CTRLB.FLMAP` bitfield. This applies to the AVR64* and AVR128* devices.
- It's not known at assembler-time which emulation will be used.

This implies the compiler was configured with GNU Binutils that implement PR31124.

`__AVR_RODATA_IN_RAM__`

This macro is undefined when the code is compiled for a core architecture.

When the code is compiled for a device, the macro is defined to 1 when the `.rodata` sections for read-only data is located in RAM; and defined to 0, otherwise.

`__WITH_AVRLIBC__`

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

__HAVE_SIGNAL_N__

The compiler supports the `signal(num)` and `interrupt(num)` Section 6.4.2.5 [function attributes], page 657, with an argument *num* that specifies the number of the interrupt service routine.

__HAVE_DOUBLE_MULTILIB__

Defined if `-mdouble=` acts as a multilib option.

__HAVE_DOUBLE32__**__HAVE_DOUBLE64__**

Defined if the compiler supports 32-bit double resp. 64-bit double. The actual layout is specified by option `-mdouble=`.

__DEFAULT_DOUBLE__

The size in bits of `double` if `-mdouble=` is not set. To test the layout of `double` in a program, use the built-in macro `__SIZEOF_DOUBLE__`.

__HAVE_LONG_DOUBLE32__**__HAVE_LONG_DOUBLE64__****__HAVE_LONG_DOUBLE_MULTILIB__****__DEFAULT_LONG_DOUBLE__**

Same as above, but for `long double` instead of `double`.

__WITH_DOUBLE_COMPARISON__

Reflects the `--with-double-comparison={tristate|bool|libf7}` configure option and is defined to 2 or 3.

__WITH_LIBF7_LIBGCC__**__WITH_LIBF7_MATH__****__WITH_LIBF7_MATH_SYMBOLS__**

Reflects the `--with-libf7={libgcc|math|math-symbols}` configure option.

3.20.6.5 AVR Internal Options

The following options are used internally by the compiler and to communicate between device specs files and the compiler proper. You don't need to set these options by hand, in particular they are not optimization options. Using these options in the wrong way may lead to sub-optimal or wrong code. They are documented for completeness, and in order to get a better understanding of device specs files.

-mn-flash=*num*

Assume that the flash memory has a size of *num* times 64 KiB. This determines which `__flashN` address spaces are available.

-mflmap The device has the FLMAP bit field located in special function register `NVMCTRL_CTRLB`.

-mrmw Assume that the device supports the Read-Modify-Write instructions `XCH`, `LAC`, `LAS` and `LAT`.

-mshort-calls

Assume that `RJMP` and `RCALL` can target the whole program memory. This option is used for multilib generation and selection for the devices from architecture `avrxcmega3`.

- mskip-bug** Generate code without skips (CPSE, SBRS, SBRC, SBIS, SBIC) over 32-bit instructions.
- msp8** Treat the stack pointer register as an 8-bit register, i.e. assume the high byte of the stack pointer is zero. This option is used by the compiler to select and build multilibs for architectures **avr2** and **avr25**. These architectures mix devices with and without SPH.

3.20.7 Blackfin Options

- mcpu=cpu[-sirevision]**

Specifies the name of the target Blackfin processor. Currently, *cpu* can be one of 'bf512', 'bf514', 'bf516', 'bf518', 'bf522', 'bf523', 'bf524', 'bf525', 'bf526', 'bf527', 'bf531', 'bf532', 'bf533', 'bf534', 'bf536', 'bf537', 'bf538', 'bf539', 'bf542', 'bf544', 'bf547', 'bf548', 'bf549', 'bf542m', 'bf544m', 'bf547m', 'bf548m', 'bf549m', 'bf561', 'bf592'.

The optional *sirevision* specifies the silicon revision of the target Blackfin processor. Any workarounds available for the targeted silicon revision are enabled. If *sirevision* is 'none', no workarounds are enabled. If *sirevision* is 'any', all workarounds for the targeted processor are enabled. The `__SILICON_REVISION__` macro is defined to two hexadecimal digits representing the major and minor numbers in the silicon revision. If *sirevision* is 'none', the `__SILICON_REVISION__` is not defined. If *sirevision* is 'any', the `__SILICON_REVISION__` is defined to be 0xffff. If this optional *sirevision* is not used, GCC assumes the latest known silicon revision of the targeted Blackfin processor.

GCC defines a preprocessor macro for the specified *cpu*. For the 'bfin-elf' toolchain, this option causes the hardware BSP provided by libgloss to be linked in if **-msim** is not given.

Without this option, 'bf532' is used as the processor by default.

Note that support for 'bf561' is incomplete. For 'bf561', only the preprocessor macro is defined.
- msim** Specifies that the program will be run on the simulator. This causes the simulator BSP provided by libgloss to be linked in. This option has effect only for 'bfin-elf' toolchain. Certain other options, such as **-mid-shared-library** and **-mfdpic**, imply **-msim**.
- momit-leaf-frame-pointer**

Don't keep the frame pointer in a register for leaf functions. This avoids the instructions to save, set up and restore frame pointers and makes an extra register available in leaf functions.
- mspecld-anomaly**
- mno-specld-anomaly**

When enabled, the compiler ensures that the generated code does not contain speculative loads after jump instructions. If this option is used, `__WORKAROUND_SPECULATIVE_LOADS` is defined.

`-mcsync-anomaly`

`-mno-csync-anomaly`

When enabled, the compiler ensures that the generated code does not contain CSYNC or SSYNC instructions too soon after conditional branches. If this option is used, `__WORKAROUND_SPECULATIVE_SYNCS` is defined.

`-mlow64k`

`-mno-low64k`

When enabled, the compiler is free to take advantage of the knowledge that the entire program fits into the low 64k of memory. The default behavior is to assume that the program is arbitrarily large (`-mno-low64k`).

`-mstack-check-l1`

Do stack checking using information placed into L1 scratchpad memory by the uClinux kernel.

`-mid-shared-library`

`-mno-id-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`. The default is `-mno-id-shared-library`, to generate code that doesn't assume ID-based shared libraries are being used.

`-mleaf-id-shared-library`

`-mno-leaf-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.

The default is `-mno-leaf-id-shared-library`, in which the no assumption is made that the code being compiled won't link against any ID shared libraries. Slower code is generated for jump and call insns.

`-mshared-library-id=n`

Specifies the identification number of the ID-based shared library being compiled. Specifying a value of 0 generates more compact code; specifying other values forces the allocation of that number to the current library but is no more space- or time-efficient than omitting this option.

`-msep-data`

`-mno-sep-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. The default is `-mno-sep-data`, which tells GCC to generate code that assumes that the data segment follows the text segment.

`-mlong-calls`

`-mno-long-calls`

Tells the compiler to perform function calls by first loading the address of the function into a register and then performing a subroutine call on this register.

This switch is needed if the target function lies outside of the 24-bit addressing range of the offset-based version of subroutine call instruction.

This feature is not enabled by default. Specifying `-mno-long-calls` restores the default behavior. Note these switches have no effect on how the compiler generates code to handle function calls via function pointers.

-mfast-fp

Link with the fast floating-point library. This library relaxes some of the IEEE floating-point standard's rules for checking inputs against Not-a-Number (NaN), in the interest of performance.

-minline-plt

Enable inlining of PLT entries in function calls to functions that are not known to bind locally. It has no effect without `-mfdpic`.

-mmulticore

Build a standalone application for multicore Blackfin processors. This option causes proper start files and link scripts supporting multicore to be used, and defines the macro `__BFIN_MULTICORE`. It can only be used with `-mcpu=bf561[-sirevision]`.

This option can be used with `-mcorea` or `-mcoreb`, which selects the one-application-per-core programming model. Without `-mcorea` or `-mcoreb`, the single-application/dual-core programming model is used. In this model, the main function of Core B should be named as `coreb_main`.

If this option is not used, the single-core application programming model is used.

-mcorea

Build a standalone application for Core A of BF561 when using the one-application-per-core programming model. Proper start files and link scripts are used to support Core A, and the macro `__BFIN_COREA` is defined. This option can only be used in conjunction with `-mmulticore`.

-mcoreb

Build a standalone application for Core B of BF561 when using the one-application-per-core programming model. Proper start files and link scripts are used to support Core B, and the macro `__BFIN_COREB` is defined. When this option is used, `coreb_main` should be used instead of `main`. This option can only be used in conjunction with `-mmulticore`.

-msdram

Build a standalone application for SDRAM. Proper start files and link scripts are used to put the application into SDRAM, and the macro `__BFIN_SDRAM` is defined. The loader should initialize SDRAM before loading the application.

-micplb

Assume that ICPLBs are enabled at run time. This has an effect on certain anomaly workarounds. For Linux targets, the default is to assume ICPLBs are enabled; for standalone applications the default is off.

3.20.8 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'`.

`-mdsp`
`-mno-dsp`
`-medsp`
`-mno-edsp`
`-mvdsp`
`-mno-vdsp`
 Enable C-SKY DSP, Enhanced DSP, or Vector DSP instructions, respectively. All of these options default to off.

`-mdiv`
`-mno-div` Generate divide instructions. Default is off.

`-msmart`
`-mno-smart`
 Generate code for Smart Mode, using only registers numbered 0-7 to allow use of 16-bit instructions. This option is ignored for CK801 where this is the required behavior, and it defaults to on for CK802. For other targets, the default is off.

`-mhigh-registers`
`-mno-high-registers`
 Generate code using the high registers numbered 16-31. This option is not supported on CK801, CK802, or CK803, and is enabled by default for other processors.

`-manchor`
`-mno-anchor`
 Generate code using global anchor symbol addresses.

`-mpushpop`
`-mno-pushpop`
 Generate code using `push` and `pop` instructions. This option defaults to on.

`-mmultiple-stld`
`-mno-multiple-stld`
 Generate code using `stm` and `ldm` instructions. This option isn't supported on CK801 but is enabled by default on other processors.

`-mconstpool`
`-mno-constpool`
 Create constant pools in the compiler instead of deferring it to the assembler. This option is the default and required for correct code generation on CK801 and CK802, and is optional on other processors.

`-mstack-size`
`-mno-stack-size`
 Emit `.stack_size` directives for each function in the assembly output. This option defaults to off.

`-mccrt`
`-mno-ccrt`
 Generate code for the C-SKY compiler runtime instead of `libgcc`. This option defaults to off.

In addition a C preprocessor macro is defined, based upon the setting of this option. Possible values are: `__RL78_MUL_NONE__`, `__RL78_MUL_G13__` or `__RL78_MUL_G14__`.

`-mcpu=g10`
`-mcpu=g13`
`-mcpu=g14`
`-mcpu=r178`

Specifies the RL78 core to target. The default is the G14 core, also known as an S3 core or just RL78. The G13 or S2 core does not have multiply or divide instructions, instead it uses a hardware peripheral for these operations. The G10 or S1 core does not have register banks, so it uses a different calling convention.

If this option is set it also selects the type of hardware multiply support to use, unless this is overridden by an explicit `-mmul=none` option on the command line. Thus specifying `-mcpu=g13` enables the use of the G13 hardware multiply peripheral and specifying `-mcpu=g10` disables the use of hardware multiplications altogether.

Note, although the RL78/G14 core is the default target, specifying `-mcpu=g14` or `-mcpu=r178` on the command line does change the behavior of the toolchain since it also enables G14 hardware multiply support. If these options are not specified on the command line, then software multiplication routines are used even though the code targets the RL78 core. This is for backward compatibility with older toolchains which did not have hardware multiply and divide support.

In addition a C preprocessor macro is defined, based upon the setting of this option. Possible values are: `__RL78_G10__`, `__RL78_G13__` or `__RL78_G14__`.

`-mg10`
`-mg13`
`-mg14`
`-mr178`

These are aliases for the corresponding `-mcpu=` option. They are provided for backwards compatibility.

`-mallregs`

Allow the compiler to use all of the available registers. By default registers `r24..r31` are reserved for use in interrupt handlers. With this option enabled these registers can be used in ordinary functions as well.

`-mrelax`
`-mno-relax`

Enable/disable assembler and linker relaxation. This is enabled by default at `-Os`.

`-mes0` Assume ES is zero throughout program execution, and use it to address read-only data.

`-msave-mduc-in-interrupts`
`-mno-save-mduc-in-interrupts`

Specifies that interrupt handler functions should preserve the MDUC registers. This is only necessary if normal code might use the MDUC registers, for example

because it performs multiplication and division operations. The default is to ignore the MDUC registers as this makes the interrupt handlers faster. The target option `-mg13` needs to be passed for this to work as this feature is only available on the G13 target (S2 core). The MDUC registers are only saved if the interrupt handler performs a multiplication or division operation or it calls another function.

3.20.43 IBM RS/6000 and PowerPC Options

These ‘-m’ options are defined for the IBM RS/6000 and PowerPC:

`-mcpu=cpu_type`

Set architecture type, register usage, and instruction scheduling parameters for machine type *cpu_type*. Supported values for *cpu_type* are ‘401’, ‘403’, ‘405’, ‘405fp’, ‘440’, ‘440fp’, ‘464’, ‘464fp’, ‘476’, ‘476fp’, ‘505’, ‘601’, ‘602’, ‘603’, ‘603e’, ‘604’, ‘604e’, ‘620’, ‘630’, ‘740’, ‘7400’, ‘7450’, ‘750’, ‘801’, ‘821’, ‘823’, ‘860’, ‘970’, ‘8540’, ‘a2’, ‘e300c2’, ‘e300c3’, ‘e500mc’, ‘e500mc64’, ‘e5500’, ‘e6500’, ‘ec603e’, ‘G3’, ‘G4’, ‘G5’, ‘titan’, ‘power3’, ‘power4’, ‘power5’, ‘power5+’, ‘power6’, ‘power6x’, ‘power7’, ‘power8’, ‘power9’, ‘power10’, ‘power11’, ‘future’, ‘powerpc’, ‘powerpc64’, ‘powerpc64le’, ‘rs64’, and ‘native’.

`-mcpu=powerpc`, `-mcpu=powerpc64`, and `-mcpu=powerpc64le` specify pure 32-bit PowerPC (either endian), 64-bit big endian PowerPC and 64-bit little endian PowerPC architecture machine types, with an appropriate, generic processor model assumed for scheduling purposes.

Specifying ‘native’ as cpu type detects and selects the architecture option that corresponds to the host processor of the system performing the compilation. `-mcpu=native` has no effect if GCC does not recognize the processor.

The other options specify a specific processor. Code generated under those options runs best on that processor, and may not run at all on others.

The `-mcpu` options automatically enable or disable the following options:

```
-maltivec -mfprnd -mhard-float -mmfcrf -mmultiple
-mpopcntb -mpopcntd -mpowerpc64
-mpowerpc-gpopt -mpowerpc-gfxopt
-mmulhw -mdlmzb -mmfpgpr -mvsx
-mcrypto -mhtm -mpower8-fusion
-mquad-memory -mquad-memory-atomic -mfloat128
-mfloat128-hardware -mprefixed -mpcrel -mma
-mrop-protect
```

The particular options set for any particular CPU varies between compiler versions, depending on what setting seems to produce optimal code for that CPU; it doesn’t necessarily reflect the actual hardware’s capabilities. If you wish to set an individual option to a particular value, you may specify it after the `-mcpu` option, like `-mcpu=970 -mno-altivec`.

On AIX, the `-maltivec` and `-mpowerpc64` options are not enabled or disabled by the `-mcpu` option at present because AIX does not have full support for these options. You may still enable or disable them individually if you’re sure it’ll work in your environment.

- 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 Lichtblau 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 CAAO integration, `fdlibm` updates.
- Gael Thomas for `VMClassLoader` boot packages support suggestions.

- Andreas Tobler for Darwin and Solaris testing and fixing, `Qt4` support for Darwin / macOS, `Graphics2D` support, `gtk+` updates.
- Dalibor Topic for better `DEBUG` support, build cleanups and Kaffe integration. `Qt4` build infrastructure, `SHA1PRNG` and `GdkPixbufDecoder` 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 `GdkPixbufDecoder` 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 Billingham
- 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.

Appendix A Indices

A.1 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 `-foption` 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.

#

..... 41

A

all-warnings 104
 all_load 384
 allowable_client 385
 analyzer 170
 ansi 3, 45, 797, 1113
 arch 382
 arch_errors_fatal 384
 asm_macosx_version_min 383
 assemble 36
 aux-info 48

B

bind_at_load 384
 B 285
 Bdynamic 512
 Bstatic 512
 bundle 384
 bundle_loader 384

C

c 36, 276
 CC 273
 client_name 385
 comments 273
 comments-in-macros 273
 compatibility_version 385
 compile 36
 compile-std-module 52
 coverage 246
 crt0 447
 current_version 385
 C 273

D

d 274, 298
 da 301
 dA 301
 dD 274, 301
 dead_strip 385
 debug 189
 define-macro 267
 dependencies 268
 dependency-file 382
 dH 301
 dI 274
 dM 274
 dN 274
 dp 301
 dP 301
 dump 274, 298
 dumpbase 38
 dumpbase-ext 39
 dumpdir 39
 dumpfullversion 316
 dumpmachine 315
 dumpspecs 316
 dumpversion 316
 dU 274
 dx 301
 dylib_file 385
 dylinker 385
 dylinker_install_name 385
 dynamic 385
 dynamiclib 384
 D 267

E

e 278
 embed-dir 284
 embed-directory 284
 entry 278
 exported_symbols_list 385
 extra-warnings 105
 E 36, 276
 EB 420, 441
 EL 420, 441

F

fabi-compat-version.....	54	fcommon.....	289, 603
fabi-version.....	52	fcompare-debug.....	311
faccess-control.....	54	fcompare-debug-second.....	311
fada-spec-parent.....	44	fcompare-elim.....	236
faggressive-loop-optimizations.....	206	fconcepts.....	55
falign-functions.....	227	fconcepts-diagnostics-depth.....	56
falign-jumps.....	229	fcond-mismatch.....	49
falign-labels.....	228	fcondition-coverage.....	246
falign-loops.....	228	fconserve-stack.....	213
faligned-new.....	54	fconstant-cfstrings.....	383
fallocation-dce.....	229	fconstant-string-class.....	82
fallow-store-data-races.....	229	fconstexpr-cache-depth.....	56
fanalyzer.....	170	fconstexpr-depth.....	56
fanalyzer-assume-nothrow.....	185	fconstexpr-fp-exception.....	56
fanalyzer-call-summaries.....	185	fconstexpr-loop-limit.....	56
fanalyzer-checker.....	185	fconstexpr-ops-limit.....	56
fanalyzer-debug-text-art.....	185	fcontract-checks-outlined.....	57
fanalyzer-feasibility.....	185	fcontract-disable-optimized-checks.....	57
fanalyzer-fine-grained.....	186	fcontract-evaluation-semantic.....	57
fanalyzer-show-duplicate-count.....	186	fcontracts.....	56
fanalyzer-show-events-in- system-headers.....	186	fcontracts-client-check.....	57
fanalyzer-simplify-supergraph.....	186	fcontracts-conservative-ipa.....	57
fanalyzer-state-merge.....	186	fcontracts-definition-check.....	58
fanalyzer-state-purge.....	186	fcoroutines.....	58
fanalyzer-suppress-followups.....	186	fcprop-registers.....	236
fanalyzer-transitivity.....	187	fcrossjumping.....	207
fanalyzer-undo-inlining.....	187	fcse-follow-jumps.....	205
fanalyzer-verbose-edges.....	187	fcse-skip-blocks.....	205
fanalyzer-verbose-state-changes.....	187	fcx-fortran-rules.....	241
fanalyzer-verbosity.....	187	fcx-limited-range.....	241
fapple-kext.....	382	fcx-method.....	241
fasan-shadow-offset.....	257	fdata-sections.....	244
fasm.....	48	fdbg-cnt.....	314
fassociative-math.....	239	fdbg-cnt-list.....	314
fassume-sane-operators-new-delete.....	54	fdce.....	207
fasynchronous-unwind-tables.....	288	fdebug-cpp.....	275
fauto-inc-dec.....	207	fdebug-prefix-map.....	191
fauto-profile.....	237	fdebug-types-section.....	193
fauto-profile-inlining.....	237	fdeclone-ctor-dtor.....	207
favoid-store-forwarding.....	200	fdefer-pop.....	200
fbit-tests.....	292	fdelayed-branch.....	210
fbranch-count-reg.....	204	fdelete-dead-exceptions.....	288
fbranch-probabilities.....	241	fdelete-null-pointer-checks.....	207
fbuiltin.....	48	fdep-fusion.....	242
fcall-saved.....	293	fdeps-.....	50
fcall-used.....	293	fdeps-file.....	50
fcaller-saves.....	213	fdeps-format.....	50
fcallgraph-info.....	297	fdeps-target.....	50
fcanon-prefix-map.....	43	fdevirtualize.....	208
fcannotical-system-headers.....	271	fdevirtualize-at-ltrans.....	208
fcf-protection.....	259	fdevirtualize-speculatively.....	208
fchar8_t.....	55	fdiagnostics-add-output.....	97
fcheck-new.....	55	fdiagnostics-add- output=experimental-html.....	99
fchecking.....	310	fdiagnostics-add-output=sarif.....	98
fcode-hoisting.....	214	fdiagnostics-add-output=text.....	98
fcombine-stack-adjustments.....	213	fdiagnostics-all-candidates.....	58
		fdiagnostics-color.....	88

fdiagnostics-column-origin.....	96	fdump-lang.....	304
fdiagnostics-column-unit.....	96	fdump-lang-all.....	304
fdiagnostics-escape-format.....	96	fdump-lang-class.....	58
fdiagnostics-format.....	97	fdump-lang-module.....	58
fdiagnostics-format=sarif-file.....	97	fdump-lang-raw.....	58
fdiagnostics-format=sarif-stderr.....	97	fdump-lang-tinst.....	58
fdiagnostics-format=text.....	97	fdump-noaddr.....	302
fdiagnostics-generate-patch.....	93	fdump-passes.....	304
fdiagnostics-json-formatting.....	100	fdump-rtl-alignments.....	298
fdiagnostics-minimum-margin-width.....	92	fdump-rtl-all.....	301
fdiagnostics-parseable-fixits.....	93	fdump-rtl-asmcons.....	298
fdiagnostics-path-format.....	94	fdump-rtl-auto_inc_dec.....	298
fdiagnostics-plain-output.....	88	fdump-rtl-barriers.....	298
fdiagnostics-set-output.....	100	fdump-rtl-bbpart.....	298
fdiagnostics-show-caret.....	91	fdump-rtl-bbro.....	298
fdiagnostics-show-context.....	92	fdump-rtl-btl2.....	298
fdiagnostics-show-cwe.....	91	fdump-rtl-bypass.....	298
fdiagnostics-show-event-links.....	91	fdump-rtl-ce1.....	299
fdiagnostics-show-highlight-colors.....	92	fdump-rtl-ce2.....	299
fdiagnostics-show-labels.....	91	fdump-rtl-ce3.....	299
fdiagnostics-show-line-numbers.....	92	fdump-rtl-combine.....	298
fdiagnostics-show-location.....	88	fdump-rtl-compotos.....	298
fdiagnostics-show-nesting.....	97	fdump-rtl-cprop_hardreg.....	299
fdiagnostics-show-nesting-levels.....	97	fdump-rtl-csa.....	299
fdiagnostics-show-nesting-locations.....	97	fdump-rtl-cse1.....	299
fdiagnostics-show-option.....	91	fdump-rtl-cse2.....	299
fdiagnostics-show-path-depths.....	95	fdump-rtl-dbr.....	299
fdiagnostics-show-rules.....	92	fdump-rtl-dce.....	299
fdiagnostics-show-template-tree.....	93	fdump-rtl-dce1.....	299
fdiagnostics-text-art-charset.....	96	fdump-rtl-dce2.....	299
fdiagnostics-urls.....	90	fdump-rtl-dfinish.....	301
fdirectives-only.....	270	fdump-rtl-dfinit.....	301
fdisable-.....	308	fdump-rtl-eh.....	299
fdollars-in-identifiers.....	271, 1103	fdump-rtl-eh_ranges.....	299
fdse.....	207	fdump-rtl-expand.....	299
fdump-ada-spec.....	44	fdump-rtl-fwprop1.....	299
fdump-ada-spec-slim.....	44	fdump-rtl-fwprop2.....	299
fdump-analyzer.....	188	fdump-rtl-gcse1.....	299
fdump-analyzer-callgraph.....	188	fdump-rtl-gcse2.....	299
fdump-analyzer-exploded-graph.....	188	fdump-rtl-init-regs.....	299
fdump-analyzer-exploded-nodes.....	188	fdump-rtl-initvals.....	299
fdump-analyzer-exploded-nodes-2.....	188	fdump-rtl-into_cfglayout.....	299
fdump-analyzer-exploded-nodes-3.....	188	fdump-rtl-ira.....	300
fdump-analyzer-exploded-paths.....	188	fdump-rtl-jump.....	300
fdump-analyzer-feasibility.....	188	fdump-rtl-loop2.....	300
fdump-analyzer-infinite-loop.....	188	fdump-rtl-mach.....	300
fdump-analyzer-json.....	188	fdump-rtl-mode_sw.....	300
fdump-analyzer-state-purge.....	188	fdump-rtl-outof_cfglayout.....	300
fdump-analyzer-stderr.....	188	fdump-rtl-pass.....	298
fdump-analyzer-supergraph.....	188	fdump-rtl-peekhole2.....	300
fdump-analyzer-untracked.....	189	fdump-rtl-postreload.....	300
fdump-debug.....	302	fdump-rtl-pro_and_epilogue.....	300
fdump-earlydebug.....	302	fdump-rtl-ree.....	300
fdump-final-insns.....	311	fdump-rtl-regclass.....	301
fdump-go-spec.....	44	fdump-rtl-rnreg.....	300
fdump-internal-locations.....	302	fdump-rtl-sched1.....	300
fdump-ipa.....	302	fdump-rtl-sched2.....	300
fdump-ipa-clones.....	303	fdump-rtl-seqabstr.....	300

fdump-rtl-shorten.....	300	fgcse-after-reload.....	206
fdump-rtl-sms.....	300	fgcse-las.....	206
fdump-rtl-split1.....	300	fgcse-lm.....	206
fdump-rtl-split2.....	300	fgcse-sm.....	206
fdump-rtl-split3.....	300	fgimple.....	49
fdump-rtl-split4.....	300	fgnu-keywords.....	59
fdump-rtl-split5.....	300	fgnu-runtime.....	83
fdump-rtl-stack.....	301	fgnu-tm.....	49
fdump-rtl-subreg1.....	301	fgnu-unique.....	288
fdump-rtl-subreg2.....	301	fgnu89-inline.....	49
fdump-rtl-subregs_of_mode_finish.....	301	fgraphite-identity.....	219
fdump-rtl-subregs_of_mode_init.....	301	fguess-branch-probability.....	225
fdump-rtl-vartrack.....	301	fhardcfr-check-exceptions.....	260
fdump-rtl-vregs.....	301	fhardcfr-check-noreturn-calls.....	260
fdump-rtl-web.....	301	fhardcfr-check-returning-calls.....	260
fdump-statistics.....	304	fhardcfr-skip-leaf.....	260
fdump-tree.....	304	fharden-compares.....	259
fdump-tree-all.....	304	fharden-conditional-branches.....	259
fdump-unnumbered.....	302	fharden-control-flow-redundancy.....	260
fdump-unnumbered-links.....	302	fhardened.....	261
fdwarf2-cfi-asm.....	196	fhoist-adjacent-loads.....	214
fearly-inlining.....	203	fhosted.....	49
felide-constructors.....	58	fident.....	290
felide-type.....	94	fif-conversion.....	207
feliminate-unused-debug-symbols.....	191	fif-conversion2.....	207
feliminate-unused-debug-types.....	196	filelist.....	385
femit-class-debug-always.....	191	fimmediate-escalation.....	59
femit-struct-debug-baseonly.....	195	fimplement-inlines.....	60
femit-struct-debug-detailed.....	195	fimplicit-constexpr.....	60
femit-struct-debug-reduced.....	195	fimplicit-inline-templates.....	60
fenable-.....	308	fimplicit-templates.....	60
fenforce-eh-specs.....	59	findirect-data.....	383
fexceptions.....	288	findirect-inlining.....	202
fexcess-precision.....	238	finhibit-size-directive.....	290
fexec-charset.....	271	finline.....	202
fexpensive-optimizations.....	208	finline-atomics.....	202
fext-dce.....	208	finline-functions.....	202
fext-numeric-literals.....	64	finline-functions-called-once.....	203
fextended-identifiers.....	271	finline-limit.....	203
fextern-tls-init.....	59	finline-small-functions.....	202
ffast-math.....	238	finline-stringops.....	202
ffat-lto-objects.....	235	finput-charset.....	272
ffile-prefix-map.....	43	finstrument-functions.....	265, 620
ffinite-loops.....	218	finstrument-functions- exclude-file-list.....	266
ffinite-math-only.....	240	finstrument-functions-exclude- function-list.....	266
ffix-and-continue.....	383	finstrument-functions-once.....	265
ffixed.....	293	fipa-bit-cp.....	216
ffloat-store.....	238, 1108	fipa-cp.....	215
ffold-mem-offsets.....	236	fipa-cp-clone.....	215
ffold-simple-inlines.....	59	fipa-icf.....	216
fforward-propagate.....	200	fipa-icf-functions.....	216
ffp-contract.....	201	fipa-icf-variables.....	216
ffp-int-builtin-inexact.....	201	fipa-modref.....	215
ffreestanding.....	4, 49, 109, 611	fipa-profile.....	215
ffunction-cse.....	204	fipa-pta.....	215
ffunction-sections.....	244	fipa-pure-const.....	214
ffuse-ops-with-volatile-access.....	205		
fgcse.....	206		

fipa-ra	213	fmem-report	312
fipa-reference	214	fmem-report-wpa	313
fipa-reference-addressable	214	fmerge-all-constants	204
fipa-reorder-for-locality	215	fmerge-constants	204
fipa-sra	203	fmerge-debug-strings	191
fipa-stack-alignment	215	fmessage-length	88
fipa-strict-aliasing	227	fmin-function-alignment= <i>n</i>	229
fipa-vrp	216	fmodule-header	60
fira-algorithm	209	fmodule-implicit-inline	61
fira-hoist-pressure	209	fmodule-lazy	61
fira-loop-pressure	209	fmodule-mapper	61
fira-region	209	fmodule-only	61
fira-share-save-slots	209	fmodules	60
fira-share-spill-slots	209	fmodulo-sched	204
fira-verbose	312	fmodulo-sched-allow-regmoves	204
fisolate-erroneous-paths-attribute	217	fmove-loop-invariants	243
fisolate-erroneous-paths-dereference	217	fmove-loop-stores	243
fivar-visibility	85	fms-extensions	50, 61, 585
fivopts	221	fmultiflags	313
fjump-tables	292	fnew-inheriting-ctors	61
fkeep-inline-dllexport	203	fnew-ttp-matching	61
fkeep-inline-functions	203, 719	fnex-runtime	83
fkeep-static-consts	204	fnil-receivers	83
fkeep-static-functions	204	fno-access-control	54
flang-info-include-translate	65	fno-aggressive-loop-optimizations	206
flang-info-include-translate-not	65	fno-align-functions	227
flang-info-module-cmi	65	fno-align-jumps	229
flat_namespace	385	fno-align-labels	228
flate-combine-instructions	216	fno-align-loops	228
flax-vector-conversions	49	fno-aligned-new	54
fleading-underscore	293	fno-allocation-dce	229
flifetime-dse	208	fno-allow-store-data-races	229
flimit-function-alignment	228	fno-analyzer	170
flink-libatomic	276	fno-analyzer-assume-nothrow	185
flinker-output	276	fno-analyzer-call-summaries	185
flive-patching	216	fno-analyzer-debug-text-art	185
flive-range-shrinkage	209	fno-analyzer-feasibility	185
flocal-ivars	85	fno-analyzer-fine-grained	186
floop-block	219	fno-analyzer-show-duplicate-count	186
floop-interchange	220	fno-analyzer-show-events-in- system-headers	186
floop-nest-optimize	219	fno-analyzer-simplify-supergraph	186
floop-parallelize-all	219	fno-analyzer-state-merge	186
floop-strip-mine	219	fno-analyzer-state-purge	186
floop-unroll-and-jam	221	fno-analyzer-suppress-followups	186
flra-remat	210	fno-analyzer-suppress-followups	186
flto	231	fno-analyzer-transitivity	187
flto-compression-level	234	fno-analyzer-undo-inlining	187
flto-incremental	234	fno-analyzer-verbose-edges	187
flto-incremental-cache-size	234	fno-analyzer-verbose-state-changes	187
flto-partition	234	fno-apple-kext	382
flto-report	312	fno-asm	48
flto-report-wpa	312	fno-associative-math	239
flto-toplevel-asm-heuristics	235	fno-assume-sane-operators-new-delete	54
fmacro-prefix-map	271	fno-asynchronous-unwind-tables	288
fmalloc-dce	243	fno-auto-inc-dec	207
fmath-errno	238	fno-auto-profile	237
fmax-errors	101	fno-auto-profile-inlining	237
fmax-include-depth	271	fno-avoid-store-forwarding	200

fno-bit-tests	292	fno-diagnostics-show-nesting	97
fno-branch-count-reg	204	fno-diagnostics-show-nesting-levels	97
fno-branch-probabilities	241	fno-diagnostics-show-nesting-locations	97
fno-builtin	48, 109, 611, 797	fno-diagnostics-show-option	91
fno-caller-saves	213	fno-diagnostics-show-path-depths	95
fno-canon-prefix-map	43	fno-diagnostics-show-rules	92
fno-canonical-system-headers	271	fno-diagnostics-show-template-tree	93
fno-char8_t	55	fno-directives-only	270
fno-check-new	55	fno-dollars-in-identifiers	271
fno-checking	310	fno-dse	207
fno-code-hoisting	214	fno-dump-internal-locations	302
fno-combine-stack-adjustments	213	fno-dump-noaddr	302
fno-common	289, 603	fno-dump-passes	304
fno-compare-debug	311	fno-dump-unnumbered	302
fno-compare-elim	236	fno-dump-unnumbered-links	302
fno-concepts	55	fno-dwarf2-cfi-asm	196
fno-cond-mismatch	49	fno-early-inlining	203
fno-condition-coverage	246	fno-elide-constructors	58
fno-consume-stack	213	fno-elide-type	94
fno-constant-cfstrings	383	fno-eliminate-unused-debug-symbols	191
fno-constexpr-fp-except	56	fno-eliminate-unused-debug-types	196
fno-contract-checks-outlined	57	fno-emit-class-debug-always	191
fno-contract-disable-optimized-checks	57	fno-enforce-eh-specs	59
fno-contracts	56	fno-exceptions	288
fno-contracts-conservative-ipa	57	fno-expensive-optimizations	208
fno-coroutines	58	fno-ext-dce	208
fno-cprop-registers	236	fno-ext-numeric-literals	64
fno-crossjumping	207	fno-extended-identifiers	271
fno-cse-follow-jumps	205	fno-extern-tls-init	59
fno-cx-fortran-rules	241	fno-fast-math	238
fno-cx-limited-range	241	fno-fat-lto-objects	235
fno-data-sections	244	fno-finite-loops	218
fno-dbg-cnt-list	314	fno-finite-math-only	240
fno-dce	207	fno-float-store	238
fno-debug-cpp	275	fno-fold-mem-offsets	236
fno-debug-types-section	193	fno-fold-simple-inlines	59
fno-declone-ctor-dtor	207	fno-forward-propagate	200
fno-default-inline	719	fno-fp-int-builtin-inexact	201
fno-defer-pop	200	fno-freestanding	49, 109
fno-delayed-branch	210	fno-function-cse	204
fno-delete-dead-exceptions	288	fno-function-sections	244
fno-delete-null-pointer-checks	207	fno-fuse-ops-with-volatile-access	205
fno-dep-fusion	242	fno-gcse	206
fno-devirtualize	208	fno-gcse-after-reload	206
fno-devirtualize-at-ltrans	208	fno-gcse-las	206
fno-devirtualize-speculatively	208	fno-gcse-lm	206
fno-diagnostics-all-candidates	58	fno-gcse-sm	206
fno-diagnostics-color	88	fno-gimple	49
fno-diagnostics-generate-patch	93	fno-gnu-keywords	59
fno-diagnostics-json-formatting	100	fno-gnu-tm	49
fno-diagnostics-parseable-fixits	93	fno-gnu-unique	288
fno-diagnostics-show-caret	91	fno-gnu89-inline	49
fno-diagnostics-show-context	92	fno-graphite-identity	219
fno-diagnostics-show-cwe	91	fno-guess-branch-probability	225
fno-diagnostics-show-event-links	91	fno-hardcfr-check-exceptions	260
fno-diagnostics-show-highlight-colors	92	fno-hardcfr-check-returning-calls	260
fno-diagnostics-show-labels	91	fno-hardcfr-skip-leaf	260
fno-diagnostics-show-line-numbers	92	fno-harden-compares	259

fno-harden-conditional-branches.....	259	fno-lax-vector-conversions.....	49
fno-harden-control-flow-redundancy.....	260	fno-leading-underscore.....	293
fno-hardened.....	261	fno-lifetime-dse.....	208
fno-hoist-adjacent-loads.....	214	fno-limit-function-alignment.....	228
fno-hosted.....	49	fno-link-libatomic.....	276
fno-ident.....	290	fno-live-range-shrinkage.....	209
fno-if-conversion.....	207	fno-local-ivars.....	85
fno-if-conversion2.....	207	fno-loop-block.....	219
fno-immediate-escalation.....	59	fno-loop-interchange.....	220
fno-implement-inlines.....	60, 1034	fno-loop-nest-optimize.....	219
fno-implicit-constexpr.....	60	fno-loop-parallelize-all.....	219
fno-implicit-inline-templates.....	60	fno-loop-strip-mine.....	219
fno-implicit-templates.....	60, 1036	fno-loop-unroll-and-jam.....	221
fno-indirect-inlining.....	202	fno-lra-remat.....	210
fno-inhibit-size-directive.....	290	fno-lto.....	231
fno-inline.....	202	fno-lto-incremental.....	234
fno-inline-atomics.....	202	fno-lto-incremental-cache-size.....	234
fno-inline-functions.....	202	fno-lto-partition.....	234
fno-inline-functions-called-once.....	203	fno-lto-report.....	312
fno-inline-small-functions.....	202	fno-lto-report-wpa.....	312
fno-inline-stringops.....	202	fno-malloc-dce.....	243
fno-instrument-functions.....	265	fno-math-errno.....	238
fno-instrument-functions-once.....	265	fno-mem-report.....	312
fno-ipa-bit-cp.....	216	fno-mem-report-wpa.....	313
fno-ipa-cp.....	215	fno-merge-all-constants.....	204
fno-ipa-cp-clone.....	215	fno-merge-constants.....	204
fno-ipa-icf.....	216	fno-merge-debug-strings.....	191
fno-ipa-icf-functions.....	216	fno-module-implicit-inline.....	61
fno-ipa-icf-variables.....	216	fno-module-lazy.....	61
fno-ipa-modref.....	215	fno-modules.....	60
fno-ipa-profile.....	215	fno-modulo-sched.....	204
fno-ipa-pta.....	215	fno-modulo-sched-allow-regmoves.....	204
fno-ipa-pure-const.....	214	fno-move-loop-invariants.....	243
fno-ipa-ra.....	213	fno-move-loop-stores.....	243
fno-ipa-reference.....	214	fno-ms-extensions.....	50, 61
fno-ipa-reference-addressable.....	214	fno-new-inheriting-ctors.....	61
fno-ipa-reorder-for-locality.....	215	fno-new-ttp-matching.....	61
fno-ipa-sra.....	203	fno-nil-receivers.....	83
fno-ipa-stack-alignment.....	215	fno-non-call-exceptions.....	288
fno-ipa-strict-aliasing.....	227	fno-nonansi-builtins.....	61
fno-ipa-vrp.....	216	fno-nothrow-opt.....	61
fno-ira-hoist-pressure.....	209	fno-objc-call-cxx-ctors.....	83
fno-ira-loop-pressure.....	209	fno-objc-direct-dispatch.....	83
fno-ira-share-save-slots.....	209	fno-objc-exceptions.....	84
fno-ira-share-spill-slots.....	209	fno-objc-gc.....	84
fno-isolate-erroneous-paths-attribute.....	217	fno-objc-nilcheck.....	84
fno-isolate-erroneous- paths-dereference.....	217	fno-omit-frame-pointer.....	201
fno-ivopts.....	221	fno-openacc.....	87
fno-jump-tables.....	292	fno-openmp.....	87
fno-keep-inline-dllexport.....	203	fno-openmp-simd.....	87
fno-keep-inline-functions.....	203	fno-openmp-target-simd-clone.....	87
fno-keep-static-consts.....	204	fno-operator-names.....	62
fno-keep-static-functions.....	204	fno-opt-info.....	306
fno-lang-info-include-translate.....	65	fno-optimize-crc.....	201
fno-lang-info-include-translate-not.....	65	fno-optimize-sibling-calls.....	202
fno-lang-info-module-cmi.....	65	fno-optimize-strlen.....	202
fno-late-combine-instructions.....	216	fno-optional-diags.....	62
		fno-pack-struct.....	293

fno-partial-inlining	224	fno-sched-dep-count-heuristic	212
fno-path-coverage	246	fno-sched-group-heuristic	211
fno-pcc-struct-return	289	fno-sched-interblock	210
fno-pch-deps	272	fno-sched-last-insn-heuristic	212
fno-pch-preprocess	272	fno-sched-pressure	210
fno-peel-loops	243	fno-sched-rank-heuristic	212
fno-peephole	225	fno-sched-spec	210
fno-peephole2	225	fno-sched-spec-insn-heuristic	211
fno-permissive	103	fno-sched-spec-load	210
fno-pic	291	fno-sched-spec-load-dangerous	211
fno-PIC	292	fno-sched-stalled-insns	211
fno-pie	292	fno-sched-stalled-insns-dep	211
fno-PIE	292	fno-sched2-use-superblocks	211
fno-plan9-extensions	51	fno-schedule-fusion	242
fno-plt	292	fno-schedule-insns	210
fno-post-ipa-mem-report	313	fno-schedule-insns2	210
fno-pre-ipa-mem-report	313	fno-search-include-path	271
fno-predictive-commoning	224	fno-section-anchors	244
fno-prefetch-loop-arrays	224	fno-sel-sched-pipelining	212
fno-preprocessed	270	fno-sel-sched-pipelining-outer-loops	212
fno-pretty-templates	62	fno-selective-scheduling	212
fno-printf-return-value	225	fno-selective-scheduling2	212
fno-profile	245	fno-semantic-interposition	212
fno-profile-abs-path	248	fno-set-stack-executable	381
fno-profile-arcs	245	fno-short-enums	289
fno-profile-correction	236	fno-short-wchar	289
fno-profile-generate	248	fno-show-column	95
fno-profile-partial-training	236	fno-shrink-wrap	213
fno-profile-reorder-functions	242	fno-shrink-wrap-separate	213
fno-profile-report	313	fno-signaling-nans	240
fno-profile-use	236	fno-signed-bitfields	51
fno-profile-values	242	fno-signed-char	51
fno-random-seed	310	fno-signed-zeros	240
fno-range-for-ext-temps	62	fno-single-precision-constant	241
fno-reciprocal-math	239	fno-sized-deallocation	62
fno-record-gcc-switches	291	fno-speculatively-call- stored-functions	212
fno-ree	208	fno-split-ivs-in-unroller	224
fno-reg-struct-return	289	fno-split-loops	243
fno-rename-registers	242	fno-split-paths	224
fno-reorder-blocks	225	fno-split-stack	263
fno-reorder-blocks-algorithm	226	fno-split-wide-types	205
fno-reorder-blocks-and-partition	226	fno-split-wide-types-early	205
fno-reorder-functions	226	fno-ssa-backprop	218
fno-replace-objc-classes	84	fno-ssa-phiopt	218
fno-report-bug	302	fno-stack-check	262
fno-rerun-cse-after-loop	206	fno-stack-clash-protection	262
fno-reschedule-modulo-scheduled-loops	212	fno-stack-limit	263
fno-rounding-math	240	fno-stack-protector	261
fno-rtti	62	fno-stats	314
fno-sanitize-address-use-after-scope	258	fno-stdarg-opt	244
fno-sanitize-coverage=trace-cmp	258	fno-store-merging	222
fno-sanitize-coverage=trace-pc	258	fno-strict-aliasing	226
fno-sanitize-recover	257	fno-strict-enums	63
fno-sanitize-trap	258	fno-strict-flex-arrays	51
fno-sanitize-undefined-trap-on-error	258	fno-strict-overflow	287
fno-sanitize=all	257	fno-strict-volatile-bitfields	296
fno-save-optimization-record	308	fno-strong-eval-order	63
fno-sched-critical-path-heuristic	211		

fno-sync-libcalls.....	296	fno-use-cxa-atexit.....	63
fno-syntax-only.....	101	fno-use-cxa-get-exception-ptr.....	63
fno-test-coverage.....	248	fno-use-ld=bfd.....	277
fno-thread-jumps.....	205	fno-use-ld=gold.....	277
fno-threadsafe-statics.....	63	fno-use-ld=lld.....	277
fno-time-report.....	312	fno-use-ld=mold.....	277
fno-time-report-details.....	312	fno-use-ld=wild.....	277
fno-toplevel-reorder.....	230	fno-var-tracking.....	192
fno-tracer.....	242	fno-var-tracking-assignments.....	192
fno-trampolines.....	294	fno-var-tracking-assignments-toggle.....	312
fno-trapping-math.....	240	fno-var-tracking-uninit.....	192
fno-trapv.....	287	fno-variable-expansion-in-unroller.....	224
fno-tree-bit-ccp.....	217	fno-vect-cost-model.....	223
fno-tree-builtin-call-dce.....	218	fno-verbose-asm.....	290
fno-tree-ccp.....	218	fno-version-loops-for-strides.....	244
fno-tree-ch.....	219	fno-visibility-inlines-hidden.....	63
fno-tree-coalesce-vars.....	219	fno-visibility-ms-compat.....	64
fno-tree-copy-prop.....	214	fno-vpt.....	242
fno-tree-cselim.....	218	fno-vtv-counts.....	265
fno-tree-dce.....	218	fno-vtv-debug.....	264
fno-tree-dominator-opts.....	218	fno-weak.....	64
fno-tree-dse.....	218	fno-web.....	230
fno-tree-forwprop.....	214	fno-whole-program.....	230
fno-tree-fre.....	214	fno-working-directory.....	272
fno-tree-loop-distribute-patterns.....	220	fno-wrapv.....	287
fno-tree-loop-distribution.....	220	fno-wrapv-pointer.....	287
fno-tree-loop-if-convert.....	219	fno-writable-relocated-rdata.....	381
fno-tree-loop-im.....	221	fno-zero-initialized-in-bss.....	205
fno-tree-loop-ivcanon.....	221	fno-zero-link.....	84
fno-tree-loop-linear.....	219	fnon-call-exceptions.....	288
fno-tree-loop-optimize.....	219	fnonansi-builtins.....	61
fno-tree-loop-vectorize.....	222	fnothrow-opt.....	61
fno-tree-parallelize-loops.....	221	fobjc-abi-version.....	83
fno-tree-partial-pre.....	214	fobjc-call-cxx-ctdors.....	83
fno-tree-phirop.....	214	fobjc-direct-dispatch.....	83
fno-tree-pre.....	214	fobjc-exceptions.....	84
fno-tree-pta.....	222	fobjc-gc.....	84
fno-tree-reassoc.....	214	fobjc-nilcheck.....	84
fno-tree-scev-cprop.....	221	fobjc-std.....	84
fno-tree-sink.....	217	foffload.....	86
fno-tree-slp-vectorize.....	222	foffload-options.....	86
fno-tree-slsr.....	222	fomit-frame-pointer.....	201
fno-tree-sra.....	222	fopenacc.....	87
fno-tree-switch-conversion.....	218	fopenacc-dim.....	87
fno-tree-tail-merge.....	218	fopenmp.....	87
fno-tree-ter.....	222	fopenmp-simd.....	87
fno-tree-vectorize.....	222	fopenmp-target-simd-clone.....	87
fno-tree-vrp.....	224	foperator-names.....	62
fno-unconstrained-commons.....	207	fopt-info.....	306
fno-unit-at-a-time.....	230	foptimize-crc.....	201
fno-unreachable-traps.....	230	foptimize-sibling-calls.....	202
fno-unroll-all-loops.....	243	foptimize-strlen.....	202
fno-unroll-loops.....	243	foptional-diags.....	62
fno-unsafe-math-optimizations.....	239	for-assembler.....	275
fno-unsigned-bitfields.....	51	for-linker.....	281
fno-unsigned-char.....	51	force-link.....	282
fno-unswitch-loops.....	243	force_cpusubtype_ALL.....	384
fno-unwind-tables.....	288	force_flat_namespace.....	385

fpack-struct	293	frename-registers	242
fpartial-inlining	224	freorder-blocks	225
fpatchable-function-entry	266	freorder-blocks-algorithm	226
fpath-coverage	246	freorder-blocks-and-partition	226
fpath-coverage-limit	246	freorder-functions	226
fpcc-struct-return	289, 1105	freplace-objc-classes	84
fpch-deps	272	freport-bug	302
fpch-preprocess	272	frerun-cse-after-loop	206
fpeel-loops	243	freschedule-modulo-scheduled-loops	212
fpeephole	225	frounding-math	240
fpeephole2	225	frtti	62
fpermissive	103	fsanitize-address-use-after-scope	258
fpermitted-flt-eval-methods	50	fsanitize-coverage=trace-cmp	258
fpermitted-flt-eval-methods=c11	50	fsanitize-coverage=trace-pc	258
fpermitted-flt-eval-methods=ts-18661-3	50	fsanitize-recover	257
fpic	291	fsanitize-sections	257
fPIC	292	fsanitize-trap	258
fpie	292	fsanitize-undefined-trap-on-error	258
fPIE	292	fsanitize=address	252
fplan9-extensions	51, 585	fsanitize=alignment	256
fplt	292	fsanitize=bool	256
fplugin	43	fsanitize=bounds	255
fplugin-arg	44	fsanitize=bounds-strict	255
fpost-ipa-mem-report	313	fsanitize=builtin	257
fpre-ipa-mem-report	313	fsanitize=enum	256
fpredictive-commoning	224	fsanitize=float-cast-overflow	256
fprefetch-loop-arrays	224	fsanitize=float-divide-by-zero	256
fpreprocessed	270	fsanitize=hwaddress	252
fpretty-templates	62	fsanitize=integer-divide-by-zero	255
fprintf-return-value	225	fsanitize=kernel-address	252
fprofile	245	fsanitize=kernel-hwaddress	252
fprofile-abs-path	248	fsanitize=leak	254
fprofile-arcs	245, 839	fsanitize=memtag-stack	253
fprofile-correction	236	fsanitize=nonnull-attribute	256
fprofile-dir	248	fsanitize=null	255
fprofile-exclude-files	251	fsanitize=object-size	256
fprofile-filter-files	251	fsanitize=pointer-compare	253
fprofile-generate	248	fsanitize=pointer-overflow	257
fprofile-info-section	248	fsanitize=pointer-subtract	253
fprofile-note	250	fsanitize=return	255
fprofile-partial-training	236	fsanitize=returns-nonnull-attribute	256
fprofile-prefix-map	250	fsanitize=shadow-call-stack	253
fprofile-prefix-path	250	fsanitize=shift	254
fprofile-reorder-functions	242	fsanitize=shift-base	254
fprofile-report	313	fsanitize=shift-exponent	254
fprofile-reproducible	251	fsanitize=signed-integer-overflow	255
fprofile-update	250	fsanitize=thread	254
fprofile-use	236	fsanitize=undefined	254
fprofile-values	242	fsanitize=unreachable	255
fpu	485	fsanitize=vla-bound	255
framework	385	fsanitize=vptr	256
frandom-seed	310	fsave-optimization-record	308
frange-for-ext-temps	62	fsched-critical-path-heuristic	211
freciprocal-math	239	fsched-dep-count-heuristic	212
frecord-gcc-switches	291	fsched-group-heuristic	211
free	208	fsched-interblock	210
freflection	62	fsched-last-insn-heuristic	212
freg-struct-return	289	fsched-pressure	210

fsched-rank-heuristic	212	fstrict-flex-arrays=level	51
fsched-spec	210	fstrict-overflow	287
fsched-spec-insn-heuristic	211	fstrict-volatile-bitfields	296
fsched-spec-load	210	fstrong-eval-order	63
fsched-spec-load-dangerous	211	fstrub=all	264
fsched-stalled-insns	211	fstrub=at-calls	263
fsched-stalled-insns-dep	211	fstrub=disable	263
fsched-verbose	308	fstrub=internal	264
fsched2-use-superblocks	211	fstrub=relaxed	263
fschedule-fusion	242	fstrub=strict	263
fschedule-insns	210	fsync-libcalls	296
fschedule-insns2	210	fsyntax-only	101
fsearch-include-path	271	ftabstop	271
fsection-anchors	244	ftemplate-backtrace-limit	63
fsel-sched-pipelining	212	ftemplate-depth	63
fsel-sched-pipelining-outer-loops	212	ftest-coverage	248
fselective-scheduling	212	fthread-jumps	205
fselective-scheduling2	212	fthreadsafe-statics	63
fsemantic-interposition	212	ftime-report	312
fset-stack-executable	381	ftime-report-details	312
fshort-enums	289, 569, 628, 1112	ftls-model	294
fshort-wchar	289	ftoplevel-reorder	230
fshow-column	95	ftracer	242
fshrink-wrap	213	ftrack-macro-expansion	271
fshrink-wrap-separate	213	ftrampoline-impl	294
fsignaling-nans	240	ftrampolines	294
fsigned-bitfields	51, 1112	ftrapping-math	240
fsigned-char	51, 564	ftrapv	287
fsigned-zeros	240	ftree-bit-ccp	217
fsimd-cost-model	224	ftree-builtin-call-dce	218
fsingle-precision-constant	241	ftree-ccp	218
fsized-deallocation	62	ftree-ch	219
fspeculatively-call-stored-functions	212	ftree-coalesce-vars	219
fsplit-ivs-in-unroller	224	ftree-copy-prop	214
fsplit-loops	243	ftree-cselim	218
fsplit-paths	224	ftree-dce	218
fsplit-stack	263, 621	ftree-dominator-opts	218
fsplit-wide-types	205	ftree-dse	218
fsplit-wide-types-early	205	ftree-forwprop	214
fssa-backprop	218	ftree-fre	214
fssa-phiopt	218	ftree-loop-distribute-patterns	220
fsso-struct	52	ftree-loop-distribution	220
fstack-check	262	ftree-loop-if-convert	219
fstack-clash-protection	262	ftree-loop-im	221
fstack-limit-register	263	ftree-loop-ivcanon	221
fstack-limit-symbol	263	ftree-loop-linear	219
fstack-protector	261	ftree-loop-optimize	219
fstack-protector-all	261	ftree-loop-vectorize	222
fstack-protector-explicit	262	ftree-parallelize-loops	221
fstack-protector-strong	261	ftree-partial-pre	214
fstack-reuse	286	ftree-phiprop	214
fstack-usage	313	ftree-pre	214
fstats	314	ftree-pta	222
fstdarg-opt	244	ftree-reassoc	214
fstore-merging	222	ftree-scev-cprop	221
fstrict-aliasing	226	ftree-sink	217
fstrict-enums	63	ftree-slp-vectorize	222
fstrict-flex-arrays	51	ftree-slsr	222

ftree-sra.....	222
ftree-switch-conversion.....	218
ftree-tail-merge.....	218
ftree-ter.....	222
ftree-vectorize.....	222
ftree-vrp.....	224
ftrivial-auto-var-init.....	222
funconstrained-commons.....	207
funit-at-a-time.....	230
funreachable-traps.....	230
funroll-all-loops.....	243
funroll-loops.....	243
funsafe-math-optimizations.....	239
funsigned-bitfields.....	51, 568, 1112
funsigned-char.....	51, 564
funswitch-loops.....	243
funwind-tables.....	288
fuse-cxa-atexit.....	63
fuse-cxa-get-exception-ptr.....	63
fuse-ld=bfd.....	277
fuse-ld=gold.....	277
fuse-ld=lld.....	277
fuse-ld=mold.....	277
fuse-ld=wild.....	277
fuse-linker-plugin.....	235
fvar-tracking.....	192
fvar-tracking-assignments.....	192
fvar-tracking-assignments-toggle.....	312
fvar-tracking-uninit.....	192
fvariable-expansion-in-unroller.....	224
fvect-cost-model.....	223
fverbose-asm.....	290
fversion-loops-for-strides.....	244
fvisibility.....	294
fvisibility-inlines-hidden.....	63
fvisibility-ms-compat.....	64
fvpt.....	242
fvtable-verify.....	264
fvtv-counts.....	265
fvtv-debug.....	264
fweak.....	64
fweb.....	230
fwhole-program.....	230
fwide-exec-charset.....	272
fworking-directory.....	272
fwrapv.....	287
fwrapv-pointer.....	287
fwritable-relocated-rdata.....	381
fzero-call-used-regs.....	245
fzero-init-padding-bits=value.....	296
fzero-initialized-in-bss.....	205
fzero-link.....	84
F.....	382

G

g.....	189
gas-loc-support.....	193
gas-locview-support.....	193
gbtf.....	190
gcodeview.....	190
gcolumn-info.....	194
gctf.....	190
gdescribe-dies.....	192
gdwarf.....	189
gdwarf32.....	192
gdwarf64.....	192
gen-decls.....	85
gfull.....	383
ggdb.....	189
ggnu-pubnames.....	193
ginline-points.....	195
ginternal-reset-location-views.....	194
gno-as-loc-support.....	193
gno-as-locview-support.....	194
gno-codeview.....	190
gno-column-info.....	194
gno-describe-dies.....	192
gno-inline-points.....	195
gno-internal-reset-location-views.....	194
gno-prune-btf.....	190
gno-pubnames.....	193
gno-record-gcc-switches.....	193
gno-split-dwarf.....	192
gno-statement-frontiers.....	194
gno-strict-dwarf.....	193
gno-variable-location-views.....	194
gno-z.....	195
gprune-btf.....	190
gpubnames.....	192
grecord-gcc-switches.....	193
gsctf.....	500
gsplit-dwarf.....	192
gstatement-frontiers.....	194
gstrict-dwarf.....	193
gtoggle.....	312
gused.....	382
gvariable-location-views.....	194
gvariable-location-views=incompat5.....	194
gvms.....	190
gz.....	195
G.....	339, 408, 412, 427, 481, 507

H

headerpad_max_install_names.....	385
help.....	41
H.....	274

I

I-	283
idirafter	282
iframework	382
imacros	268
image_base	385
imultiarch	284
imultilib	284
include	268
include-barrier	283
include-directory	282
include-directory-after	282
include-prefix	283
include-with-prefix	283
include-with-prefix-after	283
include-with-prefix-before	283
init	385
install_name	385
iplugindir=	284
iprefix	283
iquote	282
isysroot	284
isystem	282
iwithprefix	283
iwithprefixbefore	283
I	282

K

keep_private_externs	385
----------------------	-----

L

l	277
language	36
library-directory	284
lobjc	277
L	284

M

m1	494
m10	447
m128bit-long-double	534
m16	548
m16-bit	376, 442
m1reg-	331
m2	494
m210	418
m2a	494
m2a-nofpu	494
m2a-single	494
m2a-single-only	494
m2e	494
m3	494
m31	489
m32	473, 506, 548
m32-bit	376

m32bit-doubles	485
m32r	411
m32r2	411
m32rx	411
m340	418
m3dnow	530
m3dnowa	530
m3e	494
m4	494
m4-100	495
m4-100-nofpu	495
m4-100-single	495
m4-100-single-only	495
m4-200	495
m4-200-nofpu	495
m4-200-single	495
m4-200-single-only	495
m4-300	495
m4-300-nofpu	495
m4-300-single	495
m4-300-single-only	495
m4-340	495
m4-400	495
m4-500	495
m4-nofpu	494
m4-single	494
m4-single-only	494
m40	447
m45	447
m4a	496
m4a-nofpu	495
m4a-single	495
m4a-single-only	495
m4al	496
m4byte-functions	418
m5200	415
m5206e	415
m528x	415
m5307	415
m5407	415
m64	444, 473, 489, 506, 548
m64bit-doubles	485
m68000	414
m68010	414
m68020	414
m68020-40	415
m68020-60	415
m68030	414
m68040	414
m68060	414
m68302	414
m68332	414
m68851	414
m68881	415
m8-bit	376
m80387	533
m8bit-idiv	546
m8byte-align	509

m96bit-long-double.....	534	mamx-fp8.....	532
mA6.....	333	mamx-int8.....	531
mA7.....	333	mamx-movrs.....	532
mabi... 317, 341, 406, 422, 441, 449, 450, 478, 541,		mamx-tf32.....	532
551		mamx-tile.....	531
mabi=call0.....	551	manchor.....	379
mabi=elfv1.....	478	mandroid.....	396
mabi=elfv2.....	478	manotate-tablejump.....	411
mabi=gnu.....	436	mapcs.....	341
mabi=ibmlongdouble.....	478	mapcs-frame.....	341
mabi=ieeelongdouble.....	478	mapp-regs.....	501, 510
mabi=mmixware.....	436	mapx-inline-asm-use-gpr32.....	547
mabi=windowed.....	551	mapxf.....	532
mabcalls.....	423	mARC600.....	333
mabm.....	530	mARC601.....	333
mabort-on-noreturn.....	354	mARC700.....	333
mabs=2008.....	425	march.. 320, 331, 342, 374, 375, 397, 400, 405, 412,	
mabs=legacy.....	425	420, 444, 451, 490, 513	
mabsdata.....	362	march-map.....	444
mac0.....	447	march=.....	377, 442
macc-4.....	395	marclinux.....	339
macc-8.....	395	marclinux_prof.....	339
maccumulate-args.....	364	marm.....	355
maccumulate-outgoing-args.....	499, 542	mas100-syntax.....	486
maddr-reg-reg-cost.....	407	masm-hex.....	438
maddress-mode=long.....	548	masm-len-notes.....	362
maddress-mode=short.....	548	masm-syntax-unified.....	357
madjust-lmul-cost.....	467	masm=diect.....	392, 541
mads.....	479	matomic.....	335
madx.....	530	matomic-libcalls.....	398
maes.....	529	matomic-model=model.....	497
maix-struct-return.....	478	matt-stubs.....	383
maix32.....	473	mauto-litpools.....	550
maix64.....	473	mauto-modify-reg.....	339
malign-300.....	397	mauto-pic.....	402
malign-data.....	466, 534	mautovec-preference.....	320
malign-double.....	534	mautovec-segment.....	467
malign-functions.....	441	mavoid-indexed-addresses.....	475
malign-int.....	416	mavx.....	527
malign-labels.....	394	mavx10.1.....	529
malign-loops.....	412	mavx10.2.....	529
malign-natural.....	474	mavx2.....	527
malign-power.....	474	mavx256-split-unaligned-load.....	546
malign-stringops.....	544	mavx256-split-unaligned-store.....	546
malloc-cc.....	393	mavx512bf16.....	528
mallow-string-insns.....	487	mavx512bitalg.....	528
mallregs.....	468	mavx512bmm.....	528
maltivec.....	470	mavx512bw.....	528
malu32.....	391	mavx512cd.....	528
malways-align.....	441	mavx512dq.....	528
malways-save-lp.....	443	mavx512f.....	527
mam33.....	437	mavx512fp16.....	528
mam33-2.....	437	mavx512ifma.....	528
mam34.....	437	mavx512vbmi.....	528
mamx-avx512.....	532	mavx512vbmi2.....	528
mamx-bf16.....	531	mavx512vl.....	528
mamx-complex.....	531	mavx512vnmi.....	528
mamx-fp16.....	531	mavx512vp2intersect.....	528

mavx512vpopcntdq	528	mcall-prologues	365
mavxifma	529	mcall-sysv	477
mavxneconvert	529	mcall-sysv-eabi	477
mavxvnni	529	mcall-sysv-noeabi	477
mavxvnniint16	529	mcallee-super-interworking	355
mavxvnniint8	529	mcaller-copies	398
max-vect-align	331	mcaller-super-interworking	355
mb	496	mcallgraph-data	418
mbackchain	488	mcase-vector-pcrel	339
mbarrel-shift-enabled	404	mcbcond	505
mbarrel-shifter	333	mcbranch-force-delay-slot	499
mbase-addresses	436	mcc-init	376
mbe32	342	mccrt	379
mbe8	342	mcet-switch	541
mbest-lib-options	376	mcfv4e	415
mbig	476	mcheck-zero-division	407, 429
mbig-endian	317, 342, 374, 377, 391, 401, 418, 419, 441, 466, 476	mcix	388
mbig-endian-data	486	mcl	537
mbig-switch	509	mcldemote	531
mbigtable	496	mclear-hwcap	500
mbionic	396	mclflushopt	529
mbit-align	475	mclwb	529
mbit-word	475	mclzero	530
mbitfield	416	mcmem	336
mbitops	336, 496	mcmodel=	317, 408, 443, 446, 465, 470, 506, 548
mblock-compare-inline-limit	480	mcmodel=kernel	548
mblock-compare-inline-loop-limit	480	mcmodel=large	317, 446, 465, 470, 548
mblock-move-inline-limit	480	mcmodel=medany	465
mblock-ops-unaligned-vsx	484	mcmodel=medium	470, 548
mbmi	530	mcmodel=medlow	465
mbmi2	530	mcmodel=small	317, 446, 470, 548
mboard	445	mcmodel=tiny	317
mbranch-cost	329, 365, 407, 412, 433, 450	mcmov	441, 446
mbranch-cost=	379	mcmov	329
mbranch-cost=num	499	mcmpb	471
mbranch-index	338	mcmpccxadd	531
mbranch-likely	433	mcmse	357
mbranch-predict	436	mco-re	392
mbranch-protection	323, 358	mcode-density	335
mbrcc	339	mcode-density-frame	340
mbreak-code	407	mcode-readable	428
mbss-plt	471	mcode-region	440
mbswap	391	mcode-xonly	428
mbuild-constants	388	mcoherent-ldcw	398
mbwx	388	mcompact-branches=always	433
mc68000	414	mcompact-branches=never	433
mc68020	414	mcompact-branches=optimal	433
mcache	378	mcompat-align-parm	483
mcache-block-size	442	mcompress	396
mcall-aixdesc	477	mcond-exec	395
mcall-eabi	477	mcond-move	395
mcall-freebsd	477	mcond-move-float	407
mcall-linux	477	mcond-move-int	407
mcall-main	363	mconfig-fpu=	443
mcall-ms2sysv-xlogues	541	mconfig-mul=	443
mcall-netbsd	478	mconfig-register-ports=	443
mcall-openbsd	478	mconsole	380
		mconst-align	376

mconst16.....	550	MD.....	270
mconstant-cfstrings.....	383	mea.....	335
mconstant-gp.....	402	meabi.....	479
mconstpool.....	379	nearly-cbranchsi.....	340
mcarea.....	374	nearly-ldp-fusion.....	323
mcarea.....	374	nearly-ra.....	323
mcp.....	378	nearly-stop-bits.....	402
mcpu... 322, 333, 351, 375, 389, 392, 396, 413, 461,		meb.....	420, 438
468, 469, 485, 502, 511		medsp.....	378
mcpu=..... 372, 377, 419, 442		mel.....	420, 438
mcpu32.....	414	melf.....	436
mcrc.....	427	melrw.....	378
mcrc32.....	539	memb.....	479
mcrt.dll.....	380	membedded-data.....	428
mcrypto.....	472	menable-sysreg-checking.....	317
mcsr-check.....	466	menqcmd.....	531
mcsync-anomaly.....	372	mep.....	507
mctor-dtor.....	443	mepsilon.....	435
mcvt.....	362	mes0.....	468
mcx16.....	539	mesa.....	489
md.....	510	metrax100.....	375
md-float.....	510	metrax4.....	375
mdalign.....	496	meva.....	426
mdata-align.....	376	mexplicit-relocs..... 389, 408, 429, 465	
mdata-region.....	440	mexr.....	397
mdaz-ftz.....	536	mext-dsp.....	442
mdebug..... 412, 490, 511		mext-fpu-dp.....	442
mdebug-main=prefix.....	512	mext-fpu-fma.....	442
mdec-asm.....	447	mext-fpu-sp.....	442
mdirect-extern-access..... 408, 549		mext-perf.....	441
mdisable-callt.....	508	mext-perf2.....	441
mdisable-fpregs.....	398	mext-string.....	442
mdisable-indexing.....	398	mextern-sdata.....	428
mdispatch-scheduler.....	549	mextra-l32r-costs.....	551
mdiv..... 379, 415, 418, 450		mf16c.....	529
mdiv-rem.....	335	mfancy-math-387.....	533
mdiv=strategy.....	498	mfast-fp.....	374
mdiv32.....	410	mfast-indirect-calls.....	398
mdivide-breaks.....	429	mfaster-structs.....	502
mdivide-enabled.....	404	mfddiv.....	450
mdivide-traps.....	429	mfddivdu.....	378
mdivsi3_libfunc=name.....	499	mfdpic..... 357, 394, 500	
mdll.....	380	mfence-tso.....	450
mdlmzb.....	475	mfentry..... 493, 545	
mdmx.....	426	mfentry-name.....	545
mdouble..... 362, 393		mfentry-section.....	545
mdouble-float..... 378, 406, 425, 446		mfidoa.....	415
mdpfp.....	335	mfillzero.....	449
mdpfp-compact.....	335	mfix.....	388
mdpfp-fast.....	335	mfix-24k.....	431
mdpfp-lrsr.....	335	mfix-and-continue.....	383
mdsbt.....	375	mfix-at697f.....	505
mdsp..... 378, 426		mfix-cmse-cve-2021-35465.....	357
mdspr2.....	426	mfix-cortex-a53-835769.....	318
mdump-tune-features.....	537	mfix-cortex-a53-843419.....	318
mdwarf2-asm.....	402	mfix-cortex-a57-aes-1742098.....	356
mdword.....	393	mfix-cortex-a72-aes-1655431.....	356
mdynamic-no-pic..... 383, 476		mfix-cortex-m3-ldrd.....	356

mfix-gr712rc	506	mfsrra	500
mfix-r10000	431	mft32b	396
mfix-r4000	431	mfull-regs	441
mfix-r4400	431	mfull-toc	473
mfix-r5900	431	mfunction-return	493, 547
mfix-rm7000	431	mfunction-return-mem	493
mfix-sb1	432	mfunction-return-reg	493
mfix-ut699	506	mfuse-add	365
mfix-ut700	506	mfuse-move	365
mfix-vr4120	431	mfuse-move2	365
mfix-vr4130	432	mfused-madd	430, 475, 491
mfix4300	432	mfixsr	530
mfixed-cc	393	MF	269
mfixed-range	398, 402, 499	mg	510
mflat	501	mg-float	510
mflip-mips16	422	mg10	468
mflmap	371	mg13	468
mfloat-abi	342, 377, 441	mg14	468
mfloat-ieee	389	mgang-private-size	333
mfloat-vax	389	mgas	398
mfloat128	472	mgas-isr-prologues	362
mfloat128-hardware	473	mgather	547
mflush-func	412, 433	mgcc-abi	509
mflush-trap	412	mgeneral-regs-only	317, 342, 546
mfma	529	mgfni	530
mfma4	529	mgfs	509
mfmaf	505	mginv	427
mfmovd	496	mglibc	396
mforce-indirect-call	541	mgnu	510
mforce-l32	551	mgnu-as	401
mforce-no-pic	550	mgnu-asm	447
mfp-as-gp	441	mgnu-attribute	478
mfp-exceptions	434	mgnu-ld	399, 401
mfp-iarith	330	mgomp	445
mfp-in-toc	473	mgp32	424
mfp-mode	330	mgp64	424
mfp-regs	386	mgpopt	428
mfp-ret-in-387	533	mgpr-32	393
mfp-rounding-mode	387	mgpr-64	393
mfp-trap-mode	387	mgprel-ro	394
mfp16-format	353	MG	269
mfp32	424	mh	397
mfp64	424	mhalf-reg-file	329
mfpmath	238, 532	mhard-dfp	471, 488
mfpr-32	393	mhard-div	445
mfpr-64	393	mhard-float	393, 415, 419, 424, 446, 474, 488, 501, 509, 511, 533
mfprnd	471	mhard-mul	446
mfpu	336, 353, 406, 447, 501, 511	mhard-quad-float	501
mfpu=	378	mharden-sls	323, 547
mfpxx	424	mhardlit	418
mfraction-convert-truncate	363	mhigh-registers	379
mframe-header-opt	435	mhle	530
mframe-limit	391	mhotpatch	492
mfrecipe	410	mhp-ld	399
mfriz	483	mhreset	531
mfscs	499	mhtm	472, 490
mfsgsbase	529	mhw-abs	441
mfsmuld	505		

mhwmult	439	mips64r3	422
miamcu	548	mips64r5	422
micplb	374	mips64r6	422
mid-shared-library	373, 417	mirq-ctrl-saved	338
mieee	386, 496	misa	444
mieee-conformant	388	misa-spec	450
mieee-fp	533	misel	472
mieee-with-inexact	386	misize	338, 497
mieee128-constant	485	misr-secure	442
milp32	403	misr-vector-size	442
mimadd	430	missue-rate	412
mimpure-text	500	mistack	378
mincoming-stack-boundary	537	mjli-always	333
minindexed-loads	340	mjmp32	391
mindirect-branch	493, 546	mjmpext	391
mindirect-branch-call	493	mjsr	488
mindirect-branch-cs-prefix	547	mjump-tables-in-data-section	510
mindirect-branch-jump	493	mkernl	383
mindirect-branch-register	547	mk1	531
mindirect-branch-table	493	mknuthdiv	436
minline-all-stringops	544	ml	496
minline-asm-r15	444	mlam	549
minline-atomics	464	mlam-bh	410
minline-float-divide-max-throughput	402	mlamcas	410
minline-float-divide-min-latency	402	mlarge	439
minline-ic_invalidate	496	mlarge-data	389
minline-int-divide	402	mlarge-data-threshold	535
minline-int-divide-max-throughput	402	mlarge-text	389
minline-int-divide-min-latency	402	mlasx	407
minline-memops-threshold	392	mlate-ldp-fusion	323
minline-plt	374, 394	mld-seq-sa	410
minline-sqrt-max-throughput	402	mleaf-id-shared-library	373
minline-sqrt-min-latency	402	mlegacy-threads	411
minline-strcmp	464	mlibfuncs	435
minline-stringops-dynamically	544	mlibrary-pic	394
minline-strlen	464	mlinked-fp	394
minline-strncmp	464	mlinker-opt	399
minrt	440, 448	mlittle	476
minsert-sched-nops	477	mlittle-endian	317, 342, 375, 377, 391, 401, 418, 419, 441, 466, 476
minstrument-return	545	mlittle-endian-data	486
mint-register	487	mliw	437
mint16	447	mll64	335
mint32	397, 447	mllsc	425
mint8	362	mload-store-pairs	430
minterlink-compressed	422	mlocal-sdata	427
minterlink-mips16	422	mlock	338
mips1	421	mlong-calls	329, 339, 354, 373, 375, 394, 399, 430, 507
mips16	422	mlong-double	362
mips2	421	mlong-double-128	391, 488, 534
mips3	421	mlong-double-64	391, 488, 534
mips32	421	mlong-double-80	534
mips32r3	421	mlong-jump-table-offsets	418
mips32r5	421	mlong-jumps	508
mips32r6	422	mlong-load-store	399
mips3d	426	mlong32	427
mips4	421	mlong64	427
mips64	422		
mips64r2	422		

mlongcall.....	481	mmovcc.....	463
mlongcalls.....	551	mmovdir64b.....	531
mloongson-ext.....	427	mmovdiri.....	531
mloongson-ext2.....	427	mmove-max.....	538
mloongson-mmi.....	427	mmovrs.....	532
mloop.....	449, 509	mmp.....	378
mlow-precision-div.....	319	mmpy-option.....	336
mlow-precision-recip-sqrt.....	319	mms-bitfields.....	542
mlow-precision-sqrt.....	319	mmsa.....	427
mlow64k.....	373	mmt.....	426
mlp64.....	403	mmul.....	449, 467
mlpc-width.....	338	mmul-bug-workaround.....	375
mlra.....	447, 488, 500, 510	mmul.x.....	438
mlra-priority-compact.....	340	mmul32x16.....	335
mlra-priority-noncompact.....	340	mmul64.....	335
mlra-priority-none.....	340	mmuladd.....	393
mlsx.....	407	mmulhw.....	475
mlwp.....	530	mmult-bug.....	437
mlxc1-sxc1.....	435	mmultcost.....	341
mlzcnt.....	530	mmulti-cond-exec.....	395
mmacosx-version-min.....	383	mmulticore.....	374
mmad.....	430	mmultiple.....	474
mmadd4.....	435	mmultiple-stld.....	379
mmain.....	491	mmultiply-enabled.....	405
mmain-is-OS_task.....	362	mmusl.....	396
mmainkernel.....	444	mmvcle.....	490
mmalloc64.....	512	mmvme.....	479
mmmanual-endbr.....	541	mmwait.....	539
mmax.....	388	mmwaitx.....	530
mmax-constant-size.....	486	MM.....	269
mmax-inline-memcpy-size.....	408	mn.....	397
mmax-inline-shift=.....	440	mn-flash.....	371
mmax-stackframe.....	375	mnan=2008.....	425
mmax-vectorization.....	319, 465	mnan=legacy.....	425
mmay-round-for-trunc.....	330	mnarrow-gp-writes.....	323
mmcount-ra-address.....	435	mneeded.....	549
mmcu.....	359, 426, 448	mnested-cond-exec.....	395
mmcu=.....	438	mno-16-bit.....	442
MMD.....	270	mno-3dnow.....	530
mmedia.....	393	mno-3dnowa.....	530
mmedium-calls.....	339	mno-4byte-functions.....	418
mmemcpy.....	407, 419, 430	mno-80387.....	533
mmemcpy-strategy=strategy.....	544	mno-8bit-idiv.....	546
mmemory-latency.....	390	mno-8byte-align.....	509
mmemory-model.....	506	mno-abicalls.....	423
mmemset-strategy=strategy.....	544	mno-abm.....	530
mmfcrf.....	471	mno-abort-on-noreturn.....	354
mmicromips.....	426	mno-ac0.....	447
mmillicode.....	340	mno-accumulate-outgoing-args.....	499, 542
mmminimal-toc.....	473	mno-adjust-lmul-cost.....	467
mmips16e2.....	422	mno-adx.....	530
mma.....	484	mno-aes.....	529
mmmx.....	527	mno-align-double.....	534
mmodel.....	411	mno-align-functions.....	441
mmodel=large.....	411	mno-align-int.....	416
mmodel=medium.....	411	mno-align-labels.....	394
mmodel=small.....	411	mno-align-loops.....	412
mmovbe.....	539	mno-align-stringops.....	544

mno-allow-string-insns	487	mno-avxifma	529
mno-allregs	468	mno-avxneconvert	529
mno-altivec	470	mno-avxvnni	529
mno-alu32	391	mno-avxvnniint16	529
mno-always-align	441	mno-avxvnniint8	529
mno-always-save-lp	443	mno-backchain	488
mno-am33	437	mno-barrel-shift-enabled	404
mno-am33-2	437	mno-barrel-shifter	333
mno-am34	437	mno-base-addresses	436
mno-amx-avx512	532	mno-big-switch	509
mno-amx-bf16	531	mno-bit-align	475
mno-amx-complex	531	mno-bit-word	475
mno-amx-fp16	531	mno-bitfield	416
mno-amx-fp8	532	mno-bitops	336
mno-amx-int8	531	mno-block-ops-unaligned-vsx	484
mno-amx-movrs	532	mno-bmi	530
mno-amx-tf32	532	mno-bmi2	530
mno-amx-tile	531	mno-branch-index	338
mno-anchor	379	mno-branch-likely	433
mno-android	396	mno-branch-predict	436
mno-annotate-tablejump	411	mno-brcc	339
mno-apcs-frame	341	mno-bswap	391
mno-app-regs	501, 510	mno-bwx	388
mno-apx-inline-asm-use-gpr32	547	mno-cache	378
mno-apxf	532	mno-call-main	363
mno-arclinux	339	mno-call-ms2sysv-xlogues	541
mno-arclinux_prof	339	mno-callee-super-interworking	355
mno-as100-syntax	486	mno-caller-copies	398
mno-asm-hex	438	mno-caller-super-interworking	355
mno-asm-syntax-unified	357	mno-callgraph-data	418
mno-atomic	335	mno-case-vector-pcrel	339
mno-atomic-libcalls	398	mno-cbcond	505
mno-att-stubs	383	mno-cc-init	376
mno-auto-litpools	550	mno-ccrt	379
mno-auto-modify-reg	339	mno-cet-switch	541
mno-autovec-segment	467	mno-check-zero-division	407, 429
mno-avoid-indexed-addresses	475	mno-cix	388
mno-avx	527	mno-cld	537
mno-avx10.1	529	mno-cldemote	531
mno-avx10.2	529	mno-clear-hwcap	500
mno-avx2	527	mno-clflushopt	529
mno-avx256-split-unaligned-load	546	mno-clwb	529
mno-avx256-split-unaligned-store	546	mno-clzero	530
mno-avx512bf16	528	mno-cmem	336
mno-avx512bitalg	528	mno-cmov	441
mno-avx512bmm	528	mno-cmove	329
mno-avx512bw	528	mno-cmpb	471
mno-avx512cd	528	mno-cmpccxadd	531
mno-avx512dq	528	mno-co-re	392
mno-avx512f	527	mno-code-density	335
mno-avx512fp16	528	mno-code-density-frame	340
mno-avx512ifma	528	mno-coherent-ldcw	398
mno-avx512vbmi	528	mno-compatible-align-parm	483
mno-avx512vbmi2	528	mno-compress	396
mno-avx512vl	528	mno-cond-exec	339, 395
mno-avx512vnni	528	mno-cond-move	395
mno-avx512vp2intersect	528	mno-cond-move-float	407
mno-avx512vpopcntdq	528	mno-cond-move-int	407

mno-const-align.....	376	mno-explicit-relocs.....	389, 408, 429, 465
mno-const16.....	550	mno-exr.....	397
mno-constant-cfstrings.....	383	mno-ext-dsp.....	442
mno-constpool.....	379	mno-ext-fpu-dp.....	442
mno-corea.....	374	mno-ext-fpu-fma.....	442
mno-coreb.....	374	mno-ext-fpu-sp.....	442
mno-cp.....	378	mno-ext-perf.....	441
mno-crc.....	427	mno-ext-perf2.....	441
mno-crc32.....	539	mno-ext-string.....	442
mno-crt0.....	437, 438	mno-extern-sdata.....	428
mno-crypto.....	472	mno-f16c.....	529
mno-csr-check.....	466	mno-fancy-math-387.....	533
mno-csync-anomaly.....	372	mno-fast-fp.....	374
mno-ctor-dtor.....	443	mno-fast-indirect-calls.....	398
mno-cx16.....	539	mno-faster-structs.....	502
mno-data-align.....	376	mno-fdiv.....	450
mno-data-in-code.....	428	mno-fdivdu.....	378
mno-daz-ftz.....	536	mno-fdpic.....	357, 394
mno-debug.....	490	mno-fence-tso.....	450
mno-default.....	537	mno-fentry.....	493, 545
mno-direct-extern-access.....	549, 702	mno-fillzero.....	449
mno-disable-callt.....	508	mno-fix.....	388
mno-disable-fpregs.....	398	mno-fix-24k.....	431
mno-disable-indexing.....	398	mno-fix-and-continue.....	383
mno-div.....	379, 415, 418, 450	mno-fix-cmse-cve-2021-35465.....	357
mno-div-rem.....	335	mno-fix-cortex-a53-835769.....	318
mno-div32.....	410	mno-fix-cortex-a53-843419.....	318
mno-divide-enabled.....	404	mno-fix-cortex-a57-aes-1742098.....	356
mno-dlmzb.....	475	mno-fix-cortex-a72-aes-1655431.....	356
mno-double.....	393	mno-fix-cortex-m3-ldrd.....	356
mno-double-float.....	378, 446	mno-fix-r10000.....	431
mno-dpfp.....	335	mno-fix-r4000.....	431
mno-dpfp-compact.....	335	mno-fix-r4400.....	431
mno-dpfp-fast.....	335	mno-fix-r5900.....	431
mno-dpfp-lrsr.....	335	mno-fix-rm7000.....	431
mno-dsbt.....	375	mno-fix-sb1.....	432
mno-dsp.....	378, 426	mno-fix-vr4120.....	431
mno-dspr2.....	426	mno-fix-vr4130.....	432
mno-dwarf2-asm.....	402	mno-fix4300.....	432
mno-dword.....	393	mno-flat.....	501
mno-dynamic-no-pic.....	383	mno-flip-mips16.....	422
mno-ea.....	335	mno-float.....	424
mno-eabi.....	479	mno-float128.....	472
mno-early-cbranchsi.....	340	mno-float128-hardware.....	473
mno-early-ldp-fusion.....	323	mno-flush-func.....	412, 433
mno-early-stop-bits.....	402	mno-flush-trap.....	412
mno-edsp.....	378	mno-fma.....	529
mno-eflags.....	395	mno-fma4.....	529
mno-el.....	420	mno-fmaf.....	505
mno-elrw.....	378	mno-force-indirect-call.....	541
mno-embedded-data.....	428	mno-force-l32.....	551
mno-enqcmd.....	531	mno-force-no-pic.....	550
mno-ep.....	507	mno-fp-as-gp.....	441
mno-epsilon.....	435	mno-fp-exceptions.....	434
mno-es0.....	468	mno-fp-iarith.....	330
mno-etrax100.....	375	mno-fp-in-toc.....	473
mno-etrax4.....	375	mno-fp-regs.....	386
mno-eva.....	426	mno-fp-ret-in-387.....	533

mno-fprnd.....	471	mno-interlink-compressed.....	422
mno-fpu.....	501, 511	mno-interlink-mips16.....	422
mno-frame-header-opt.....	435	mno-interrupts.....	363
mno-frecipe.....	410	mno-isel.....	472
mno-friz.....	483	mno-isize.....	338
mno-fsca.....	499	mno-istack.....	378
mno-fsgsbase.....	529	mno-jmp32.....	391
mno-fsmuld.....	505	mno-jmpext.....	391
mno-fsrra.....	500	mno-jsr.....	488
mno-ft32b.....	396	mno-jump-tables-in-data-section.....	510
mno-full-toc.....	473	mno-kernel.....	383
mno-fused-madd.....	430, 475, 491	mno-kl.....	531
mno-fxsr.....	530	mno-knuthdiv.....	436
mno-gas.....	398	mno-lam-bh.....	410
mno-gather.....	547	mno-lamcas.....	410
mno-gfni.....	530	mno-lasx.....	407
mno-ginv.....	427	mno-late-ldp-fusion.....	323
mno-gnu-as.....	401	mno-ld-seq-sa.....	410
mno-gnu-attribute.....	478	mno-leaf-id-shared-library.....	373
mno-gnu-ld.....	401	mno-libfuncs.....	435
mno-gomp.....	445	mno-library-pic.....	394
mno-gpopt.....	428	mno-linked-fp.....	394
mno-gprel-ro.....	394	mno-liw.....	437
mno-h.....	397	mno-ll64.....	335
mno-half-reg-file.....	329	mno-llsc.....	425
mno-hard-dfp.....	471, 488	mno-load-store-pairs.....	430
mno-hardlit.....	418	mno-local-sdata.....	427
mno-high-registers.....	379	mno-lock.....	338
mno-hle.....	530	mno-long-calls.....	329, 339, 354, 373, 375, 394, 399, 430, 507
mno-hreset.....	531	mno-long-jumps.....	508
mno-htm.....	472, 490	mno-long-load-store.....	399
mno-hw-abs.....	441	mno-longcall.....	481
mno-iamcu.....	548	mno-longcalls.....	551
mno-icplb.....	374	mno-loongson-ext.....	427
mno-id-shared-library.....	373, 417	mno-loongson-ext2.....	427
mno-ieee.....	496	mno-loongson-mmi.....	427
mno-ieee-fp.....	533	mno-loop.....	449
mno-ieee128-constant.....	485	mno-low-precision-div.....	319
mno-imadd.....	430	mno-low-precision-recip-sqrt.....	319
mno-impure-text.....	500	mno-low-precision-sqrt.....	319
mno-indexed-loads.....	340	mno-low64k.....	373
mno-indirect-branch-cs-prefix.....	547	mno-lra.....	447, 488, 500, 510
mno-indirect-branch-register.....	547	mno-lsim.....	393, 418
mno-indirect-branch-table.....	493	mno-lsx.....	407
mno-inline-all-stringops.....	544	mno-lwp.....	530
mno-inline-asm-r15.....	444	mno-lxc1-sxc1.....	435
mno-inline-atomics.....	464	mno-lzcnt.....	530
mno-inline-float-divide.....	402	mno-mad.....	430
mno-inline-ic_invalidate.....	496	mno-madd4.....	435
mno-inline-int-divide.....	402	mno-main.....	491
mno-inline-plt.....	374, 394	mno-manual-endbr.....	541
mno-inline-sqrt.....	402	mno-max.....	388
mno-inline-strcmp.....	464	mno-max-vectorization.....	319, 465
mno-inline-stringops-dynamically.....	544	mno-may-round-for-trunc.....	330
mno-inline-strlen.....	464	mno-mcount-ra-address.....	435
mno-inline-strncmp.....	464	mno-mcu.....	426
mno-int16.....	447	mno-mdmx.....	426
mno-int32.....	447		

mno-media.....	393	mno-pass-mrelax-to-as.....	409
mno-medium-calls.....	339	mno-pc-relative-literal-loads.....	322
mno-memcpy.....	407, 419, 430	mno-pclmul.....	529
mno-mfcrf.....	471	mno-pconfig.....	529
mno-millicode.....	340	mno-pcrel.....	484
mno-minimal-toc.....	473	mno-pddebug.....	376
mno-mips16.....	422	mno-pic.....	401
mno-mips16e2.....	422	mno-pic-data-is-text-relative... 354, 420, 492	
mno-mips3d.....	426	mno-pid.....	487
mno-mjli-always.....	333	mno-pku.....	530
mno-mma.....	484	mno-plt.....	424
mno-mmcmips.....	426	mno-pltseq.....	481
mno-mmx.....	527	mno-pmem-wrap-around.....	363
Mno-modules.....	269	mno-pointers-to-nested-functions.....	483
mno-movbe.....	539	mno-poke-function-name.....	355
mno-movcc.....	463	mno-popc.....	505
mno-movdir64b.....	531	mno-popcnt.....	530
mno-movdiri.....	531	mno-popcntb.....	471
mno-movrs.....	532	mno-popcntd.....	471
mno-mp.....	378	mno-portable-runtime.....	400
mno-ms-bitfields.....	542	mno-post-inc.....	331
mno-msa.....	427	mno-postmodify.....	331
mno-mt.....	426	mno-power8-fusion.....	472
mno-mul.....	449	mno-powerpc-gfxopt.....	470
mno-mul-bug-workaround.....	375	mno-powerpc-gpopt.....	470
mno-mul.x.....	438	mno-powerpc64.....	470
mno-mul32x16.....	335	mno-prefer-avx128.....	538
mno-mul64.....	335	mno-prefer-short-insn-regs.....	329
mno-muladd.....	393	mno-prefetchi.....	531
mno-mulhw.....	475	mno-prefixed.....	484
mno-mult-bug.....	437	mno-preserve-args.....	494
mno-multi-cond-exec.....	395	mno-pretend-cmove.....	500
mno-multicore.....	374	mno-prfchw.....	530
mno-multiple.....	474	mno-privileged.....	484
mno-multiple-stld.....	379	mno-profile-kernel.....	470
mno-multiply-enabled.....	405	mno-prolog-function.....	507
mno-mvcle.....	490	mno-prologue-epilogue.....	376
mno-mwait.....	539	mno-prototype.....	479
mno-mwaitx.....	530	mno-ptwrite.....	529
mno-n.....	397	mno-pure-code.....	357
mno-needed.....	549	mno-push-args.....	542
mno-nested-cond-exec.....	395	mno-pushpop.....	379
mno-nodiv.....	396	mno-qmath.....	510
mno-nop-fun-dllimport.....	380	mno-quad-memory.....	472
mno-nop-mcount.....	493, 545	mno-quad-memory-atomic.....	472
mno-nopm.....	396	mno-quickcall.....	397
mno-noreturn-no-callee-saved-registers... 539		mno-raoint.....	531
mno-norm.....	335	mno-rdpid.....	530
mno-odd-spreg.....	425	mno-rdrnd.....	529
mno-omit-leaf-frame-pointer.. 318, 372, 466, 544		mno-rdseed.....	530
mno-optimize.....	444	mno-readonly-in-sdata.....	480
mno-optimize-membar.....	396	mno-recip.....	482, 539
mno-ordered.....	400	mno-recip-precision.....	482
mno-outline-atomics.....	319	mno-record-mcount.....	493, 545
mno-pack.....	395	mno-record-return.....	545
mno-packed-stack.....	489	mno-red-zone.....	548
mno-paired-single.....	426	mno-register-names.....	401
mno-partial-vector-fp-math.....	538	mno-regnames.....	481

mno-relax	409, 439, 443, 449, 466, 468, 508	mno-short-calls	330
mno-relax-cmpxchg-loop	546	mno-shorten-memrefs	464
mno-relax-hint	443	mno-shstk	539
mno-relax-immediates	418	mno-side-effects	376
mno-relax-pic-calls	434	mno-sign-extend-enabled	405
mno-relocatable	476	mno-sim	380, 396, 439, 467, 486
mno-relocatable-lib	476	mno-simd	335
mno-renesas	496	mno-single-exit	436
mno-restrict-it	357	mno-single-pic-base	354
mno-ret-in-naked-func	443	mno-skip-rax-setup	546
mno-return-pointer-on-d0	437	mno-slow-bytes	418
mno-rf16	338	mno-slow-flash-data	357
mno-riscv-attribute	466	mno-sm3	531
mno-rop-protect	484	mno-sm4	531
mno-round-nearest	329	mno-small-divides	419
mno-rtd	416, 535	mno-small-exec	489
mno-rtm	530	mno-small-model	393
mno-s	397	mno-small-sld	510
mno-s2600	397	mno-small16	330
mno-safe-bwa	388	mno-smart	379
mno-safe-partial	388	mno-smartmips	426
mno-sahf	539	mno-smov	391
mno-save-mduc-in-interrupts	468	mno-soft-cmpsf	329
mno-save-restore	463	mno-soft-float	386, 400
mno-save-toc-indirect	483	mno-soft-mult	400
mno-scalar-strict-align	464	mno-soft-stack	444
mno-scatter	547	mno-space-regs	399
mno-scc	395	mno-specld-anomaly	372
mno-sched-ar-data-spec	403	mno-spfp	335
mno-sched-ar-in-data-spec	403	mno-spfp-compact	335
mno-sched-br-data-spec	403	mno-spfp-fast	335
mno-sched-br-in-data-spec	403	mno-splat-float-constant	485
mno-sched-control-spec	403	mno-splat-word-constant	485
mno-sched-count-spec-in-critical-path	404	mno-split	447
mno-sched-fp-mem-deps-zero-cost	404	mno-split-addresses	429
mno-sched-in-control-spec	403	mno-split-lohi	331
mno-sched-max-memory-insns-hard-limit	404	mno-split-patch-nops	471
mno-sched-prolog	341	mno-split-vecmove-early	331
mno-sched-prolog-epilog	443	mno-sse	527
mno-sched-spec-control-ldc	404	mno-sse2	527
mno-sched-spec-ldc	404	mno-sse2avx	545
mno-sched-stop-bits-after-every-cycle	404	mno-sse3	527
mno-scq	410	mno-sse4	527
mno-sdata	339, 401, 480	mno-sse4.1	527
mno-sdiv	391	mno-sse4.2	527
mno-sdram	374	mno-sse4a	527
mno-security	378	mno-ssse3	527
mno-sel-sched-dont-check-control-spec	404	mno-stack-align	376
mno-sep-data	373, 417	mno-stack-arg-probe	536
mno-serialize	531	mno-stack-bias	507
mno-serialize-volatile	550	mno-stack-check-l1	373
mno-set-program-start	436	mno-stack-guard	491
mno-setlb	438	mno-stack-protector-guard-record	492
mno-sgx	530	mno-stack-size	379, 491
mno-sha	529	mno-stackrealign	536
mno-sha512	531	mno-std-struct-return	502
mno-shared	423	mno-strict-align	318, 407, 417, 430, 464, 476,
mno-short	416		510, 551

mno-stv	538	mno-vis4b	505
mno-subxc	505	mno-vliw-branch	395
mno-sum-in-toc	473	mno-volatile-asm-stop	401
mno-swap	335	mno-volatile-cache	339
mno-swape	338	mno-vpclmulqdq	530
mno-sx	397	mno-vrsave	471
mno-sym32	427	mno-vsx	472
mno-symbol-stubs	384	mno-vx	490
mno-synci	434	mno-vzeroupper	538
mno-target-align	550	mno-waitpkg	530
mno-tbm	530	mno-warn-altivec-long	485
mno-text-section-literals	550	mno-warn-devices-csv	441
mno-thumb-interwork	341	mno-warn-dynamicstack	491
mno-tiny-printf	440	mno-warn-mcu	439
mno-tls-direct-seg-refs	545	mno-warn-multiple-fast-interrupts	487
mno-tls-kernel	390	mno-wbnoinvd	530
mno-tls-markers	482	mno-wide-bitfields	418
mno-tomcat-stats	396	mno-widekl	531
mno-toplevel-symbols	436	mno-win32	381
mno-tpcs-frame	355	mno-windows	381
mno-tpcs-leaf-frame	355	mno-word-relocations	356
mno-tpf-trace	491	mno-xbpf	392
mno-tpf-trace-skip	491	mno-xgot	417, 424
mno-track-speculation	319	mno-xl-barrel-shift	419
mno-trap-unaligned-atomic	377	mno-xl-compatible	474
mno-trap-using-break8	376	mno-xl-float-convert	419
mno-trust	378	mno-xl-float-sqrt	419
mno-tsxldtrk	531	mno-xl-gp-opt	419
mno-uintr	531	mno-xl-mode-bootstrap	420
mno-unaligned-access	356, 430, 443	mno-xl-mode-executable	420
mno-unaligned-atomic-may-use-library	377	mno-xl-mode-novectors	420
mno-unaligned-doubles	502	mno-xl-mode-xmdstub	420
mno-unaligned-symbols	494	mno-xl-multiply-high	419
mno-unicode	381	mno-xl-pattern-compare	419
mno-uniform-simt	445	mno-xl-prefetch	420
mno-uninit-const-in-rodata	428	mno-xl-soft-div	419
mno-unroll-only-small-loops	549	mno-xl-soft-mul	419
mno-update	475	mno-xop	530
mno-use-lower-region-prefix	440	mno-xpa	427
mno-user-enabled	405	mno-xsave	530
mno-user-mode	502	mno-xsavec	530
mno-usermode	498	mno-xsaveopt	530
mno-usermsr	532	mno-xsaves	530
mno-v3-atomics	391	mno-xtls	418
mno-v3push	442	mno-xy	338
mno-v8plus	504	mno-zdcbranch	499
mno-vaes	530	mno-zero-extend	436
mno-vdsp	378	mno-zvector	490
mno-vect-double	331	mnobitfield	416
mno-vect8-ret-in-mem	535	mnodiv	396
mno-vector-strict-align	465	mnofdpic	500
mno-vh	442	mnomacsave	496
mno-virt	426	mnop-fun-dllimport	380
mno-vis	504	mnop-mcount	493, 545
mno-vis2	504	mnopm	396
mno-vis3	504	mnopops	329
mno-vis3b	504	mnoreturn-no-callee-saved-registers	539
mno-vis4	505	mnorm	335

mnortd	416	mpretend-cmove	500
mnoshort	416	mprfchw	530
modd-spreg	425	mprioritize-restricted-insns	476
momit-leaf-frame-pointer	318, 372, 466, 544	mprivileged	484
mone-byte-bool	383	mprofile-kernel	470
moptimize	444	mprolog-function	507
moptimize-membar	396	mprologue-epilogue	376
mordered	400	mprototype	479
moutline-atomics	319	mptr32	506
moverride	322	mptr64	506
moverride-best-lib-options	376	mptwrite	529
mpa-risc-1-0	398	mptx	444
mpa-risc-1-1	398	mpure-code	357
mpa-risc-2-0	398	mpush-args	542
mpack	395	mpushpop	379
mpacked-stack	489	MP	269
mpaired-single	426	mqmath	510
mpartial-vector-fp-math	538	mquad-memory	472
mpass-mrelax-to-as	409	mquad-memory-atomic	472
mpc-relative-literal-loads	322	mquickcall	397
mpc32	535	MQ	270
mpc64	535	mr10k-cache-barrier	432
mpc80	535	mraoint	531
mpclmul	529	mrdrnd	530
mpconfig	529	mrdrnd	529
mpcrel	416, 484	mrdrnd	530
mpdebug	376	mreadonly-in-sdata	480
mpe	474	mrecip	409, 482, 539
mpe-aligned-commons	381	mrecip-precision	482
mpic-data-is-text-relative	354, 420, 492	mrecip=	539
mpic-register	354	mrecip=opt	409, 482
mpid	487	mrecord-mcount	493, 545
mpku	530	mrecord-return	545
mplt	424	mred-zone	548
mpltseq	481	mreduced-regs	441
mpmem-wrap-around	363	mregister-names	401
mpointer-size=size	512	mregnames	481
mpointers-to-nested-functions	483	mregparm	535
mpoke-function-name	355	mrelax	363, 397, 409, 437, 439, 443, 466, 468, 486, 496, 508
mpopc	505	mrelax-cmpxchg-loop	546
mpopcnt	530	mrelax-hint	443
mpopcntb	471	mrelax-immediates	418
mpopcntd	471	mrelax-pic-calls	434
mportable-runtime	400	mrelocatable	476
mpost-inc	331	mrelocatable-lib	476
mpost-modify	331	mrenesas	496
mpower8-fusion	472	mrestrict-it	357
mpowerpc-gfxopt	470	mret-in-naked-func	443
mpowerpc-gpopt	470	mreturn-pointer-on-d0	437
mpowerpc64	470	mrfl6	338
mprefer-avx128	538	mrgf-banked-regs	338
mprefer-short-insn-regs	329	mrh850-abi	509
mprefer-vector-width	538	mriscv-attribute	466
mprefergot	498	mrl78	468
mpreferred-stack-boundary	463, 536	mrnw	371
mprefetchi	531	mrodata-in-ram	363
mprefixed	484	mrop-protect	484
mpreserve-args	494		

mrord	446	msetlb	438
mrord	446	msect	446
mround-nearest	329	msfimm	446
mrttd	416, 535, 688	msgx	530
mrtm	530	msha	529
mrtpl	512	msha512	531
mrvv-max-lmul	467	mshared	411, 423
mrvv-vector-bits	467	mshared-library-id	373, 417
ms	397	mshftimm	446
ms2600	397	mshort	416
msafe-bwa	388	mshort-calls	330, 371
msafe-partial	388	mshorten-memrefs	464
msahf	539	mshstk	539
msave-acc-in-interrupts	487	mside-effects	376
msave-mduc-in-interrupts	468	msign-extend-enabled	405
msave-restore	463	msign-return-address	322
msave-toc-indirect	483	msilicon-errata	440
mscalar-strict-align	464	msilicon-errata-warn	440
mscatter	547	msim	372, 375, 380, 396, 439, 467, 479, 486, 511, 549
mscc	395	msimd	335, 406
msched-ar-data-spec	403	msingle-exit	436
msched-ar-in-data-spec	403	msingle-float	406, 424
msched-br-data-spec	403	msingle-pic-base	354, 476
msched-br-in-data-spec	403	msio	400
msched-control-spec	403	msize-level	340
msched-costly-dep	476	mskip-bug	371
msched-count-spec-in-critical-path	404	mskip-rax-setup	546
msched-fp-mem-deps-zero-cost	404	mslow-bytes	418
msched-in-control-spec	403	mslow-flash-data	357
msched-max-memory-insns	404	mslowbyte	397
msched-max-memory-insns-hard-limit	404	msm3	531
msched-prolog	341, 380	msm4	531
msched-prolog-epilog	443	msmall	439
msched-spec-control-ldc	404	msmall-data	389
msched-spec-ldc	404	msmall-data-limit	463, 486
msched-stop-bits-after-every-cycle	404	msmall-divides	419
mschedule	400	msmall-exec	489
mscq	410	msmall-model	393
msda	508	msmall-sld	510
msdata	339, 401, 411, 480	msmall-text	389
msdata=all	375	msmall16	330
msdata=data	480	msmart	379
msdata=default	375, 480	msmartmips	426
msdata=eabi	479	msmov	391
msdata=none	375, 411, 480	msmp	512
msdata=sdata	411	msoft-cmpsf	329
msdata=sysv	480	msoft-div	445
msdata=use	412	msoft-float	335, 386, 393, 400, 406, 415, 419, 424, 446, 447, 474, 488, 501, 509, 511, 533
msdiv	391	msoft-mul	446
msdram	374	msoft-mult	400
msecure-plt	471	msoft-quad-float	501
msecurity	378	msoft-stack	444
msel-sched-dont-check-control-spec	404	msoft-stack-reserve-local	445
msep-data	373, 417	misp8	372
mserialize	531	mspace	507
mserialize-volatile	550	mspace-regs	399
mset-data-start	436		
mset-program-start	436		

mspecld-anomaly	372	msymbol-stubs	384
mspfp	335	msynci	434
mspfp-compact	335	mtarget-align	550
mspfp-fast	335	mtarget-linker	384
msplat-float-constant	485	mtas	498
msplat-word-constant	485	mtbm	530
msplit	447	mtda	507
msplit-addresses	429	mtext-section-literals	550
msplit-bit-shift	365	mtthreads	380, 411
msplit-ldst	365	mthumb	355
msplit-lohi	331	mthumb-interwork	341
msplit-patch-nops	471	mtiny-printf	440
msplit-vecmove-early	331	mtiny-stack	363
msram-ecc	333	mtls	394
msse	527	mtls-dialect	356, 410, 542
msse2	527	mtls-dialect=desc	318, 467
msse2avx	545	mtls-dialect=trad	467
msse3	527	mtls-dialect=traditional	318
msse4	527	mtls-direct-seg-refs	545
msse4.1	527	mtls-kernel	390
msse4.2	527	mtls-markers	482
msse4a	527	mtls-size	318, 390, 403
mseregparm	535	mTLS	394
mssse3	527	mtomcat-stats	396
mstack-align	376	mtoplevel-symbols	436
mstack-arg-probe	536	mtp	317, 356
mstack-bias	507	mtp-regno	336
mstack-check-l1	373	mtpcs-frame	355
mstack-guard	491	mtpcs-leaf-frame	355
mstack-increment	418	mtpf-trace	491
mstack-offset	329	mtpf-trace-skip	491
mstack-protector-guard	318, 357, 466, 484, 492, 546	mtraceback	478
mstack-protector-guard-offset	318, 357, 466, 484, 546	mtrack-speculation	319
mstack-protector-guard-record	492	mtrap-precision	387
mstack-protector-guard-reg	318, 466, 484, 546	mtrap-unaligned-atomic	377
mstack-protector-guard-symbol	546	mtrap-using-break8	376
mstack-size	379, 491	mtrust	378
mstackrealign	536	mtsxldtrk	531
mstd-struct-return	502	mtune	321, 331, 341, 350, 375, 390, 403, 405, 413, 421, 437, 462, 470, 491, 504, 511, 526
mstore-max	538	mtune-ctrl=feature-list	537
mstrict-align	318, 407, 417, 430, 464, 476, 551	MT	269
mstrict-X	365	muclibc	396
mstring-compare-inline-limit	480	muintr	531
mstringop-strategy	464	multcost=number	498
mstringop-strategy=alg	544	multilib-library-pic	394
mstructure-size-boundary	353	munaligned-access	356, 430, 443
mstv	538	munaligned-atomic-may-use-library	377
msubxc	505	munaligned-doubles	502
msum-in-toc	473	munaligned-symbols	494
msv-mode	511	municode	381
msve-vector-bits	324	muniform-simt	445
msvr4-struct-return	478	muninit-const-in-rodata	428
mswap	335	munix	510
mswape	338	munix-asm	447
msx	397	munordered-float	446
msym32	427	munroll-only-small-loops	549
		mupdate	475

muse-libstdc-wrappers	381	mwindows	381
muse-lower-region-prefix	440	mword-relocations	356
muse-nonzero-bits	365	mwsio	400
muser-enabled	405	mx32	548
muser-mode	502, 511	mxbpf	392
musermode	498	mxgot	417, 424
musermsr	532	mxl-barrel-shift	419
mv3-atomics	391	mxl-compat	474
mv3push	442	mxl-float-convert	419
mv850	508	mxl-float-sqrt	419
mv850e	508	mxl-gp-opt	419
mv850e1	508	mxl-mode-bootstrap	420
mv850e2	508	mxl-mode-executable	420
mv850e2v3	508	mxl-mode-novectors	420
mv850e2v4	508	mxl-mode-xmdstub	420
mv850e3v5	508	mxl-multiply-high	419
mv850es	508	mxl-pattern-compare	419
mv8plus	504	mxl-prefetch	420
mvaes	530	mxl-reorder	419
mvaxc-alignment	510	mxl-soft-div	419
mvdsp	378	mxl-soft-mul	419
mveclibabi	483, 540	mxnack	333
mvect-double	331	mxop	530
mvect8-ret-in-mem	535	mxpa	427
mvector-strict-align	465	mxsave	530
mvectorize-with-neon-double	358	mxsavec	530
mvectorize-with-neon-quad	358	mxsaveopt	530
mvh	442	mxsaves	530
mvirt	426	mxtls	418
mvis	504	mxy	338
mvis2	504	myellowknife	479
mvis3	504	mzarch	489
mvis3b	504	mzda	508
mvis4	505	mzdcbranch	499
mvis4b	505	mzero-extend	436
mvliw-branch	395	mzilsd-strict-align	465
mvms-return-codes	512	mzilsd-word-align	465
mvolatile-asm-stop	401	mzvector	490
mvolatile-cache	339	M	268
mvpclmulqdq	530		
mvr4130-align	434	N	
mvrsave	471	n	282
mvsx	472	no-canonical-prefixes	285
mvthreads	512	no-integrated-cpp	275
mvx	490	no-line-commands	273
mvxworks	479	no-pie	279
mvzeroupper	538	no-standard-includes	284
mwaitpkg	530	no-standard-libraries	278
mwarn-altivec-long	485	no-sysroot-suffix	286
mwarn-devices-csv	441	no-warnings	101
mwarn-dynamicstack	491	nocpp	431
mwarn-framesize	491	nodefaultexport	385
mwarn-mcu	439	nodefaultlibs	278
mwarn-multiple-fast-interrupts	487	nodefaulttrpaths	384
mwbnoinvd	530	nodevicelib	363
mwide-bitfields	418	nodevicespecs	364
mwidekl	531	nofpu	485
mwin32	381		

nolibc	278
nolibdld	401
non-static	512
nostartfiles	277
nostdinc	284
nostdinc++	65, 284
nostdlib	278
nostdlib++	278
N	282

O

o	37
optimize	197
oslib	447
output	37
O	197
O0	199
O1	197
O2	198
O3	199
ObjC	384
ObjC++	384
Ofast	200
Og	200
Os	199
Oz	200

P

p	245, 620
pagezero_size	385
param	316
pass-exit-codes	43
pedantic	3, 102, 575, 792, 1115
pedantic-errors	3, 102, 1115
pg	245, 620
pie	279
pipe	43
prefix	285
preprocess	36
print-autofdo-gcov-version	314
print-file-name	314
print-libgcc-file-name	315
print-missing-file-dependencies	269
print-multi-directory	314
print-multi-lib	314
print-multi-os-directory	315
print-multiarch	315
print-objc-runtime-info	86
print-prog-name	315
print-search-dirs	315
print-sysroot	315
print-sysroot-headers-suffix	315
printf	448
profile	245
pthread	268, 279
P	273

Q

Q	42, 312
Qn	290
Qy	290

R

r	279
rdynamic	279
read_only_relocs	385
remap	274

S

s	279
save-temps	310
save-temps=cwd	310
save-temps=obj	310
scanf	448
sectalign	385
sectcreate	385
seg_addr_table	385
seg1addr	385
segaddr	385
segprot	385
segs_read_only_addr	385
segs_read_write_addr	385
shared	279
shared-libgcc	279
sim	377
sim2	377
specs	310
static	279, 401
static-libasan	280
static-libgcc	279
static-libhwasan	280
static-liblsan	280
static-libstdc++	281
static-libtsan	280
static-libubsan	281
static-pie	279
std	3, 45, 797, 1113
stdlib	65
sub_library	385
sub_umbrella	385
symbolic	281
sysroot	285
S	36, 276

T

t	282
target-help	41
Tbss	282
Tdata	282
threads	401
time	310
tno-android-cc	397
tno-android-ld	397
trace-includes	274
traditional	273, 1103
traditional-cpp	273
trigraphs	273
Ttext	282
twolevel_namespace	385
twolevel_namespace_hints	385
T	281

U

u	282
umbrella	385
undef	268
undefine-macro	267
undefined	385
unexported_symbols_list	385
user-dependencies	269
U	267

V

v	41
verbose	41
version	43

W

w	101
W	105, 161, 163, 1104
Wa	275
Wabbreviated-auto-in-template-arg	65
Wabi	106
Wabi-tag	65
Wabsolute-value	148
Waddr-space-convert	364
Waddress	156
Waddress-of-packed-member	157
Waggregate-return	158
Waggressive-loop-optimizations	158
Waligned-new	79
Wall	104, 1106
Walloc-size	135
Walloc-size-larger-than=	136
Walloc-zero	135
Walloca	136
Walloca-larger-than=	136
Wanalyzer-allocation-size	172
Wanalyzer-deref-before-check	172

Wanalyzer-div-by-zero	172
Wanalyzer-double-fclose	172
Wanalyzer-double-free	172
Wanalyzer-exposure-through-output-file	173
Wanalyzer-exposure-through-uninit-copy	173
Wanalyzer-fd-access-mode-mismatch	173
Wanalyzer-fd-double-close	173
Wanalyzer-fd-leak	173
Wanalyzer-fd-phase-mismatch	174
Wanalyzer-fd-type-mismatch	174
Wanalyzer-fd-use-after-close	174
Wanalyzer-fd-use-without-check	174
Wanalyzer-file-leak	174
Wanalyzer-free-of-non-heap	174
Wanalyzer-imprecise-fp-arithmetic	175
Wanalyzer-infinite-loop	175
Wanalyzer-infinite-recursion	176
Wanalyzer-jump-through-null	176
Wanalyzer-malloc-leak	176
Wanalyzer-mismatching-deallocation	176
Wanalyzer-mkostemp-redundant-flags	176
Wanalyzer-mktemp-missing-placeholder	177
Wanalyzer-mktemp-of-string-literal	177
Wanalyzer-null-argument	178
Wanalyzer-null-dereference	178
Wanalyzer-out-of-bounds	177
Wanalyzer-overlapping-buffers	177
Wanalyzer-possible-null-argument	178
Wanalyzer-possible-null-dereference	178
Wanalyzer-putenv-of-auto-var	178
Wanalyzer-shift-count-negative	179
Wanalyzer-shift-count-overflow	179
Wanalyzer-stale-setjmp-buffer	179
Wanalyzer-symbol-too-complex	171
Wanalyzer-tainted-allocation-size	179
Wanalyzer-tainted-array-index	180
Wanalyzer-tainted-assertion	179
Wanalyzer-tainted-divisor	180
Wanalyzer-tainted-offset	181
Wanalyzer-tainted-size	181
Wanalyzer-throw-of-unexpected-type	181
Wanalyzer-too-complex	171
Wanalyzer-undefined-behavior-ptrdiff	181
Wanalyzer-undefined-behavior-strtok	181
Wanalyzer-unsafe-call-within-signal-handler	182
Wanalyzer-use-after-free	182
Wanalyzer-use-of-pointer-in-stale-stack-frame	182
Wanalyzer-use-of-uninitialized-value	183
Wanalyzer-va-arg-type-mismatch	182
Wanalyzer-va-list-exhausted	182
Wanalyzer-va-list-leak	182
Wanalyzer-va-list-use-after-va-end	183
Wanalyzer-write-to-const	183
Wanalyzer-write-to-string-literal	183
Warith-conversion	137
Warray-bounds	137

Warray-compare	138	Wcoverage-too-many-paths	109
Warray-parameter	138	Wcpp	109
Wasm-len-notes	364	Wctad-maybe-unsupported	66
Wassign-intercept	85	Wctor-dtor-privacy	66
Wattribute-alias	139	Wdangling-else	153
Wattribute-warning	163	Wdangling-pointer	153
Wattributes	158	Wdangling-reference	66
Wauto-profile	136	Wdate-time	154
Wbad-function-cast	149	Wdeclaration-after-statement	145
Wbidi-chars	140	Wdeclaration-missing-parameter-type	160
Wbidi-chars=	140	Wdefaulted-function-deleted	80
Wbool-compare	140	Wdelete-incomplete	80
Wbool-operation	140	Wdelete-non-virtual-dtor	67
Wbuiltin-declaration-mismatch	158	Wdeprecated	163
Wbuiltin-macro-redefined	159	Wdeprecated-copy	67
Wc++-compat	150	Wdeprecated-copy-dtor	67
Wc++11-compat	150	Wdeprecated-declarations	163
Wc++11-extensions	150	Wdeprecated-enum-enum-conversion	68
Wc++14-compat	150	Wdeprecated-enum-float-conversion	68
Wc++14-extensions	150	Wdeprecated-literal-operator	68
Wc++17-compat	150	Wdeprecated-non-prototype	160
Wc++17-extensions	150	Wdeprecated-openmp	163
Wc++20-compat	150	Wdeprecated-variadic-comma-omission	68
Wc++20-extensions	150	Wdesignated-init	169
Wc++23-extensions	151	Wdisabled-optimization	168
Wc++26-compat	150	Wdiscarded-array-qualifiers	141
Wc++26-extensions	151	Wdiscarded-qualifiers	141
Wc11-c23-compat	149	Wdiv-by-zero	142
Wc11-c2x-compat	149	Wdouble-promotion	109
Wc23-c2y-compat	149	Wduplicate-decl-specifier	109
Wc90-c99-compat	149	Wduplicated-branches	141
Wc99-c11-compat	149	Wduplicated-cond	141
Wcalloc-transposed-args	136	weak_framework	385
Wcannot-profile	136	weak_reference_mismatches	385
Wcast-align	151	Weffc++	73
Wcast-align=strict	151	Welaborated-enum-base	68
Wcast-function-type	151	Wempty-body	154
Wcast-qual	151	Wendif-labels	149, 154
Wcast-user-defined	151	Wenum-compare	154
Wcatch-value	80	Wenum-conversion	154
Wchanges-meaning	108	Wenum-int-mismatch	154
Wchar-subscripts	108	Werror	101
Wclass-conversion	79	Werror=	101
Wclass-memaccess	70	Wexceptions	74
Wclobbered	152	Wexpansion-to-defined	148
Wco-re	392	Wexpose-global-module-tu-local	78
Wcomma-subscript	66	Wexternal-tu-local	78
Wcomment	148	Wextra	105, 161, 163
Wcomments	148	Wextra-semi	80
Wcompare-distinct-pointer-types	152	Wfatal-errors	101
Wcomplain-wrong-lang	152	Wflex-array-member-not-at-end	155
Wconditionally-supported	80	Wfloat-conversion	155
Wconstant-logical-operand	157	Wfloat-equal	143
Wconversion	152	Wformat	109, 110, 135, 610
Wconversion-null	82	Wformat-contains-nul	110
Wcoverage-invalid-line-number	109	Wformat-diag	110
Wcoverage-mismatch	108	Wformat-extra-args	110
Wcoverage-too-many-conditions	108	Wformat-nonliteral	112, 611

Wformat-overflow	110, 111	Wmemset-transposed-args	156
Wformat-security	112	Wmisleading-indentation	117
Wformat-signedness	112	Wmismatched-dealloc	119
Wformat-truncation	112	Wmismatched-new-delete	76
Wformat-y2k	113	Wmismatched-tags	76
Wformat-zero-length	112	Wmissing-attributes	118
Wformat=	109	Wmissing-braces	119
Wformat=1	110	Wmissing-declarations	160
Wformat=2	110	Wmissing-field-initializers	161
Wframe-address	141	Wmissing-format-attribute	135
Wframe-larger-than=	146	Wmissing-include-dirs	119
Wfree-labels	159	Wmissing-noreturn	135
Wfree-nonheap-object	146	Wmissing-parameter-name	160
Wglobal-module	81	Wmissing-parameter-type	160
Whardened	115	Wmissing-profile	119
whatsloaded	385	Wmissing-prototypes	160
Wheader-guard	159	Wmissing-requires	161
whyload	385	Wmissing-template-keyword	161
Wif-not-aligned	117	Wmissing-variable-declarations	160
Wignored-attributes	117	Wmisspelled-isr	364
Wignored-qualifiers	117	Wmultichar	162
Wimplicit	115	Wmultiple-inheritance	76
Wimplicit-fallthrough	115	Wmultiple-parameter-fwd-decl-lists	160
Wimplicit-fallthrough=	115	Wmultistatement-macros	120
Wimplicit-function-declaration	115	Wmusttail-local-addr	113
Wimplicit-int	114	Wnamespaces	77
Winaccessible-base	81	Wnarrowing	70
Wincompatible-pointer-types	141	Wnested-externs	165
Winfinite-recursion	114	Wno-abbreviated-auto-in-template-arg	65
Winherited-variadic-ctor	81	Wno-abi	106
Winit-list-lifetime	68	Wno-abi-tag	65
Winit-self	114	Wno-absolute-value	148
Winline	165, 719	Wno-addr-space-convert	364
Wint-conversion	141	Wno-address	156
Wint-in-bool-context	165	Wno-address-of-packed-member	157
Wint-to-pointer-cast	166	Wno-aggregate-return	158
Winterference-size	165	Wno-aggressive-loop-optimizations	158
Winvalid-constexpr	69	Wno-aligned-new	79
Winvalid-imported-macros	69	Wno-all	104
Winvalid-memory-model	128	Wno-alloc-size	135
Winvalid-offsetof	81	Wno-alloc-size-larger-than	136
Winvalid-pch	166	Wno-alloc-zero	135
Winvalid-utf8	166	Wno-alloca	136
Wjump-misses-init	155	Wno-alloca-larger-than	136, 137
Wkeyword-macro	159	Wno-analyzer-allocation-size	172
Wl	281	Wno-analyzer-deref-before-check	172
Wlarger-than-byte-size	146	Wno-analyzer-div-by-zero	172
Wlarger-than=	146	Wno-analyzer-double-fclose	172
Wleading-whitespace=	143	Wno-analyzer-double-free	172
Wliteral-suffix	69	Wno-analyzer-exposure- through-output-file	173
Wlogical-not-parentheses	158	Wno-analyzer-exposure- through-uninit-copy	173
Wlogical-op	157	Wno-analyzer-fd-access-mode-mismatch	173
Wlong-long	166	Wno-analyzer-fd-double-close	173
Wlto-type-mismatch	168	Wno-analyzer-fd-leak	173
Wmain	117	Wno-analyzer-fd-phase-mismatch	174
Wmaybe-musttail-local-addr	113	Wno-analyzer-fd-type-mismatch	174
Wmaybe-uninitialized	129		
Wmemset-elt-size	156		

Wno-analyzer-fd-use-after-close.....	174	Wno-bad-function-cast.....	149
Wno-analyzer-fd-use-without-check.....	174	Wno-bidi-chars.....	140
Wno-analyzer-file-leak.....	174	Wno-bool-compare.....	140
Wno-analyzer-free-of-non-heap.....	174	Wno-bool-operation.....	140
Wno-analyzer-imprecise-fp-arithmetic.....	175	Wno-builtin-declaration-mismatch.....	158
Wno-analyzer-infinite-loop.....	175	Wno-builtin-macro-redefined.....	159
Wno-analyzer-infinite-recursion.....	176	Wno-c++-compat.....	150
Wno-analyzer-jump-through-null.....	176	Wno-c++11-compat.....	150
Wno-analyzer-malloc-leak.....	176	Wno-c++11-extensions.....	150
Wno-analyzer-mismatching-deallocation....	176	Wno-c++14-compat.....	150
Wno-analyzer-mkostemp-redundant-flags....	176	Wno-c++14-extensions.....	150
Wno-analyzer-mktemp- missing-placeholder.....	177	Wno-c++17-compat.....	150
Wno-analyzer-mktemp-of-string-literal....	177	Wno-c++17-extensions.....	150
Wno-analyzer-null-argument.....	178	Wno-c++20-compat.....	150
Wno-analyzer-null-dereference.....	178	Wno-c++20-extensions.....	150
Wno-analyzer-out-of-bounds.....	177	Wno-c++23-extensions.....	151
Wno-analyzer-overlapping-buffers.....	177	Wno-c++26-compat.....	150
Wno-analyzer-possible-null-argument.....	178	Wno-c++26-extensions.....	151
Wno-analyzer-possible-null-dereference....	178	Wno-c11-c23-compat.....	149
Wno-analyzer-putenv-of-auto-var.....	178	Wno-c11-c2x-compat.....	149
Wno-analyzer-shift-count-negative.....	179	Wno-c23-c2y-compat.....	149
Wno-analyzer-shift-count-overflow.....	179	Wno-c90-c99-compat.....	149
Wno-analyzer-stale-setjmp-buffer.....	179	Wno-c99-c11-compat.....	149
Wno-analyzer-symbol-too-complex.....	171	Wno-calloc-transposed-args.....	136
Wno-analyzer-tainted-allocation-size.....	179	Wno-cannot-profile.....	136
Wno-analyzer-tainted-array-index.....	180	Wno-cast-align.....	151
Wno-analyzer-tainted-assertion.....	179	Wno-cast-align=strict.....	151
Wno-analyzer-tainted-divisor.....	180	Wno-cast-function-type.....	151
Wno-analyzer-tainted-offset.....	181	Wno-cast-qual.....	151
Wno-analyzer-tainted-size.....	181	Wno-cast-user-defined.....	151
Wno-analyzer-throw-of-unexpected-type....	181	Wno-catch-value.....	80
Wno-analyzer-too-complex.....	171	Wno-changes-meaning.....	108
Wno-analyzer-undefined- behavior-ptrdiff.....	181	Wno-char-subscripts.....	108
Wno-analyzer-undefined-behavior-strtok... 181		Wno-class-conversion.....	79
Wno-analyzer-unsafe-call-within- signal-handler.....	182	Wno-class-memaccess.....	70
Wno-analyzer-use-after-free.....	182	Wno-clobbered.....	152
Wno-analyzer-use-of-pointer-in- stale-stack-frame.....	182	Wno-comma-subscript.....	66
Wno-analyzer-use-of- uninitialized-value.....	183	Wno-comment.....	148
Wno-analyzer-va-arg-type-mismatch.....	182	Wno-comments.....	148
Wno-analyzer-va-list-exhausted.....	182	Wno-compare-distinct-pointer-types.....	152
Wno-analyzer-va-list-leak.....	182	Wno-complain-wrong-lang.....	152
Wno-analyzer-va-list-use-after-va-end....	183	Wno-conditionally-supported.....	80
Wno-analyzer-write-to-const.....	183	Wno-constant-logical-operand.....	157
Wno-analyzer-write-to-string-literal....	183	Wno-conversion.....	152
Wno-arith-conversion.....	137	Wno-conversion-null.....	82
Wno-array-bounds.....	137	Wno-coverage-invalid-line-number.....	109
Wno-array-compare.....	138	Wno-coverage-mismatch.....	108
Wno-array-parameter.....	138	Wno-coverage-too-many-conditions.....	108
Wno-assign-intercept.....	85	Wno-coverage-too-many-paths.....	109
Wno-attribute-alias.....	139	Wno-cpp.....	109
Wno-attribute-warning.....	163	Wno-ctad-maybe-unsupported.....	66
Wno-attributes.....	158	Wno-ctor-dtor-privacy.....	66
Wno-auto-profile.....	136	Wno-dangling-else.....	153
		Wno-dangling-pointer.....	153
		Wno-dangling-reference.....	66
		Wno-date-time.....	154
		Wno-declaration-after-statement.....	145
		Wno-declaration-missing-parameter-type... 160	

Wno-defaulted-function-deleted.....	80	Wno-header-guard.....	159
Wno-delete-incomplete.....	80	Wno-if-not-aligned.....	117
Wno-delete-non-virtual-dtor.....	67	Wno-ignored-attributes.....	117
Wno-deprecated.....	163	Wno-ignored-qualifiers.....	117
Wno-deprecated-copy.....	67	Wno-implicit.....	115
Wno-deprecated-copy-dtor.....	67	Wno-implicit-fallthrough.....	115
Wno-deprecated-declarations.....	163	Wno-implicit-function-declaration.....	115
Wno-deprecated-enum-enum-conversion.....	68	Wno-implicit-int.....	114
Wno-deprecated-enum-float-conversion.....	68	Wno-inaccessible-base.....	81
Wno-deprecated-literal-operator.....	68	Wno-incompatible-pointer-types.....	141
Wno-deprecated-non-prototype.....	160	Wno-infinite-recursion.....	114
Wno-deprecated-openmp.....	163	Wno-inherited-variadic-ctor.....	81
Wno-deprecated-variadic-comma-omission....	68	Wno-init-list-lifetime.....	68
Wno-designated-init.....	169	Wno-init-self.....	114
Wno-disabled-optimization.....	168	Wno-inline.....	165
Wno-discarded-array-qualifiers.....	141	Wno-int-conversion.....	141
Wno-discarded-qualifiers.....	141	Wno-int-in-bool-context.....	165
Wno-div-by-zero.....	142	Wno-int-to-pointer-cast.....	166
Wno-double-promotion.....	109	Wno-interference-size.....	165
Wno-duplicate-decl-specifier.....	109	Wno-invalid-constexpr.....	69
Wno-duplicated-branches.....	141	Wno-invalid-imported-macros.....	69
Wno-duplicated-cond.....	141	Wno-invalid-memory-model.....	128
Wno-efc++.....	73	Wno-invalid-offsetof.....	81
Wno-elaborated-enum-base.....	68	Wno-invalid-pch.....	166
Wno-empty-body.....	154	Wno-invalid-utf8.....	166
Wno-endif-labels.....	149, 154	Wno-jump-misses-init.....	155
Wno-enum-compare.....	154	Wno-keyword-macro.....	159
Wno-enum-conversion.....	154	Wno-larger-than.....	146
Wno-enum-int-mismatch.....	154	Wno-literal-suffix.....	69
Wno-error.....	101	Wno-logical-not-parentheses.....	158
Wno-error=.....	101	Wno-logical-op.....	157
Wno-exceptions.....	74	Wno-long-long.....	166
Wno-expansion-to-defined.....	148	Wno-lto-type-mismatch.....	168
Wno-expose-global-module-tu-local.....	78	Wno-main.....	117
Wno-external-tu-local.....	78	Wno-maybe-musttail-local-addr.....	113
Wno-extra.....	105, 161, 163	Wno-maybe-uninitialized.....	129
Wno-extra-semi.....	80	Wno-memset-elt-size.....	156
Wno-fatal-errors.....	101	Wno-memset-transposed-args.....	156
Wno-flex-array-member-not-at-end.....	155	Wno-misleading-indentation.....	117
Wno-float-conversion.....	155	Wno-mismatched-dealloc.....	119
Wno-float-equal.....	143	Wno-mismatched-new-delete.....	76
Wno-format.....	109, 135	Wno-mismatched-tags.....	76
Wno-format-contains-nul.....	110	Wno-missing-attributes.....	118
Wno-format-diag.....	110	Wno-missing-braces.....	119
Wno-format-extra-args.....	110	Wno-missing-declarations.....	160
Wno-format-nonliteral.....	112	Wno-missing-field-initializers.....	161
Wno-format-overflow.....	110, 111	Wno-missing-format-attribute.....	135
Wno-format-security.....	112	Wno-missing-include-dirs.....	119
Wno-format-signedness.....	112	Wno-missing-noreturn.....	135
Wno-format-truncation.....	112	Wno-missing-parameter-name.....	160
Wno-format-y2k.....	113	Wno-missing-parameter-type.....	160
Wno-format-zero-length.....	112	Wno-missing-profile.....	119
Wno-frame-address.....	141	Wno-missing-prototypes.....	160
Wno-frame-larger-than.....	146	Wno-missing-requires.....	161
Wno-free-labels.....	159	Wno-missing-template-keyword.....	161
Wno-free-nonheap-object.....	146	Wno-missing-variable-declarations.....	160
Wno-global-module.....	81	Wno-misspelled-isr.....	364
Wno-hardened.....	115	Wno-multichar.....	162

Wno-multiple-inheritance.....	76	Wno-restrict	164
Wno-multiple-parameter-fwd-decl-lists....	160	Wno-return-local-addr	121
Wno-multistatement-macros.....	120	Wno-return-mismatch	122
Wno-musttail-local-addr.....	113	Wno-return-type.....	122
Wno-namespaces	77	Wno-scalar-storage-order.....	155
Wno-narrowing	70	Wno-selector.....	85
Wno-nested-externs.....	165	Wno-self-move	120
Wno-noexcept.....	70	Wno-sequence-point.....	121
Wno-noexcept-type.....	70	Wno-sfinae-incomplete	74
Wno-non-c-typedef-for-linkage.....	74	Wno-shadow.....	145
Wno-non-template-friend.....	74	Wno-shadow-ivar	145
Wno-non-virtual-dtor	71	Wno-shadow=compatible-local	145
Wno-nonnull.....	113	Wno-shadow=global.....	145
Wno-nonnull-compare	113	Wno-shadow=local.....	145
Wno-nonportable-cfstrings.....	384	Wno-shift-count-negative.....	122
Wno-normalized.....	162	Wno-shift-count-overflow.....	122
Wno-nrvo.....	114	Wno-shift-negative-value.....	122
Wno-null-dereference	113	Wno-shift-overflow.....	122
Wno-objc-root-class.....	85	Wno-sign-compare.....	155
Wno-odr	163	Wno-sign-conversion	155
Wno-old-style-cast.....	74	Wno-sign-promo	75
Wno-old-style-declaration.....	159	Wno-sized-deallocation.....	81
Wno-old-style-definition.....	159	Wno-sizeof-array-argument.....	156
Wno-openacc-dims.....	333	Wno-sizeof-array-div	155
Wno-openacc-parallelism.....	163	Wno-sizeof-pointer-div.....	156
Wno-openmp.....	163	Wno-sizeof-pointer-memaccess	156
Wno-openmp-simd.....	163	Wno-stack-protector	168
Wno-overflow	163	Wno-stack-usage.....	147
Wno-overlength-strings.....	168	Wno-strict-aliasing	130
Wno-overloaded-virtual.....	75	Wno-strict-flex-arrays	134
Wno-override-init.....	163	Wno-strict-null-sentinel.....	74
Wno-override-init-side-effects.....	163	Wno-strict-overflow	131
Wno-packed.....	163	Wno-strict-prototypes.....	159
Wno-packed-bitfield-compat	164	Wno-strict-selector-match.....	85
Wno-packed-not-aligned.....	164	Wno-string-compare.....	131
Wno-padded.....	164	Wno-stringop-overflow	131, 132
Wno-parentheses	120	Wno-stringop-overread.....	133
Wno-pedantic	102	Wno-stringop-truncation.....	133
Wno-pedantic-ms-format.....	147	Wno-subobject-linkage	73
Wno-pessimizing-move	71	Wno-suggest-attribute=.....	134
Wno-placement-new.....	79	Wno-suggest-attribute=cold	135
Wno-pmf-conversions.....	75, 1037	Wno-suggest-attribute=const	135
Wno-pointer-arith.....	147	Wno-suggest-attribute=format	135
Wno-pointer-compare	147	Wno-suggest-attribute=malloc	135
Wno-pointer-sign	168	Wno-suggest-attribute=noreturn.....	135
Wno-pointer-to-int-cast.....	166	Wno-suggest-attribute=pure	135
Wno-pragma-once-outside-header.....	130	Wno-suggest-attribute=returns_nonnull....	135
Wno-pragmas.....	130	Wno-suggest-final-methods.....	82
Wno-prio-ctor-dtor.....	130	Wno-suggest-final-types	81
Wno-property-assign-default	85	Wno-suggest-override	82
Wno-protocol.....	85	Wno-switch.....	123
Wno-psabi.....	108	Wno-switch-bool.....	123
Wno-range-loop-construct.....	72	Wno-switch-default.....	123
Wno-redundant-decls	164	Wno-switch-enum.....	123
Wno-redundant-move.....	72	Wno-switch-outside-range.....	123
Wno-redundant-tags.....	73	Wno-switch-unreachable	123
Wno-register.....	71	Wno-sync-nand.....	124
Wno-reorder	71	Wno-system-headers.....	142

Wno-tautological-compare.....	142	Wnonportable-cfstrings.....	384
Wno-template-body.....	77	Wnormalized.....	162
Wno-template-id-ctor.....	77	Wnormalized=.....	162
Wno-template-names-tu-local.....	77	Wnrvo.....	114
Wno-templates.....	75	Wnull-dereference.....	113
Wno-terminate.....	78	Wobjc-root-class.....	85
Wno-traditional.....	144	Wodr.....	163
Wno-traditional-conversion.....	145	Wold-style-cast.....	74
Wno-trailing-whitespace.....	143	Wold-style-declaration.....	159
Wno-trampolines.....	143	Wold-style-definition.....	159
Wno-trigraphs.....	148	Wopenacc-dims.....	333
Wno-trivial-auto-var-init.....	124	Wopenacc-parallelism.....	163
Wno-tsan.....	148	Wopenmp.....	163
Wno-type-limits.....	148	Wopenmp-simd.....	163
Wno-undeclared-selector.....	86	Woverflow.....	163
Wno-undef.....	148	Woverlength-strings.....	168
Wno-unicode.....	166	Woverloaded-virtual.....	75
Wno-uninitialized.....	128	Woverride-init.....	163
Wno-unknown-pragmas.....	130	Woverride-init-side-effects.....	163
Wno-unsafe-loop-optimizations.....	147	Wp.....	275
Wno-unsuffixed-float-constants.....	168	Wpacked.....	163
Wno-terminated-string-initialization..	138	Wpacked-bitfield-compat.....	164
Wno-unused.....	126	Wpacked-not-aligned.....	164
Wno-unused-but-set-parameter.....	124	Wpadded.....	164
Wno-unused-but-set-variable.....	124	Wparentheses.....	120
Wno-unused-const-variable.....	126	Wpedantic.....	102
Wno-unused-function.....	125	Wpedantic-ms-format.....	147
Wno-unused-label.....	125	Wpessimizing-move.....	71
Wno-unused-local-typedefs.....	125	Wplacement-new.....	79
Wno-unused-macros.....	148	Wpmf-conversions.....	75
Wno-unused-parameter.....	125	Wpointer-arith.....	147, 794
Wno-unused-result.....	126	Wpointer-compare.....	147
Wno-unused-value.....	126	Wpointer-sign.....	168
Wno-unused-variable.....	126	Wpointer-to-int-cast.....	166
Wno-use-after-free.....	127	Wpragma-once-outside-header.....	130
Wno-useless-cast.....	128	Wpragmas.....	130
Wno-varargs.....	166	Wprio-ctor-dtor.....	130
Wno-variadic-macros.....	166	Wproperty-assign-default.....	85
Wno-vector-operation-performance.....	166	Wprotocol.....	85
Wno-vexing-parse.....	78	Wpsabi.....	108
Wno-virtual-inheritance.....	77	Wrange-loop-construct.....	72
Wno-virtual-move-assign.....	77	wrapper.....	43
Wno-vla.....	166	Wredundant-decls.....	164
Wno-vla-larger-than.....	167	Wredundant-move.....	72
Wno-vla-parameter.....	167	Wredundant-tags.....	73
Wno-volatile.....	79	Wregister.....	71
Wno-volatile-register-var.....	167	Wreorder.....	71
Wno-write-strings.....	152	Wrestrict.....	164
Wno-xor-used-as-pow.....	168	Wreturn-local-addr.....	121
Wno-zero-as-null-pointer-constant.....	142	Wreturn-mismatch.....	122
Wno-zero-length-bounds.....	142	Wreturn-type.....	122
Wnoexcept.....	70	write-dependencies.....	270
Wnoexcept-type.....	70	write-user-dependencies.....	270
Wnon-c-typedef-for-linkage.....	74	Wscalar-storage-order.....	155
Wnon-template-friend.....	74	Wselector.....	85
Wnon-virtual-dtor.....	71	Wself-move.....	120
Wnonnull.....	113	Wsequence-point.....	121
Wnonnull-compare.....	113	Wsfinae-incomplete.....	74

Wshadow	145	Wtrailing-whitespace=	143
Wshadow-ivar	145	Wtrampolines	143
Wshadow=compatible-local	145	Wtrigraphs	148
Wshadow=global	145	Wtrivial-auto-var-init	124
Wshadow=local	145	Wtsan	148
Wshift-count-negative	122	Wtype-limits	148
Wshift-count-overflow	122	Wundeclared-selector	86
Wshift-negative-value	122	Wundef	148
Wshift-overflow	122	Wunicode	166
Wsign-compare	155	Wuninitialized	128
Wsign-conversion	155	Wunknown-pragmas	130
Wsign-promo	75	Wunsafe-loop-optimizations	147
Wsizeof-deallocation	81	Wunsuffixed-float-constants	168
Wsizeof-array-argument	156	Wunterminated-string-initialization	138
Wsizeof-array-div	155	Wunused	126
Wsizeof-pointer-div	156	Wunused-but-set-parameter	124
Wsizeof-pointer-memaccess	156	Wunused-but-set-parameter=	124
Wstack-protector	168	Wunused-but-set-variable	124
Wstack-usage	147	Wunused-but-set-variable=	124
Wstrict-aliasing	130	Wunused-const-variable	126
Wstrict-flex-arrays	134	Wunused-function	125
Wstrict-null-sentinel	74	Wunused-label	125
Wstrict-overflow	131	Wunused-local-typedefs	125
Wstrict-prototypes	159	Wunused-macros	148
Wstrict-selector-match	85	Wunused-parameter	125
Wstring-compare	131	Wunused-result	126
Wstringop-overflow	131, 132	Wunused-value	126
Wstringop-overread	133	Wunused-variable	126
Wstringop-truncation	133	Wuse-after-free	127
Wsubobject-linkage	73	Wuseless-cast	128
Wsuggest-attribute=	134	Wvarargs	166
Wsuggest-attribute=cold	135	Wvariadic-macros	166
Wsuggest-attribute=const	135	Wvector-operation-performance	166
Wsuggest-attribute=format	135	Wvexing-parse	78
Wsuggest-attribute=malloc	135	Wvirtual-inheritance	77
Wsuggest-attribute=noreturn	135	Wvirtual-move-assign	77
Wsuggest-attribute=pure	135	Wvla	166
Wsuggest-attribute=returns_nonnull	135	Wvla-larger-than=	167
Wsuggest-final-methods	82	Wvla-parameter	167
Wsuggest-final-types	81	Wvolatile	79
Wsuggest-override	82	Wvolatile-register-var	167
Wswitch	123	Wwrite-strings	152
Wswitch-bool	123	Wxor-used-as-pow	168
Wswitch-default	123	Wzero-as-null-pointer-constant	142
Wswitch-enum	123	Wzero-init-padding-bits=	169
Wswitch-outside-range	123	Wzero-length-bounds	142
Wswitch-unreachable	123		
Wsync-nand	124		
Wsystem-headers	142		
Wtautological-compare	142		
Wtemplate-body	77		
Wtemplate-id-ctor	77		
Wtemplate-names-tu-local	77		
Wtemplates	75		
Wterminate	78		
Wtraditional	144		
Wtraditional-conversion	145		
Wtrailing-whitespace	143		

X

x	36
Xassembler	275
Xbind-lazy	512
Xbind-now	512
Xlinker	281
Xpreprocessor	275

Y

Ym	507
YP	507

Z

z	282
Z	282

A.2 Attribute Index

This index lists GNU extension attributes only, e.g. those that should use the ‘gnu::’ namespace prefix in the standard C and C++ attribute syntax.

A

abi_tag	1037
absdata, AVR	661
access	595
address, AVR	660
alias	597
aligned	597
alloc_align	599
alloc_size	599
altivec, PowerPC	682
always_inline	600
amdgpu_hsa_kernel, AMD GCN	652
arch=, ARM	656
artificial	600
assume	600
assume_aligned	601
aux, ARC	654

B

below100, Xstormy16	702
break_handler, MicroBlaze	671
brk_interrupt, RL78	684
btfl_decl_tag	601
btfl_type_tag	601

C

callee_pop_aggregate_return, x86	689
cdecl, x86-32	688
cf_check, x86	701
cleanup	602
code_readable, MIPS	676
cold	602, 1039
common	603
const	603
constructor	604
copy	604
counted_by	605
critical, MSP430	676

D

deprecated	607
designated_init	608
destructor	604
disinterrupt, Epiphany	663
dllexport, Microsoft Windows	672
dllimport, Microsoft Windows	672

E

eightbit_data, H8/300	665
either, MSP430	677
error	608
exception, NDS32	678
exception_handler, Blackfin	661
expected_throw	608
externally_visible	608

F

fallthrough	608
far, MIPS	675
fast_interrupt, MicroBlaze	671
fast_interrupt, RX	684
fastcall, x86-32	688
fd_arg	609
fd_arg_read	609
fd_arg_write	609
fentry_name, x86	701
fentry_section, x86	701
flatten	610
force_align_arg_pointer, x86	690
force_l32, Xtensa	702
format	610
format_arg	611
forwarder_section, Epiphany	663
function_return, x86	700
function_vector, H8/300	664
function_vector, SH	686

G

gcc_struct, PowerPC	682
gcc_struct, x86	702
general-regs-only, ARM	655
gnu_inline	612

H

hardbool	612
hot	602, 1039
hotpatch, S/390	685

I

ifunc	613
indirect_branch, x86	700
indirect_return, x86	701
init_priority	1037
interrupt	615
interrupt(<i>num</i>), AVR	657
interrupt, ARC	653
interrupt, ARM	655
interrupt, AVR	657
interrupt, C-SKY	663
interrupt, Epiphany	663
interrupt, m68k	671
interrupt, M32R/D	670
interrupt, MIPS	674
interrupt, MSP430	677
interrupt, NDS32	678
interrupt, RISC-V	682
interrupt, RL78	684
interrupt, RX	684
interrupt, V850	688
interrupt, Visium	688
interrupt, x86	690
interrupt, Xstormy16	702
interrupt_handler	615
interrupt_handler, Blackfin	661
interrupt_handler, H8/300	664
interrupt_handler, m68k	671
interrupt_handler, MicroBlaze	671
interrupt_handler, SH	686
interrupt_handler, V850	688
interrupt_thread, fido	671
io, AVR	660
io_low, AVR	660
isr, ARM	655
isr, C-SKY	663

J

jli_always, ARC	654
jli_fixed, ARC	654

K

keep_interrupts_masked, MIPS	674
kernel helper, BPF	663
kernel, Nvidia PTX	679
kspisusp, Blackfin	661

L

l1_data, Blackfin	662
l1_data_A, Blackfin	662
l1_data_B, Blackfin	662
l1_text, Blackfin	661
l2, Blackfin	662
leaf	615
long_call, ARC	654
long_call, ARM	655
long_call, Epiphany	664
long_call, MIPS	675
longcall, Blackfin	662
longcall, PowerPC	679
lower, MSP430	677

M

malloc	616
may_alias	617
medium_call, ARC	654
micromips, MIPS	676
mips16, MIPS	675
mode	618
model, IA-64	665
model, LoongArch	670
model, M32R/D	670
monitor, H8/300	665
ms_abi, x86	689
ms_hook_prologue, x86	689
ms_struct, PowerPC	682
ms_struct, x86	702
musttail	618

N

naked	619
near, MIPS	675
nested, NDS32	678
nested_ready, NDS32	678
nesting, Blackfin	662
nmi, NDS32	678
nmi_handler, Blackfin	662
no_callee_saved_registers, x86	690
no_caller_saved_registers, x86	690
no_dangling	1038
no_gccisr, AVR	658
no_icf	620
no_instrument_function	620
no_reorder	620
no_sanitize_address	620
no_sanitize_thread	621
no_sanitize_undefined	621
no_split_stack	621
no_stack_limit	621
no_stack_protector	621
noblock, AVR	658
nocf_check, x86	700
noclone	621

<code>nocommon</code>	603
<code>nocompression</code> , MIPS.....	676
<code>nodirect_extern_access</code> , x86.....	702
<code>noinit</code>	621
<code>noinline</code>	621
<code>noipa</code>	622
<code>nomicromps</code> , MIPS.....	676
<code>nomips16</code> , MIPS.....	675
<code>nonnull</code>	622
<code>nonnull_if_nonzero</code>	623
<code>nonstring</code>	623
<code>noplt</code>	624
<code>noreturn</code>	624
<code>nosave_low_regs</code> , SH.....	687
<code>not_nested</code> , NDS32.....	678
<code>nothrow</code>	625
<code>notshared</code> , ARM.....	656
<code>null_terminated_string_arg</code>	625

O

<code>objc_nullability</code>	626
<code>objc_root_class</code>	626
<code>optimize</code>	626
<code>OS_main</code> , AVR.....	659
<code>OS_task</code> , AVR.....	659
<code>OS_Task</code> , H8/300.....	665

P

<code>packed</code>	627
<code>partial_save</code> , NDS32.....	678
<code>patchable_function_entry</code>	628
<code>pcs</code> , ARM.....	655
<code>persistent</code>	628
<code>prefer-vector-width</code> , x86.....	699
<code>preserve_access_index</code> , BPF.....	663
<code>preserve_none</code> , x86.....	690
<code>progmem</code> , AVR.....	659
<code>pure</code>	628

R

<code>reentrant</code> , MSP430.....	677
<code>regparm</code> , x86.....	689
<code>renesas</code> , SH.....	687
<code>reproducible</code>	629
<code>resbank</code> , SH.....	687
<code>reset</code> , NDS32.....	678
<code>retain</code>	629
<code>returns_nonnull</code>	630
<code>returns_twice</code>	630
<code>riscv_vector_cc</code> , RISC-V.....	683

S

<code>saddr</code> , RL78.....	684
<code>save_all</code> , NDS32.....	678
<code>save_volatiles</code> , MicroBlaze.....	671
<code>saveall</code> , Blackfin.....	662
<code>saveall</code> , H8/300.....	664
<code>scalar_storage_order</code>	630
<code>sda</code> , V850.....	688
<code>section</code>	631
<code>secure_call</code> , ARC.....	654
<code>selectany</code> , Microsoft Windows.....	673
<code>sentinel</code>	631
<code>shared</code> , Microsoft Windows.....	673
<code>shared</code> , Nvidia PTX.....	679
<code>short_call</code> , ARC.....	654
<code>short_call</code> , ARM.....	655
<code>short_call</code> , Epiphany.....	664
<code>short_call</code> , MIPS.....	675
<code>shortcall</code> , Blackfin.....	662
<code>shortcall</code> , PowerPC.....	679
<code>signal(num)</code> , AVR.....	657
<code>signal</code> , AVR.....	657
<code>simd</code>	632
<code>sp_switch</code> , SH.....	687
<code>sseregparm</code> , x86.....	689
<code>stack_protect</code>	632
<code>stdcall</code> , x86-32.....	690
<code>strict_flex_array</code>	632
<code>strub</code>	633
<code>symver</code>	637
<code>syscall_linkage</code> , IA-64.....	665
<code>sysv_abi</code> , x86.....	689

T

<code>tainted_args</code>	638
<code>target</code>	638
<code>target("3dnow")</code> , x86.....	691
<code>target("3dnowa")</code> , x86.....	691
<code>target("80387")</code> , x86.....	699
<code>target("abm")</code> , x86.....	692
<code>target("adx")</code> , x86.....	692
<code>target("aes")</code> , x86.....	692
<code>target("align-stringops")</code> , x86.....	699
<code>target("altivec")</code> , PowerPC.....	679
<code>target("amx-avx512")</code> , x86.....	698
<code>target("amx-bf16")</code> , x86.....	696
<code>target("amx-complex")</code> , x86.....	697
<code>target("amx-fp16")</code> , x86.....	697
<code>target("amx-fp8")</code> , x86.....	698
<code>target("amx-int8")</code> , x86.....	696
<code>target("amx-movrs")</code> , x86.....	698
<code>target("amx-tf32")</code> , x86.....	698
<code>target("amx-tile")</code> , x86.....	696
<code>target("apxf")</code> , x86.....	698
<code>target("arch=")</code> , AArch64.....	650
<code>target("arch=")</code> , LoongArch.....	666
<code>target("arch=")</code> , RISC-V.....	683

target("arch=ARCH"), x86.....	699	target("general-regs-only"), x86	699
target("arm"), ARM.....	656	target("gfni"), x86	694
target("avoid-indexed- addresses"), PowerPC.....	681	target("hard-dfp"), PowerPC.....	680
target("avx"), x86.....	692	target("hle"), x86.....	694
target("avx10.1"), x86.....	698	target("hreset"), x86.....	697
target("avx10.2"), x86.....	698	target("ieee-fp"), x86.....	699
target("avx2"), x86.....	692	target("indirect_return"), AArch64.....	651
target("avx512bitalg"), x86.....	692	target("inline-all-stringops"), x86.....	699
target("avx512bw"), x86.....	692	target("inline-stringops- dynamically"), x86	699
target("avx512cd"), x86.....	692	target("isel"), PowerPC.....	680
target("avx512dq"), x86.....	692	target("kl"), x86.....	697
target("avx512er"), x86.....	692	target("lam-bh"), LoongArch.....	667
target("avx512f"), x86.....	692	target("lamcas"), LoongArch.....	667
target("avx512ifma"), x86.....	692	target("lasx"), LoongArch.....	666
target("avx512vbmi"), x86.....	692	target("ld-seq-sa"), LoongArch.....	668
target("avx512vbmi2"), x86.....	692	target("longcall"), PowerPC.....	682
target("avx512vl"), x86.....	692	target("lsx"), LoongArch.....	666
target("avx512vnni"), x86.....	693	target("lwp"), x86.....	694
target("avx512vpopcntdq"), x86.....	693	target("lzcmt"), x86.....	694
target("avxifma"), x86.....	697	target("max-vectorization"), AArch64.....	651
target("avxneconvert"), x86.....	697	target("max-vectorization"), RISC-V.....	683
target("avxvnni"), x86.....	697	target("mfcrrf"), PowerPC.....	680
target("avxvnniint16"), x86.....	697	target("mmx"), x86.....	694
target("avxvnniint8"), x86.....	697	target("movbe"), x86.....	694
target("bmi"), x86.....	693	target("movdir64b"), x86.....	694
target("bmi2"), x86.....	693	target("movdiri"), x86.....	694
target("branch-protection"), AArch64.....	650	target("movrs"), x86.....	698
target("cld"), x86.....	698	target("mulhw"), PowerPC.....	680
target("cldemote"), x86.....	693	target("multiple"), PowerPC.....	680
target("clflushopt"), x86.....	693	target("mwait"), x86.....	694
target("clwb"), x86.....	693	target("mwaitx"), x86.....	694
target("clzero"), x86.....	693	target("omit-leaf-frame- pointer"), AArch64.....	650
target("cmodel="), AArch64.....	650	target("outline-atomics"), AArch64.....	651
target("cmodel="), LoongArch.....	666	target("paired"), PowerPC.....	681
target("cmpb"), PowerPC.....	679	target("pclmul"), x86.....	694
target("cmpccxadd"), x86.....	697	target("pconfig"), x86.....	694
target("cpu="), AArch64.....	650	target("pku"), x86.....	694
target("cpu="), RISC-V.....	683	target("popcnt"), x86.....	694
target("cpu=CPU"), PowerPC.....	682	target("popcntb"), PowerPC.....	680
target("crc32"), x86.....	693	target("popcntd"), PowerPC.....	680
target("cx16"), x86.....	693	target("powerpc-gfxopt"), PowerPC.....	681
target("default"), x86.....	693	target("powerpc-gpopt"), PowerPC.....	681
target("div32"), LoongArch.....	667	target("prefetchi"), x86.....	697
target("dlmzb"), PowerPC.....	679	target("preserve_none"), AArch64.....	651
target("f16c"), x86.....	693	target("prfchw"), x86.....	694
target("fancy-math-387"), x86.....	698	target("ptwrite"), x86.....	695
target("fix-cortex-a53- 835769"), AArch64.....	649	target("raoint"), x86.....	697
target("fma"), x86.....	693	target("rdpid"), x86.....	695
target("fma4"), x86.....	693	target("rdrnd"), x86.....	695
target("fpmath=FPATH"), x86.....	699	target("rdseed"), x86.....	695
target("fprnd"), PowerPC.....	680	target("recip"), x86.....	699
target("fpu="), ARM.....	656	target("recip-precision"), PowerPC.....	681
target("friz"), PowerPC.....	681	target("recipe"), LoongArch.....	667
target("fsgsbase"), x86.....	693	target("rtm"), x86.....	695
target("fxsr"), x86.....	693	target("sahf"), x86.....	695
target("general-regs-only"), AArch64.....	649	target("scq"), LoongArch.....	667

target("sgx"), x86.....	695	target_clones.....	639
target("sha"), x86.....	695	target_clones, LoongArch.....	668
target("sha512"), x86.....	698	target_version.....	639
target("shstk"), x86.....	695	target_version, LoongArch.....	669
target("sign-return-address"), AArch64...	650	tda, V850.....	688
target("sm3"), x86.....	698	thiscall, x86-32.....	688
target("sm4"), x86.....	698	tiny_data, H8/300.....	665
target("sse"), x86.....	695	tls_model.....	640
target("sse2"), x86.....	695	transparent_union.....	640
target("sse3"), x86.....	695	trap_exit, SH.....	687
target("sse4"), x86.....	695	trapa_handler, SH.....	687
target("sse4.1"), x86.....	695		
target("sse4.2"), x86.....	695		
target("sse4a"), x86.....	695		
target("ssse3"), x86.....	696		
target("strict-align"), AArch64.....	650		
target("strict-align"), LoongArch.....	666		
target("string"), PowerPC.....	681		
target("tbn"), x86.....	696		
target("thumb"), ARM.....	656		
target("tls-dialect="), AArch64.....	650		
target("tune="), AArch64.....	650		
target("tune="), LoongArch.....	666		
target("tune="), RISC-V.....	683		
target("tune=TUNE"), PowerPC.....	682		
target("tune=TUNE"), x86.....	699		
target("uintr"), x86.....	696		
target("update"), PowerPC.....	680		
target("usermsr"), x86.....	698		
target("vaes"), x86.....	696		
target("vpclmulqdq"), x86.....	696		
target("vsx"), PowerPC.....	681		
target("waitpkg"), x86.....	696		
target("wbnoinvd"), x86.....	696		
target("widekl"), x86.....	697		
target("xop"), x86.....	696		
target("xsave"), x86.....	696		
target("xsavec"), x86.....	696		
target("xsaveopt"), x86.....	696		
target("xsaves"), x86.....	696		
target, AArch64.....	649		
target, ARM.....	656		
target, LoongArch.....	666		
target, PowerPC.....	679		
target, RISC-V.....	683		
target, S/390.....	685		
target, x86.....	691		
		U	
		unavailable.....	641
		uncached, ARC.....	654
		uninitialized.....	641
		unsequenced.....	641
		unused.....	642
		upper, MSP430.....	677
		use_debug_exception_return, MIPS.....	675
		use_hazard_barrier_return, MIPS.....	676
		use_shadow_register_set, MIPS.....	674
		used.....	642
		V	
		vector, RX.....	685
		vector_size.....	642
		version_id, IA-64.....	665
		visibility.....	643
		W	
		wakeup, MSP430.....	677
		warm, NDS32.....	678
		warn_unused.....	1038
		warn_unused_result.....	646
		warning.....	608
		weak.....	646
		weakref.....	646
		Z	
		zda, V850.....	688
		zero_call_used_regs.....	647

A.3 Concept and Symbol Index

#		>	
#pragma	706	'>' in constraint	745
#pragma implementation	1034		
#pragma implementation, implied	1034	?	
#pragma interface	1033	? : extensions	790
		? : side effect	790
\$		-	
\$	792	'_' in variables in macros	786
		__atomic builtins	822
%		__atomic_add_fetch	825
'%' in constraint	748	__atomic_always_lock_free	827
		__atomic_and_fetch	825
&		__atomic_clear	826
'&' in constraint	748	__atomic_compare_exchange	825
,		__atomic_compare_exchange_n	825
'	1104	__atomic_exchange	825
+		__atomic_exchange_n	825
'+' in constraint	748	__atomic_fetch_add	826
-		__atomic_fetch_and	826
'-' in constraint	748	__atomic_fetch_nand	826
-lgcc, use with -nodefaultlibs	278	__atomic_fetch_or	826
-lgcc, use with -nostdlib	278	__atomic_fetch_sub	826
-march feature modifiers	324	__atomic_fetch_xor	826
-mcpu feature modifiers	324	__atomic_is_lock_free	827
-nodefaultlibs and unresolved references	278	__atomic_load	824
-nostdlib and unresolved references	278	__atomic_load_n	824
.		__atomic_nand_fetch	826
.sdata/.sdata2 references (PowerPC)	481	__atomic_or_fetch	825
/		__atomic_signal_fence	827
//	791	__atomic_store	824
:		__atomic_store_n	824
':' in constraint	747	__atomic_sub_fetch	825
		__atomic_test_and_set	826
<		__atomic_thread_fence	827
'<' in constraint	745	__atomic_xor_fetch	825
=		__auto_type in GNU C	787
'=' in constraint	748	__builtin__clear_cache	841
		__builtin__memcpy_chk	830
		__builtin__memmove_chk	830
		__builtin__memcpy_chk	830
		__builtin__memset_chk	830
		__builtin__snprintf_chk	831
		__builtin__sprintf_chk	831
		__builtin__stpcpy_chk	830
		__builtin__strcat_chk	830
		__builtin__strcpy_chk	830
		__builtin__strncat_chk	830
		__builtin__strncpy_chk	830
		__builtin__strub_enter	818
		__builtin__strub_leave	818
		__builtin__strub_update	818
		__builtin__vsnprintf_chk	832
		__builtin__vsprintf_chk	831
		__builtin_add_overflow	809
		__builtin_add_overflow_p	810

__builtin_addc.....	811	__builtin_bswap128.....	806
__builtin_addcl.....	811	__builtin_bswap16.....	806
__builtin_addcll.....	811	__builtin_bswap32.....	806
__builtin_addf128_round_to_odd.....	928	__builtin_bswap64.....	806
__builtin_alloca.....	812	__builtin_bswapg.....	806
__builtin_alloca_with_align.....	812	__builtin_btf_type_id.....	856
__builtin_alloca_with_align_and_max.....	813	__builtin_call_with_static_chain.....	816
__builtin_apply.....	814	__builtin_cfuged.....	930
__builtin_apply_args.....	814	__builtin_choose_expr.....	835
__builtin_arc_aligned.....	845	__builtin_classify_type.....	842
__builtin_arc_brk.....	845	__builtin_clear_padding.....	838
__builtin_arc_core_read.....	845	__builtin_clrslb.....	802
__builtin_arc_core_write.....	845	__builtin_clrslbg.....	803
__builtin_arc_divaw.....	845	__builtin_clrslbl.....	802
__builtin_arc_flag.....	845	__builtin_clrslbl1.....	803
__builtin_arc_lr.....	845	__builtin_clz.....	802
__builtin_arc_mul64.....	845	__builtin_clzg.....	803
__builtin_arc_mulu64.....	846	__builtin_clzl.....	802
__builtin_arc_nop.....	846	__builtin_clzll.....	803
__builtin_arc_norm.....	846	__builtin_cntlzd.....	930
__builtin_arc_normw.....	846	__builtin_cnttzm.....	931
__builtin_arc_rtie.....	846	__builtin_complex.....	577
__builtin_arc_sleep.....	846	__builtin_constant_p.....	836
__builtin_arc_sr.....	846	__builtin_convertvector.....	821
__builtin_arc_swap.....	846	__builtin_copysignq.....	1002
__builtin_arc_swi.....	846	__builtin_counted_by_ref.....	837
__builtin_arc_sync.....	846	__builtin_cpu_init.....	921, 1003
__builtin_arc_trap_s.....	847	__builtin_cpu_is.....	921, 1003
__builtin_arc_unimp_s.....	847	__builtin_cpu_supports.....	922, 1006
__builtin_assoc_barrier.....	840	__builtin_crc16_data16.....	808
__builtin_assume_aligned.....	841	__builtin_crc16_data8.....	808
__builtin_avr_cli.....	852	__builtin_crc32_data16.....	808
__builtin_avr_delay_cycles.....	852	__builtin_crc32_data32.....	808
__builtin_avr_flash_segment.....	853	__builtin_crc32_data8.....	808
__builtin_avr_fmuls.....	852	__builtin_crc64_data16.....	808
__builtin_avr_fmulsu.....	852	__builtin_crc64_data32.....	808
__builtin_avr_insert_bits.....	852	__builtin_crc64_data64.....	808
__builtin_avr_mask1.....	853	__builtin_crc64_data8.....	808
__builtin_avr_nop.....	852	__builtin_crc8_data8.....	808
__builtin_avr_nops.....	853	__builtin_ctz.....	802
__builtin_avr_sei.....	852	__builtin_ctzg.....	803
__builtin_avr_sleep.....	852	__builtin_ctzl.....	802
__builtin_avr_strlen_flash.....	854	__builtin_ctzll.....	803
__builtin_avr_strlen_flashx.....	854	__builtin_darn.....	929
__builtin_avr_strlen_memx.....	854	__builtin_darn_32.....	929
__builtin_avr_swap.....	852	__builtin_darn_raw.....	929
__builtin_avr_wdr.....	852	__builtin_divf128_round_to_odd.....	928
__builtin_bit_cast.....	838	__builtin_dynamic_object_size.....	830
__builtin_bitreverse128.....	807	__builtin_expect.....	839
__builtin_bitreverse16.....	806	__builtin_expect_with_probability.....	839
__builtin_bitreverse32.....	806	__builtin_extend_pointer.....	843
__builtin_bitreverse64.....	806	__builtin_extract_return_addr.....	816
__builtin_bitreverse8.....	806	__builtin_fabsq.....	1002
__builtin_bitreverseseg.....	807	__builtin_ffs.....	802
__builtin_bpf_load_byte.....	854	__builtin_ffsg.....	803
__builtin_bpf_load_half.....	854	__builtin_ffsl.....	802
__builtin_bpf_load_word.....	854	__builtin_ffsll.....	802
		__builtin_FILE.....	841

__builtin_fmaf128_round_to_odd.....	929	__builtin_mulf128_round_to_odd.....	928
__builtin_fpclassify.....	799	__builtin_nan.....	800
__builtin_frame_address.....	817	__builtin_nand128.....	801
__builtin_frob_return_addr.....	817	__builtin_nand32.....	800
__builtin_FUNCTION.....	841	__builtin_nand64.....	800
__builtin_goacc_parlevel_id.....	843	__builtin_nanf.....	801
__builtin_goacc_parlevel_size.....	843	__builtin_nanfn.....	801
__builtin_has_attribute.....	833	__builtin_nanfnx.....	801
__builtin_huge_val.....	799	__builtin_nanl.....	801
__builtin_huge_valf.....	799	__builtin_nanq.....	1002
__builtin_huge_valfn.....	799	__builtin_nans.....	801
__builtin_huge_valfnx.....	799	__builtin_nansd128.....	801
__builtin_huge_vall.....	799	__builtin_nansd32.....	801
__builtin_huge_valq.....	1002	__builtin_nansd64.....	801
__builtin_ia32_crc32di.....	1016	__builtin_nansf.....	801
__builtin_ia32_crc32hi.....	1016	__builtin_nansfn.....	801
__builtin_ia32_crc32qi.....	1016	__builtin_nansfnx.....	801
__builtin_ia32_crc32si.....	1016	__builtin_nansl.....	801
__builtin_ia32_loadhps.....	1010	__builtin_nansq.....	1002
__builtin_ia32_loadlps.....	1010	__builtin_nds32_isb.....	920
__builtin_ia32_loadss.....	1010	__builtin_nds32_isync.....	920
__builtin_ia32_loadups.....	1010	__builtin_nds32_mfsr.....	920
__builtin_ia32_pause.....	1002	__builtin_nds32_mfusr.....	921
__builtin_ia32_pclmulqdq128.....	1022	__builtin_nds32_mtsr.....	921
__builtin_ia32_storehps.....	1010	__builtin_nds32_mtusr.....	921
__builtin_ia32_storelps.....	1011	__builtin_nds32_setgie_dis.....	921
__builtin_ia32_storeups.....	1010	__builtin_nds32_setgie_en.....	921
__builtin_ia32_vec_ext_v16qi.....	1015	__builtin_non_tx_store.....	997
__builtin_ia32_vec_ext_v2di.....	1016	__builtin_nvptx_brev.....	921
__builtin_ia32_vec_ext_v4sf.....	1015	__builtin_nvptx_brevll.....	921
__builtin_ia32_vec_ext_v4si.....	1015	__builtin_object_size.....	830
__builtin_ia32_vec_set_v16qi.....	1015	__builtin_offsetof.....	788
__builtin_ia32_vec_set_v2di.....	1015	__builtin_operator_delete.....	832
__builtin_ia32_vec_set_v4sf.....	1015	__builtin_operator_new.....	832
__builtin_ia32_vec_set_v4si.....	1015	__builtin_parity.....	802
__builtin_inf.....	800	__builtin_parityg.....	803
__builtin_infd128.....	800	__builtin_parityl.....	802
__builtin_infd32.....	800	__builtin_parityll.....	803
__builtin_infd64.....	800	__builtin_pdepd.....	931
__builtin_inff.....	800	__builtin_pextd.....	931
__builtin_inffn.....	800	__builtin_popcount.....	802, 1016
__builtin_inffnx.....	800	__builtin_popcountg.....	803
__builtin_infl.....	800	__builtin_popcountl.....	802, 1016
__builtin_infq.....	1002	__builtin_popcountll.....	803, 1016
__builtin_is_constant_evaluated.....	837	__builtin_powi.....	802
__builtin_is_virtual_base_of.....	1043	__builtin_powif.....	802
__builtin_iseqsig.....	797	__builtin_powil.....	802
__builtin_isfinite.....	797	__builtin_prefetch.....	842
__builtin_isgreater.....	797	__builtin_preserve_access_index.....	855
__builtin_isgreaterequal.....	797	__builtin_preserve_enum_value.....	856
__builtin_isinf_sign.....	800	__builtin_preserve_field_info.....	855
__builtin_isnormal.....	797	__builtin_preserve_type_info.....	857
__builtin_issignaling.....	801	__builtin_return.....	815
__builtin_isunordered.....	797	__builtin_return_address.....	816
__builtin_LINE.....	841	__builtin_rev_crc16_data16.....	807
__builtin_longjmp.....	814	__builtin_rev_crc16_data8.....	807
__builtin_mul_overflow.....	810	__builtin_rev_crc32_data16.....	807
__builtin_mul_overflow_p.....	810	__builtin_rev_crc32_data32.....	807

__builtin_rev_crc32_data8.....	807	__builtin_riscv_cv_simd_avg_sc_h.....	977
__builtin_rev_crc64_data16.....	808	__builtin_riscv_cv_simd_avg_u_b.....	978
__builtin_rev_crc64_data32.....	808	__builtin_riscv_cv_simd_avg_u_h.....	978
__builtin_rev_crc64_data64.....	807	__builtin_riscv_cv_simd_avg_u_sc_b.....	978
__builtin_rev_crc64_data8.....	807	__builtin_riscv_cv_simd_avg_u_sc_h.....	978
__builtin_rev_crc8_data8.....	807	__builtin_riscv_cv_simd_cmpeq_b.....	987
__builtin_riscv_cv_alu_addN.....	975	__builtin_riscv_cv_simd_cmpeq_h.....	987
__builtin_riscv_cv_alu_addRN.....	975	__builtin_riscv_cv_simd_cmpeq_sc_b..	987, 988
__builtin_riscv_cv_alu_adduN.....	975	__builtin_riscv_cv_simd_cmpeq_sc_h.....	987
__builtin_riscv_cv_alu_adduRN.....	976	__builtin_riscv_cv_simd_cmpge_b.....	989
__builtin_riscv_cv_alu_clip.....	975	__builtin_riscv_cv_simd_cmpge_h.....	989
__builtin_riscv_cv_alu_clipu.....	975	__builtin_riscv_cv_simd_cmpge_sc_b.....	989
__builtin_riscv_cv_alu_extbs.....	975	__builtin_riscv_cv_simd_cmpge_sc_h.....	989
__builtin_riscv_cv_alu_extbz.....	975	__builtin_riscv_cv_simd_cmpgeu_b.....	990
__builtin_riscv_cv_alu_exths.....	975	__builtin_riscv_cv_simd_cmpgeu_h.....	990
__builtin_riscv_cv_alu_exthz.....	975	__builtin_riscv_cv_simd_cmpgeu_sc_b.....	991
__builtin_riscv_cv_alu_max.....	975	__builtin_riscv_cv_simd_cmpgeu_sc_h.....	991
__builtin_riscv_cv_alu_maxu.....	975	__builtin_riscv_cv_simd_cmpgt_b.....	988
__builtin_riscv_cv_alu_min.....	974	__builtin_riscv_cv_simd_cmpgt_h.....	988
__builtin_riscv_cv_alu_minu.....	975	__builtin_riscv_cv_simd_cmpgt_sc_b.....	988
__builtin_riscv_cv_alu_slet.....	974	__builtin_riscv_cv_simd_cmpgt_sc_h.....	988
__builtin_riscv_cv_alu_sletu.....	974	__builtin_riscv_cv_simd_cmpgtu_b.....	990
__builtin_riscv_cv_alu_subN.....	976	__builtin_riscv_cv_simd_cmpgtu_h.....	990
__builtin_riscv_cv_alu_subRN.....	976	__builtin_riscv_cv_simd_cmpgtu_sc_b.....	990
__builtin_riscv_cv_alu_subuN.....	976	__builtin_riscv_cv_simd_cmpgtu_sc_h.....	990
__builtin_riscv_cv_alu_subuRN.....	976	__builtin_riscv_cv_simd_cmple_b.....	990
__builtin_riscv_cv_elw_elw.....	976	__builtin_riscv_cv_simd_cmple_h.....	989
__builtin_riscv_cv_mac_mac.....	973	__builtin_riscv_cv_simd_cmple_sc_b.....	990
__builtin_riscv_cv_mac_machhsN.....	974	__builtin_riscv_cv_simd_cmple_sc_h.....	990
__builtin_riscv_cv_mac_machhsRN.....	974	__builtin_riscv_cv_simd_cmpleu_b.....	991
__builtin_riscv_cv_mac_machhuN.....	974	__builtin_riscv_cv_simd_cmpleu_h.....	991
__builtin_riscv_cv_mac_machhuRN.....	974	__builtin_riscv_cv_simd_cmpleu_sc_b.....	992
__builtin_riscv_cv_mac_macsN.....	974	__builtin_riscv_cv_simd_cmpleu_sc_h..	991, 992
__builtin_riscv_cv_mac_macsRN.....	974	__builtin_riscv_cv_simd_cmplt_b.....	989
__builtin_riscv_cv_mac_macuN.....	974	__builtin_riscv_cv_simd_cmplt_h.....	989
__builtin_riscv_cv_mac_macuRN.....	974	__builtin_riscv_cv_simd_cmplt_sc_b.....	989
__builtin_riscv_cv_mac_msu.....	973	__builtin_riscv_cv_simd_cmplt_sc_h.....	989
__builtin_riscv_cv_mac_mulhhsN.....	973	__builtin_riscv_cv_simd_cmpltu_b.....	991
__builtin_riscv_cv_mac_mulhhsRN.....	974	__builtin_riscv_cv_simd_cmpltu_h.....	991
__builtin_riscv_cv_mac_mulhhuN.....	973	__builtin_riscv_cv_simd_cmpltu_sc_b.....	991
__builtin_riscv_cv_mac_mulhhuRN.....	973	__builtin_riscv_cv_simd_cmpltu_sc_h.....	991
__builtin_riscv_cv_mac_mulsN.....	973	__builtin_riscv_cv_simd_cmpne_b.....	988
__builtin_riscv_cv_mac_mulsRN.....	973	__builtin_riscv_cv_simd_cmpne_h.....	988
__builtin_riscv_cv_mac_muluN.....	973	__builtin_riscv_cv_simd_cmpne_sc_b.....	988
__builtin_riscv_cv_mac_muluRN.....	973	__builtin_riscv_cv_simd_cmpne_sc_h.....	988
__builtin_riscv_cv_simd_abs_b.....	983	__builtin_riscv_cv_simd_cplxconj.....	992
__builtin_riscv_cv_simd_abs_h.....	983	__builtin_riscv_cv_simd_cplxmul_i.....	992
__builtin_riscv_cv_simd_add_b.....	976	__builtin_riscv_cv_simd_cplxmul_r.....	992
__builtin_riscv_cv_simd_add_h.....	976, 993	__builtin_riscv_cv_simd_dotsp_b.....	984
__builtin_riscv_cv_simd_add_sc_b.....	977	__builtin_riscv_cv_simd_dotsp_h.....	984
__builtin_riscv_cv_simd_add_sc_h.....	976, 977	__builtin_riscv_cv_simd_dotsp_sc_b.....	984
__builtin_riscv_cv_simd_and_b.....	982	__builtin_riscv_cv_simd_dotsp_sc_h.....	984
__builtin_riscv_cv_simd_and_h.....	982	__builtin_riscv_cv_simd_dotup_b.....	983
__builtin_riscv_cv_simd_and_sc_b.....	983	__builtin_riscv_cv_simd_dotup_h.....	983
__builtin_riscv_cv_simd_and_sc_h.....	982, 983	__builtin_riscv_cv_simd_dotup_sc_b.....	983
__builtin_riscv_cv_simd_avg_b.....	977	__builtin_riscv_cv_simd_dotup_sc_h.....	983
__builtin_riscv_cv_simd_avg_h.....	977	__builtin_riscv_cv_simd_dotusp_b.....	983
__builtin_riscv_cv_simd_avg_sc_b.....	978	__builtin_riscv_cv_simd_dotusp_h.....	983

__builtin_riscv_cv_simd_dotusp_sc_b.....	984	__builtin_riscv_cv_simd_sll_h.....	981
__builtin_riscv_cv_simd_dotusp_sc_h.....	984	__builtin_riscv_cv_simd_sll_sc_b.....	981
__builtin_riscv_cv_simd_extract_b.....	986	__builtin_riscv_cv_simd_sll_sc_h.....	981
__builtin_riscv_cv_simd_extract_h.....	986	__builtin_riscv_cv_simd_sra_b.....	981
__builtin_riscv_cv_simd_extractu_b.....	986	__builtin_riscv_cv_simd_sra_h.....	980
__builtin_riscv_cv_simd_extractu_h.....	986	__builtin_riscv_cv_simd_sra_sc_b.....	981
__builtin_riscv_cv_simd_insert_b.....	986	__builtin_riscv_cv_simd_sra_sc_h.....	981
__builtin_riscv_cv_simd_insert_h.....	986	__builtin_riscv_cv_simd_srl_b.....	980
__builtin_riscv_cv_simd_max_b.....	979	__builtin_riscv_cv_simd_srl_h.....	980
__builtin_riscv_cv_simd_max_h.....	979	__builtin_riscv_cv_simd_srl_sc_b.....	980
__builtin_riscv_cv_simd_max_sc_b.....	979	__builtin_riscv_cv_simd_srl_sc_h.....	980
__builtin_riscv_cv_simd_max_sc_h.....	979	__builtin_riscv_cv_simd_sub_b.....	977
__builtin_riscv_cv_simd_maxu_b.....	980	__builtin_riscv_cv_simd_sub_h.....	977, 993
__builtin_riscv_cv_simd_maxu_h.....	980	__builtin_riscv_cv_simd_sub_sc_b.....	977
__builtin_riscv_cv_simd_maxu_sc_b.....	980	__builtin_riscv_cv_simd_sub_sc_h.....	977
__builtin_riscv_cv_simd_maxu_sc_h.....	980	__builtin_riscv_cv_simd_subrotmj.....	992, 993
__builtin_riscv_cv_simd_min_b.....	978	__builtin_riscv_cv_simd_xor_b.....	982
__builtin_riscv_cv_simd_min_h.....	978	__builtin_riscv_cv_simd_xor_h.....	982
__builtin_riscv_cv_simd_min_sc_b.....	978, 979	__builtin_riscv_cv_simd_xor_sc_b.....	982
__builtin_riscv_cv_simd_min_sc_h.....	978	__builtin_riscv_cv_simd_xor_sc_h.....	982
__builtin_riscv_cv_simd_minu_b.....	979	__builtin_riscv_pause.....	972
__builtin_riscv_cv_simd_minu_h.....	979	__builtin_rx_brk.....	993
__builtin_riscv_cv_simd_minu_sc_b.....	979	__builtin_rx_clrpsw.....	993
__builtin_riscv_cv_simd_minu_sc_h.....	979	__builtin_rx_int.....	993
__builtin_riscv_cv_simd_or_b.....	981	__builtin_rx_machi.....	994
__builtin_riscv_cv_simd_or_h.....	981	__builtin_rx_maclo.....	994
__builtin_riscv_cv_simd_or_sc_b.....	982	__builtin_rx_mulhi.....	994
__builtin_riscv_cv_simd_or_sc_h.....	982	__builtin_rx_mullo.....	994
__builtin_riscv_cv_simd_packhi_b.....	987	__builtin_rx_mvfacbi.....	994
__builtin_riscv_cv_simd_packhi_h.....	987	__builtin_rx_mvfacmi.....	994
__builtin_riscv_cv_simd_packlo_b.....	987	__builtin_rx_mvfc.....	994
__builtin_riscv_cv_simd_packlo_h.....	987	__builtin_rx_mvtachi.....	994
__builtin_riscv_cv_simd_sdotsp_b.....	985	__builtin_rx_mvtaclo.....	994
__builtin_riscv_cv_simd_sdotsp_h.....	985	__builtin_rx_mvtc.....	994
__builtin_riscv_cv_simd_sdotsp_sc_b.....	986	__builtin_rx_mvtipl.....	994
__builtin_riscv_cv_simd_sdotsp_sc_h.....	985	__builtin_rx_racw.....	994
__builtin_riscv_cv_simd_sdotup_b.....	984	__builtin_rx_revw.....	994
__builtin_riscv_cv_simd_sdotup_h.....	984	__builtin_rx_rmpa.....	995
__builtin_riscv_cv_simd_sdotup_sc_b.....	985	__builtin_rx_round.....	995
__builtin_riscv_cv_simd_sdotup_sc_h.....	984, 985	__builtin_rx_sat.....	995
__builtin_riscv_cv_simd_sdotusp_b.....	985	__builtin_rx_setpsw.....	995
__builtin_riscv_cv_simd_sdotusp_h.....	985	__builtin_rx_wait.....	995
__builtin_riscv_cv_simd_sdotusp_sc_b.....	985	__builtin_sadd_overflow.....	809
__builtin_riscv_cv_simd_sdotusp_sc_h.....	985	__builtin_saddl_overflow.....	809
__builtin_riscv_cv_simd_shuffle_b.....	986	__builtin_saddll_overflow.....	809
__builtin_riscv_cv_simd_shuffle_h.....	986	__builtin_set_thread_pointer.....	997
__builtin_riscv_cv_simd_shuffle_sci_h.....	986	__builtin_setjmp.....	814
__builtin_riscv_cv_simd_shuffle2_b.....	987	__builtin_sh_get_fpscr.....	997
__builtin_riscv_cv_simd_shuffle2_h.....	987	__builtin_sh_set_fpscr.....	997
__builtin_riscv_cv_simd_shufflei0_sci_b.....	986	__builtin_shuffle.....	820
__builtin_riscv_cv_simd_shufflei1_sci_b.....	986	__builtin_shufflevector.....	821
__builtin_riscv_cv_simd_shufflei2_sci_b.....	987	__builtin_smul_overflow.....	810
__builtin_riscv_cv_simd_shufflei3_sci_b.....	987	__builtin_smull_overflow.....	810
__builtin_riscv_cv_simd_sll_b.....	981	__builtin_smulll_overflow.....	810
		__builtin_speculation_safe_value.....	797, 833
		__builtin_sqrtf128_round_to_odd.....	928
		__builtin_ssub_overflow.....	809
		__builtin_ssubl_overflow.....	809

__builtin_ssubll_overflow.....	809	__flash2 AVR Named Address Spaces.....	590
__builtin_stack_address.....	817	__flash3 AVR Named Address Spaces.....	590
__builtin_stdcl_bit_ceil.....	804	__flash4 AVR Named Address Spaces.....	590
__builtin_stdcl_bit_floor.....	804	__flash5 AVR Named Address Spaces.....	590
__builtin_stdcl_bit_width.....	804	__flashx AVR Named Address Spaces.....	590
__builtin_stdcl_count_ones.....	804	__float128 data type.....	577
__builtin_stdcl_count_zeros.....	804	__float80 data type.....	577
__builtin_stdcl_first_leading_one.....	804	__force_132 Xtensa Named Address Spaces...	592
__builtin_stdcl_first_leading_zero.....	804	__fp16 data type.....	578
__builtin_stdcl_first_trailing_one.....	805	__func__ identifier.....	793
__builtin_stdcl_first_trailing_zero.....	805	__FUNCTION__ identifier.....	793
__builtin_stdcl_has_single_bit.....	805	__halt.....	972
__builtin_stdcl_leading_ones.....	805	__has_nothrow_assign.....	1042
__builtin_stdcl_leading_zeros.....	805	__has_nothrow_constructor.....	1042
__builtin_stdcl_rotate_left.....	805	__has_nothrow_copy.....	1042
__builtin_stdcl_rotate_right.....	806	__has_trivial_assign.....	1042
__builtin_stdcl_trailing_ones.....	805	__has_trivial_constructor.....	1042
__builtin_stdcl_trailing_zeros.....	805	__has_trivial_copy.....	1042
__builtin_structured_binding_size.....	1044	__has_trivial_destructor.....	1043
__builtin_sub_overflow.....	809	__has_virtual_destructor.....	1043
__builtin_sub_overflow_p.....	810	__ibm128 data type.....	577
__builtin_subc.....	811	__imag__ keyword.....	576
__builtin_subcl.....	811	__int128 data types.....	575
__builtin_subc11.....	811	__integer_pack.....	1044
__builtin_subf128_round_to_odd.....	928	__is_abstract.....	1043
__builtin_tabort.....	996	__is_aggregate.....	1043
__builtin_tbegin.....	995	__is_base_of.....	1043
__builtin_tbegin_nofloat.....	996	__is_class.....	1043
__builtin_tbegin_retry.....	996	__is_empty.....	1043
__builtin_tbegin_retry_nofloat.....	996	__is_enum.....	1044
__builtin_tbegininc.....	996	__is_final.....	1044
__builtin_tend.....	996	__is_literal_type.....	1044
__builtin_tgmath.....	836	__is_pod.....	1044
__builtin_thread_pointer.....	972, 997	__is_polymorphic.....	1044
__builtin_trap.....	839	__is_same.....	1044
__builtin_truncf128_round_to_odd.....	929	__is_standard_layout.....	1044
__builtin_tx_assist.....	996	__is_trivial.....	1044
__builtin_tx_nesting_depth.....	997	__is_union.....	1044
__builtin_types_compatible_p.....	834	__lmbd.....	972
__builtin_uadd_overflow.....	809	__memx AVR Named Address Spaces.....	590
__builtin_uaddl_overflow.....	809	__PRETTY_FUNCTION__ identifier.....	793
__builtin_uaddll_overflow.....	809	__real__ keyword.....	576
__builtin_umul_overflow.....	810	__regio_symbol PRU Named Address Spaces.....	592
__builtin_umull_overflow.....	810	__seg_fs x86 named address space.....	592
__builtin_umulll_overflow.....	810	__seg_gs x86 named address space.....	592
__builtin_unreachable.....	839	__STDC_HOSTED__.....	3
__builtin_usub_overflow.....	809	__sync builtins.....	827
__builtin_usubl_overflow.....	809	__sync_add_and_fetch.....	828
__builtin_usubll_overflow.....	810	__sync_and_and_fetch.....	828
__builtin_va_arg_pack.....	815	__sync_bool_compare_and_swap.....	829
__builtin_va_arg_pack_len.....	815	__sync_fetch_and_add.....	828
__complex__ keyword.....	575	__sync_fetch_and_and.....	828
__declspec.....	672	__sync_fetch_and_nand.....	828
__delay_cycles.....	972	__sync_fetch_and_or.....	828
__extension__.....	792	__sync_fetch_and_sub.....	828
__far RL78 Named Address Spaces.....	592	__sync_fetch_and_xor.....	828
__flash AVR Named Address Spaces.....	590	__sync_lock_release.....	829
__flash1 AVR Named Address Spaces.....	590		

<code>__sync_lock_test_and_set</code>	829
<code>__sync_nand_and_fetch</code>	829
<code>__sync_or_and_fetch</code>	828
<code>__sync_sub_and_fetch</code>	828
<code>__sync_synchronize</code>	829
<code>__sync_val_compare_and_swap</code>	829
<code>__sync_xor_and_fetch</code>	828
<code>__thread</code>	715
<code>__underlying_type</code>	1044
<code>_Accum</code> data type.....	580
<code>_Bool</code> keyword.....	789
<code>_Complex</code> keyword.....	575
<code>_Countof</code>	787
<code>_Decimal128</code> data type.....	579
<code>_Decimal32</code> data type.....	579
<code>_Decimal64</code> data type.....	579
<code>_exit</code>	797
<code>_Exit</code>	797
<code>_Float16</code> data type.....	578
<code>_Floatn</code> data types.....	577
<code>_Floatnx</code> data types.....	577
<code>_Fract</code> data type.....	580
<code>_get_ssp</code>	1029
<code>_HTM_FIRST_USER_ABORT_CODE</code>	996
<code>_HTM_TBEGIN_INDETERMINATE</code>	995
<code>_HTM_TBEGIN_PERSISTENT</code>	996
<code>_HTM_TBEGIN_STARTED</code>	995
<code>_HTM_TBEGIN_TRANSIENT</code>	996
<code>_inc_ssp</code>	1029
<code>_Maxof</code>	787
<code>_Minof</code>	787
<code>_Sat</code> data type.....	580
<code>_xabort</code>	1029
<code>_XABORT_CAPACITY</code>	1028
<code>_XABORT_CONFLICT</code>	1028
<code>_XABORT_DEBUG</code>	1028
<code>_XABORT_EXPLICIT</code>	1028
<code>_XABORT_NESTED</code>	1028
<code>_XABORT_RETRY</code>	1028
<code>_xbegin</code>	1028
<code>_xend</code>	1028
<code>_xtest</code>	1029

0

<code>'0'</code> in constraint.....	746
-------------------------------------	-----

A

<code>AArch64</code> code generation attributes.....	649
<code>AArch64</code> Options.....	317
<code>ABI</code>	1063
<code>abort</code>	797
<code>abs</code>	797
accessing volatiles.....	720, 1031
<code>acos</code>	797
<code>acosf</code>	797
<code>acosh</code>	797

<code>acoshf</code>	797
<code>acoshl</code>	797
<code>acosl</code>	797
<code>acospi</code>	797
<code>acospif</code>	797
<code>acospil</code>	797
<code>Ada</code>	1
additional floating types.....	577
address constraints.....	747
address of a label.....	783
<code>address_operand</code>	747
alignment.....	788
alignment of allocated data.....	599
alignment of data.....	601
alignment of functions and data.....	597
alignment of structure fields.....	627, 645
<code>alloca</code>	797
<code>alloca</code> vs variable-length arrays.....	581
Allow nesting in an interrupt handler on the Blackfin processor.....	662
alternate keywords.....	792
<code>AMD GCN</code> Options.....	331
<code>AMD1</code>	3
<code>ANSI C</code>	3
<code>ANSI C</code> standard.....	3
<code>ANSI C89</code>	3
<code>ANSI</code> support.....	45
<code>ANSI X3.159-1989</code>	3
apostrophes.....	1104
application binary interface.....	1063
<code>ARC</code> options.....	333
architecture-specific options.....	316
<code>ARM</code> [Annotated C++ Reference Manual]....	1045
<code>ARM</code> options.....	341
arrays of length zero.....	582
arrays of variable length.....	581
arrays, non-lvalue.....	586
<code>asin</code>	797
<code>asinf</code>	797
<code>asinh</code>	797
<code>asinhf</code>	797
<code>asinhhl</code>	797
<code>asinl</code>	797
<code>asinpil</code>	797
<code>asinpif</code>	797
<code>asinpil</code>	797
<code>asm</code> assembler template.....	727
<code>asm</code> clobbers.....	734
<code>asm</code> constraints.....	744
<code>asm</code> expressions.....	733
<code>asm</code> flag output operands.....	731
<code>asm</code> goto labels.....	737
<code>asm inline</code>	779
<code>asm</code> input operands.....	733
<code>asm</code> keyword.....	721
<code>asm</code> output operands.....	728
<code>asm</code> scratch registers.....	734
<code>asm</code> volatile.....	725

assembler names for identifiers	773
assembly code, invalid	1117
assembly language in C	721
assembly language in C, basic	721
assembly language in C, extended	723
atan	797
atan2	797
atan2f	797
atan2l	797
atan2pi	797
atan2pif	797
atan2pil	797
atanf	797
atanh	797
atanhf	797
atanhl	797
atanl	797
atanpi	797
atanpif	797
atanpil	797
atomic memory access builtins	822
attribute syntax	703
attributes	593
autoincrement/decrement addressing	745
automatic inline for C++ member fns	719
AVR Options	359

B

Backwards Compatibility	1045
base class members	1109
base type of an enum	788
basic asm	721
bcmp	797
binary compatibility	1063
Binary constants using the '0b' prefix	791
bit operation builtins	802
Blackfin Options	372
boolean type	789
bound pointer to member function	1036
break handler functions	671
bug criteria	1117
bugs	1117
bugs, known	1101
Built-in Functions	797
built-in functions	48
built-in library functions	797
built-in numeric functions	799
builtins for arithmetic overflow checking	809
builtins for atomic memory access	822
builtins for C++ new and delete operators	832
builtins for stack allocation	812
byte order	630
byte-swapping builtins	806
bzero	797

C

c++	44
C compilation options	9
C intermediate output, nonexistent	1
C language extensions	575
C language, traditional	273
C library function builtins	797
C standard	3
C standard attributes	703
C standards	3
C++	1
C++ comments	791
C++ Compiled Module Interface	561
C++ interface and implementation headers	1033
C++ language extensions	1031
C++ member fns, automatically inline	719
C++ misunderstandings	1108
C++ Module Mapper	559
C++ Module Preprocessing	560
C++ options, command-line	52
C++ pragmas, effect on inlining	1034
C++ source file suffixes	44
C++ standard attributes	703
C++ static data, declaring and defining	1108
C++11 memory model	822
C-SKY Options	377
C_INCLUDE_PATH	554
C11	3
C17	3
C1X	3
C23	3
C2X	3
C2Y	3
C6X Options	374
C89	3
C90	3
C94	3
C95	3
C99	3
C9X	3
cabs	797
cabsf	797
cabsl	797
cacos	797
cacosf	797
cacosh	797
cacoshf	797
cacoshl	797
cacosl	797
calling functions through the function vector on SH2A	686
calloc	797
carg	797
cargf	797
cargl	797
case labels in initializers	588
case ranges	790
casin	797

<code>casinf</code>	797	compiler options, Objective-C and	
<code>casinh</code>	797	Objective-C++	82
<code>casinhf</code>	797	compiler version, specifying	9
<code>casinhl</code>	797	<code>COMPILER_PATH</code>	553
<code>casinl</code>	797	complex conjugation	576
cast to a union	585	complex numbers	575
<code>catan</code>	797	compound literals	587
<code>catanf</code>	797	computed gotos	783
<code>catanh</code>	797	conditional expressions, extensions	790
<code>catanhf</code>	797	conflicting types	1107
<code>catanhl</code>	797	<code>conj</code>	797
<code>catanl</code>	797	<code>conjf</code>	797
<code>cbrt</code>	797	<code>conjl</code>	797
<code>cbrtf</code>	797	<code>const</code> applied to function	795
<code>cbrtl</code>	797	<code>const</code> qualifier	795
<code>ccos</code>	797	constants in constraints	745
<code>ccosf</code>	797	constraint modifier characters	748
<code>ccosh</code>	797	constraint, matching	746
<code>ccoshf</code>	797	constraints, <code>asm</code>	744
<code>ccoshl</code>	797	constraints, machine specific	749
<code>ccosl</code>	797	constructing calls	814
<code>ceil</code>	797	constructor expressions	587
<code>ceilf</code>	797	contributors	1147
<code>ceill</code>	797	<code>copysign</code>	797
<code>cexp</code>	797	<code>copysignf</code>	797
<code>cexpf</code>	797	<code>copysignl</code>	797
<code>cexpl</code>	797	core dump	1117
char type signedness	51	<code>cos</code>	797
character arrays, non-null-terminated	623	<code>cosf</code>	797
character arrays, null-terminated	625	<code>cosh</code>	797
character set, execution	271	<code>coshf</code>	797
character set, input	272	<code>coshl</code>	797
character set, input normalization	162	<code>cosl</code>	797
character set, wide execution	272	<code>cospi</code>	797
<code>cimag</code>	797	<code>cospif</code>	797
<code>cimagf</code>	797	<code>cospil</code>	797
<code>cimagl</code>	797	<code>CPATH</code>	554
cleanup functions	602	<code>CPLUS_INCLUDE_PATH</code>	554
<code>clog</code>	797	<code>cpow</code>	797
<code>clog10</code>	797	<code>cpowf</code>	797
<code>clog10f</code>	797	<code>cpowl</code>	797
<code>clog10l</code>	797	<code>cproj</code>	797
<code>clogf</code>	797	<code>cprojf</code>	797
<code>clogl</code>	797	<code>cprojl</code>	797
COBOL	1, 7	CRC builtins	807
code generation conventions	286	<code>creal</code>	797
code, mixed with declarations	791	<code>crealf</code>	797
cold functions and code regions	602	<code>creall</code>	797
command options	9	CRIS Options	375
comments, C++ style	791	cross compiling	9
common storage	603	<code>csin</code>	797
comparison of signed and unsigned		<code>csinf</code>	797
values, warning	155	<code>csinh</code>	797
compilation statistics	297	<code>csinhf</code>	797
compiler bugs, reporting	1117	<code>csinhl</code>	797
compiler compared to C++ preprocessor	1	<code>csinl</code>	797
compiler options, C++	52	<code>csqrt</code>	797
		<code>csqrtf</code>	797

<code>csqrtl</code>	797
<code>ctan</code>	797
<code>ctanf</code>	797
<code>ctanh</code>	797
<code>ctanhf</code>	797
<code>ctanhl</code>	797
<code>ctanl</code>	797
<code>CXX_MODULE_MAPPER</code> environment variable	61
Cygwin and MinGW Options	380

D

<code>d</code> floating point suffix	579
<code>D</code> floating point suffix	579
Darwin options	381
<code>dcgettext</code>	797
<code>dd</code> floating point suffix	579
<code>DD</code> floating point suffix	579
deallocating variable length arrays	581
debug dump options	297
debugging GCC	297
debugging information options	189
decimal floating types	579
declaration scope	1104
declarations inside expressions	780
declarations, mixed with code	791
declaring attributes	593
declaring attributes of functions	593
declaring static data in C++	1108
defining static data in C++	1108
dependencies for make as output	554, 555
dependencies, <code>make</code>	268
<code>DEPENDENCIES_OUTPUT</code>	555
dependent name lookup	1109
deprecated entities	607
designated initializers	588
designator lists	589
designators	588
developer options	297
<code>df</code> floating point suffix	579
<code>DF</code> floating point suffix	579
<code>dgettext</code>	797
diagnostic messages	88
diagnostic output formats, <code>sarif-file</code>	97
diagnostic output formats, <code>sarif-stderr</code>	97
dialect options	45
<code>diff-delete</code> GCC_COLORS capability	90
<code>diff-filename</code> GCC_COLORS capability	90
<code>diff-hunk</code> GCC_COLORS capability	90
<code>diff-insert</code> GCC_COLORS capability	90
digits in constraint	746
directory options	282
<code>dl</code> floating point suffix	579
<code>DL</code> floating point suffix	579
dollar signs in identifier names	792
double-word arithmetic	575
downward funargs	784
<code>drem</code>	797

<code>dremf</code>	797
<code>dreml</code>	797
dump options	297
D	1

E

‘E’ in constraint	746
earlyclobber operand	748
eBPF Options	391
eight-bit data on the H8/300, H8/300H, and H8S	665
<code>EIND</code>	366
empty structures	584
endianness	630
<code>enum</code> extensions	788
environment variables	552
<code>erf</code>	797
<code>erfc</code>	797
<code>erfcf</code>	797
<code>erfcl</code>	797
<code>erff</code>	797
<code>erfl</code>	797
<code>error</code> GCC_COLORS capability	89
error messages	1115
escaped newlines	791
exception handler functions, Blackfin	661
exception handler functions, NDS32	678
<code>exit</code>	797
<code>exp</code>	797
<code>exp10</code>	797
<code>exp10f</code>	797
<code>exp10l</code>	797
<code>exp2</code>	797
<code>exp2f</code>	797
<code>exp2l</code>	797
<code>EXPERIMENTAL_SARIF_SOCKET</code>	554
<code>expf</code>	797
<code>expl</code>	797
explicit register variables	774
<code>expm1</code>	797
<code>expm1f</code>	797
<code>expm1l</code>	797
expressions containing statements	780
expressions, constructor	587
extended <code>asm</code>	723
extensible constraints	747
extensions, <code>?:</code>	790
extensions, C language	575
extensions, C++ language	1031
external declaration scope	1104
extra NOP instructions at the function entry point	628

F

'F' in constraint	746	function inlining	600, 610, 612
<code>fabs</code>	797	function inlining, suppressing	621
<code>fabsf</code>	797	function instrumentation	620
<code>fabsl</code>	797	function multiversioning	639
fallthrough in switch statements	608	function pointers, arithmetic	794
fatal signal	1117	function prototype declarations	794
<code>fdim</code>	797	function versions	1040
<code>fdimf</code>	797	function, size of pointer to	794
<code>fdiml</code>	797	functions in arbitrary sections	631
FDL, GNU Free Documentation License	1139	functions that are called on	
<code>ffs</code>	797	program startup or exit	604
file name suffix	34	functions that are compiled with	
file names	276	specific optimizations	626
fixed-point types	580	functions that are dynamically resolved	613
<code>fixit-delete</code> GCC_COLORS capability	89	functions that are expected to	
<code>fixit-insert</code> GCC_COLORS capability	89	throw exceptions	608
<code>flag_enum</code> type attribute	609	functions that are passed arguments in	
flexible array members	582, 605, 632	registers on x86-32	689
<code>float</code> as function value type	1105	functions that are supposed to be	
floating point precision	1107	optimized away	608
floating-point format builtins	799	functions that behave like <code>malloc</code>	616
<code>floor</code>	797	functions that do not have	
<code>floorf</code>	797	prologue/epilogue code	619
<code>floorl</code>	797	functions that do not throw exceptions	625
<code>fma</code>	797	functions that handle interrupts	615
<code>fmaf</code>	797	functions that have a	
<code>fmal</code>	797	null-terminated argument list	631
<code>fmax</code>	797	functions that have format	
<code>fmaxf</code>	797	string arguments	610, 611
<code>fmaxl</code>	797	functions that have no side effects ...	603, 628, 629, 641
<code>fmin</code>	797	functions that have non-null	
<code>fminf</code>	797	pointer arguments	622, 623
<code>fminl</code>	797	functions that have patchable entries	628
<code>fmod</code>	797	functions that have SIMD clones	632
<code>fmodf</code>	797	functions that have tainted arguments	638
<code>fmodl</code>	797	functions that never return	624
<code>fnname</code> GCC_COLORS capability	89	functions that pop the argument	
Fortran	1	stack on x86-32	688, 690
forward declaration of an <code>enum</code>	788	functions that restrict access to	
forwarding calls	814	pointer arguments	595
<code>fprintf</code>	797	functions that return more than once	630
<code>fprintf_unlocked</code>	797	functions that return via tail call	618
<code>fputs</code>	797	functions that should have stack protection ...	632
<code>fputs_unlocked</code>	797	functions that should not be instrumented ...	620
FR30 Options	393	functions that should not be sanitized	620, 621
<code>free</code>	797	functions that should not have	
freestanding environment	3	stack limit checking	621
freestanding implementation	3	functions that should not have	
<code>frexp</code>	797	stack protection	621
<code>frexpf</code>	797	functions that take file descriptor arguments ..	609
<code>frexpl</code>	797	functions whose return value must be used	646
FRV Options	393	functions with <code>printf</code> , <code>scanf</code> , <code>strftime</code> or	
<code>fscanf</code>	797	<code>strfmon</code> style arguments	610
<code>fscanf</code> , and constant strings	1103		
FT32 Options	396		
function addressability on the M32R/D	670		
function attributes	593		

G

'g' in constraint	746
g++	44
'G' in constraint	746
G++	1
gamma	797
gamma_r	797
gammaf	797
gammaf_r	797
gammal	797
gammal_r	797
GCC	1
GCC command options	9
GCC_COLORS environment variable	88
GCC_COMPARE_DEBUG	553
GCC_DIAGNOSTICS_LOG	552
GCC_EXEC_PREFIX	553
GCC_EXTRA_DIAGNOSTIC_OUTPUT	553
GCC_URLS environment variable	90
gcov	246
gettext	797
global offset table	291
global register after <code>longjmp</code>	775
global register variables	774
GNAT	1
GNU attribute syntax	703
GNU attributes	593
GNU C Compiler	1
GNU Compiler Collection	1
Go	1
goto with computed label	783
gprof	245
grouping options	9

H

'H' in constraint	746
half-precision floating point	578
hard registers in constraint	745
hardened boolean types	612
hardware-specific options	316
hex floats	791
highlight, color	88
highlight-a GCC_COLORS capability	90
highlight-b GCC_COLORS capability	90
hk fixed-suffix	580
HK fixed-suffix	580
hosted environment	3, 49
hosted implementation	3
hot functions and code regions	602
HPPA Options	397
hr fixed-suffix	580
HR fixed-suffix	580
huge value builtins	799
hypot	797
hypotf	797
hypotl	797

I

'i' in constraint	745
'I' in constraint	746
IA-64 Options	401
IA64 atomic memory access builtins	827
IBM RS/6000 and PowerPC Options	469
identifier names, dollar signs in	792
identifiers, names in assembler code	773
ilogb	797
ilogbf	797
ilogbl	797
imaxabs	797
implementation-defined	
behavior, C language	563
implementation-defined behavior,	
C++ language	573
implied <code>#pragma implementation</code>	1034
incompatibilities of GCC	1103
incomplete <code>enum</code> types	788
increment operators	1117
index	797
indirect calls, ARC	654
indirect calls, ARM	655
indirect calls, Blackfin	662
indirect calls, Epiphany	664
indirect calls, MIPS	675
indirect calls, PowerPC	679
indirect functions	613
infinity builtins	799
initializations in expressions	587
initializers with labeled elements	588
initializers, non-constant	586
inline assembly language	721
inline automatic for C++ member fns	719
inline functions	718
inline functions, omission of	719
inlining	600, 610, 612
inlining and C++ pragmas	1034
inlining, suppressing	621
installation trouble	1101
instrumentation options	245
integer arithmetic overflow checking builtins	809
integrating function code	718
interface and implementation headers, C++	1033
intermediate C version, nonexistent	1
interrupt handlers	615
invalid assembly code	1117
invalid GCC_COLORS capability	90
invalid input	1117
invoking g++	44
isalnum	797
isalpha	797
isascii	797
isblank	797
isctrl	797
isdigit	797
isgraph	797
islower	797

ISO 9899	3
ISO C	3
ISO C standard	3
ISO C11	3
ISO C17	3
ISO C1X	3
ISO C23	3
ISO C2X	3
ISO C2Y	3
ISO C90	3
ISO C94	3
ISO C95	3
ISO C99	3
ISO C9X	3
ISO support	45
ISO/IEC 9899	3
isprint	797
ispunct	797
isspace	797
isupper	797
iswalnum	797
iswalpha	797
iswblank	797
iswcntrl	797
iswdigit	797
iswgraph	797
iswlower	797
iswprint	797
iswpunct	797
iswspace	797
iswupper	797
iswxdigit	797
isxdigit	797

J

j0	797
j0f	797
j0l	797
j1	797
j1f	797
j1l	797
jn	797
jnf	797
jnl	797

K

k fixed-suffix	580
K fixed-suffix	580
keywords, alternate	792
known causes of trouble	1101

L

labeled elements in initializers	588
labels as values	783
labs	797
language dialect options	45
LANG	552
LC_ALL	552
LC_CTYPE	552
LC_MESSAGES	552
ldexp	797
ldexpf	797
ldexpl	797
legacy builtins for atomic memory access	827
length-zero arrays	582
lgamma	797
lgamma_r	797
lgammaf	797
lgammaf_r	797
lgammal	797
lgammal_r	797
Libraries	277
library function builtins	797
LIBRARY_PATH	553
link options	276
linker garbage collection	629
linker script	281
linker visibility	643
lk fixed-suffix	580
LK fixed-suffix	580
LL integer suffix	575
llabs	797
llk fixed-suffix	580
LLK fixed-suffix	580
llr fixed-suffix	580
LLR fixed-suffix	580
llrint	797
llrintf	797
llrintl	797
llround	797
llroundf	797
llroundl	797
LM32 options	404
load address instruction	747
local labels	782
local variables in macros	786
local variables, specifying registers	775
locale	552
locus GCC_COLORS capability	89
log	797
log10	797
log10f	797
log10l	797
log1p	797
log1pf	797
log1pl	797
log2	797
log2f	797
log2l	797

<code>logb</code>	797
<code>logbf</code>	797
<code>logbl</code>	797
<code>logf</code>	797
<code>logl</code>	797
<code>long long</code> data types.....	575
<code>longjmp</code>	775
<code>longjmp</code> incompatibilities.....	1103
<code>longjmp</code> warnings.....	129
LoongArch Options.....	405
lr fixed-suffix.....	580
LR fixed-suffix.....	580
<code>lrint</code>	797
<code>lrintf</code>	797
<code>lrintl</code>	797
<code>lround</code>	797
<code>lroundf</code>	797
<code>lroundl</code>	797
LynxOS Options.....	411

M

‘m’ in constraint.....	745
M32R/D options.....	411
M680x0 options.....	412
machine specific constraints.....	749
machine-specific options.....	316
macro with variable arguments.....	789
macros, inline alternative.....	718
macros, local labels.....	782
macros, local variables in.....	786
macros, statements in expressions.....	780
macros, types of arguments.....	786
<code>make</code>	268
<code>malloc</code>	797
matching constraint.....	746
MCore options.....	418
member fns, automatically <code>inline</code>	719
<code>memchr</code>	797
<code>memcmp</code>	797
<code>memcpy</code>	797
memory references in constraints.....	745
<code>mempcpy</code>	797
<code>memset</code>	797
Mercury.....	1
message formatting.....	88
messages, warning.....	101
messages, warning and error.....	1115
MicroBlaze Options.....	419
Microsoft struct layout.....	710
middle-operands, omitted.....	790
MIPS options.....	420
misunderstandings in C++.....	1108
mixed declarations and code.....	791
mixing assembly language and C.....	721
<code>mktemp</code> , and constant strings.....	1103
MMIX Options.....	435
MN10300 options.....	437

<code>modf</code>	797
<code>modff</code>	797
<code>modfl</code>	797
modifiers in constraints.....	748
Moxie Options.....	438
ms_struct pragma.....	710
MSP430 Options.....	438
multiple alternative constraints.....	747
multiprecision arithmetic.....	575

N

‘n’ in constraint.....	746
naked functions.....	619
Named Address Spaces.....	589
names used in assembler code.....	773
naming convention, implementation headers..	1034
NaN builtins.....	799
NDS32 Options.....	441
<code>nearbyint</code>	797
<code>nearbyintf</code>	797
<code>nearbyintl</code>	797
nested functions.....	784
<code>new</code> and <code>delete</code> builtins.....	832
newlines (escaped).....	791
<code>nextafter</code>	797
<code>nextafterf</code>	797
<code>nextafterl</code>	797
<code>nexttoward</code>	797
<code>nexttowardf</code>	797
<code>nexttowardl</code>	797
NFC.....	162
NFKC.....	162
NMI handler functions on the Blackfin processor.....	662
<code>no_profile_instrument_function</code> function attribute.....	620
<code>no_sanitize</code> function attribute.....	620
<code>no_sanitize_coverage</code> function attribute.....	620
non-constant initializers.....	586
non-static inline function.....	719
nonlocal gotos.....	813
nonstring character arrays.....	623
<code>note GCC_COLORS</code> capability.....	89
<code>number of elements</code>	787
numeric builtins.....	799
Nvidia PTX options.....	444
nvptx options.....	444

O

‘o’ in constraint	745
OBJC_INCLUDE_PATH	554
Objective-C	1, 6
Objective-C and Objective-C++	
options, command-line	82
Objective-C++	1, 6
Offloading options	86
Offloading targets	86
offsettable address	745
old-style function definitions	794
omitted middle-operands	790
opaque <code>enum</code> types	788
open coding	718
OpenACC accelerator programming	87, 163
OpenACC extension support	718
OpenACC offloading options	86
OpenACC offloading targets	86
OpenACC options	86
OpenMP extension support	717
OpenMP offloading options	86
OpenMP offloading targets	86
OpenMP options	86
OpenMP parallel	87
OpenMP SIMD	87
OpenMP target SIMD clone	87
OpenRISC Options	445
operand constraints, <code>asm</code>	744
operating-system-specific options	316
optimization, per-function	626
optimize options	196
Options for Cygwin and MinGW	380
options to control diagnostics formatting	88
options to control warnings	101
options, C++	52
options, code generation	286
options, debugging	189
options, dialect	45
options, directory search	282
options, GCC command	9
options, grouping	9
options, linking	276
options, Objective-C and Objective-C++	82
options, optimization	196
options, order	9
options, Picolibc	447
options, preprocessor	267
options, profiling	245
options, program instrumentation	245
options, run-time error checking	245
order of evaluation, side effects	1114
order of options	9
other register constraints	747
output file option	37
overflow checking builtins	809
overloaded virtual function, warning	75

P

‘p’ in constraint	747
pack pragma	709
packed data	627
parameter forward declaration	582
path GCC_COLORS capability	89
PDP-11 Options	447
persistent data	628
Picolibc options	447
PIC	291
pmf	1036
pointer aliasing	617
pointer arguments	604
pointer arguments in variadic functions	794
pointer to member function	1036
pointers to arrays	795
portions of temporary objects, pointers to	1110
pow	797
pow10	797
pow10f	797
pow10l	797
PowerPC options	448
powf	797
powl	797
pragma GCC ivdep	714
pragma GCC novector	714
pragma GCC optimize	713
pragma GCC pop_options	713
pragma GCC push_options	713
pragma GCC reset_options	713
pragma GCC target	713
pragma GCC unroll <i>n</i>	714
pragma, align	708
pragma, ctable_entry	707
pragma, diagnostic	711, 712
pragma, fini	708
pragma, init	708
pragma, long_calls	707
pragma, long_calls_off	707
pragma, longcall	707
pragma, mark	708
pragma, ms_struct	710
pragma, no_long_calls	707
pragma, options align	708
pragma, pack	709
pragma, pop_macro	712
pragma, push_macro	712
pragma, redefine_extname	709
pragma, scalar_storage_order	710
pragma, segment	708
pragma, unused	708
pragma, visibility	712
pragma, weak	710
pragmas	706
pragmas in C++, effect on inlining	1034
pragmas, interface and implementation	1033
pragmas, warning of unknown	130
precompiled headers	555

preprocessing numbers	1105
preprocessing tokens	1105
preprocessor options	267
printf	797
printf_unlocked	797
prof	245
profiling options	245
program instrumentation options	245
program sections	631
promotion of formal parameters	794
PRU Options	448
push address instruction	747
putchar	797
puts	797

Q

q floating point suffix	577
Q floating point suffix	577
qsort, and global register variables	775
quote GCC_COLORS capability	89

R

r fixed-suffix	580
'r' in constraint	745
R fixed-suffix	580
RAMPD	367
RAMPX	367
RAMPY	367
RAMPZ	367
range1 GCC_COLORS capability	89
range2 GCC_COLORS capability	89
ranges in case statements	790
raw string literals	792
read-only strings	1103
realloc	797
register variable after longjmp	775
registers for local variables	775
registers in constraints	745
registers, global allocation	774
registers, global variables in	774
relocation truncated to fit (ColdFire)	417
relocation truncated to fit (MIPS)	424
remainder	797
remainderf	797
remainderl	797
remquo	797
remquof	797
remquol	797
reordering, warning	71
reporting bugs	1117
reset handler functions	678
rest argument (in macro)	789
restricted pointers	1031
restricted references	1031
restricted this pointer	1031
rindex	797

rint	797
rintf	797
rintl	797
RISC-V Options	450
RL78 Options	467
round	797
roundf	797
roundl	797
RS/6000 and PowerPC Options	469
RTTI	1032
run-time error checking options	245
run-time options	286
RX Options	485

S

's' in constraint	746
S/390 and zSeries Options	488
SARIF file output sink	98
save all registers on the Blackfin	662
save all registers on the H8/300, H8/300H, and H8S	664
scalb	797
scalbf	797
scalbl	797
scalbln	797
scalblnf	797
scalbn	797
scalbnf	797
scanf , and constant strings	1103
scanfnl	797
scope of a variable length array	581
scope of declaration	1107
scope of external declarations	1104
search path	282
setjmp	775
setjmp incompatibilities	1103
shared strings	1103
side effect in ?:	790
side effects, macro argument	780
side effects, order of evaluation	1114
signbit	797
signbitd128	797
signbitd32	797
signbitd64	797
signbitf	797
signbitl	797
signed and unsigned values, comparison warning	155
signedness of char type	51
significand	797
significandf	797
significandl	797
SIMD	87
SIMD function cloning	632
simple constraints	745
sin	797
sincos	797

<code>sincosf</code>	797
<code>sincosl</code>	797
<code>sinf</code>	797
<code>sinh</code>	797
<code>sinhf</code>	797
<code>sinhl</code>	797
<code>sinl</code>	797
<code>sinpi</code>	797
<code>sinpif</code>	797
<code>sinpil</code>	797
size of objects	599
<code>sizeof</code>	786
smaller data references	412
smaller data references (PowerPC)	481
<code>snprintf</code>	797
Solaris 2 options	500
<code>SOURCE_DATE_EPOCH</code>	555
SPARC options	501
specified registers	774
specifying compiler version and target machine ..	9
specifying machine version	9
specifying registers for local variables	775
speed of compilation	555, 557
<code>sprintf</code>	797
<code>sqrt</code>	797
<code>sqrtf</code>	797
<code>sqrtl</code>	797
<code>sscanf</code>	797
<code>sscanf</code> , and constant strings	1103
stack allocation builtins	812
stack protection	621, 632
stack scrubbing	633
standard attribute syntax	703
statements inside expressions	780
static data in C++, declaring and defining	1108
<code>stpncpy</code>	797
<code>stpncpy</code>	797
<code>strcasemp</code>	797
<code>strcat</code>	797
<code>strchr</code>	797
<code>strcmp</code>	797
<code>strcpy</code>	797
<code>strcspn</code>	797
<code>strdup</code>	797
<code>strfmon</code>	797
<code>strftime</code>	797
string constants	1103
string literals, raw	792
<code>strlen</code>	797
<code>strncasemp</code>	797
<code>strncat</code>	797
<code>strncmp</code>	797
<code>strncpy</code>	797
<code>strndup</code>	797
<code>strnlen</code>	797
<code>strpbrk</code>	797
<code>strrchr</code>	797
<code>strspn</code>	797

<code>strstr</code>	797
strub eligibility and viability	636
<code>struct</code>	584
<code>struct __htm_tdb</code>	996
structures	1105
structures with only FAMs	584
structures with only flexible array members ...	584
structures, constructor expression	587
submodel options	316
subscripting	586
subscripting and function values	586
suffixes for C++ source	44
<code>SUNPRO_DEPENDENCIES</code>	555
suppressing inlining	621
suppressing warnings	101
surprises in C++	1108
switch statement fallthrough	608
symbol versioning	637
syntax checking	101
system headers, warnings from	142

T

tail recursion	618
tainted arguments	638
<code>tan</code>	797
<code>tanf</code>	797
<code>tanh</code>	797
<code>tanhf</code>	797
<code>tanh1</code>	797
<code>tanl</code>	797
<code>tanpi</code>	797
<code>tanpif</code>	797
<code>tanpil</code>	797
target machine, specifying	9
target-specific code generation attributes	638
target-specific options	316
<code>targs GCC_COLORS</code> capability	89
TC1	3
TC2	3
TC3	3
Technical Corrigenda	3
Technical Corrigendum 1	3
Technical Corrigendum 2	3
Technical Corrigendum 3	3
template instantiation	1034
temporaries, lifetime of	1110
tentative definitions	289
<code>TERM_URLS</code> environment variable	90
<code>tgamma</code>	797
<code>tgammaf</code>	797
<code>tgamma1</code>	797
Thread-Local Storage	715
thunks	784
tiny data section on the H8/300H and H8S	665
TLS	715
<code>TMPDIR</code>	552
<code>toascii</code>	797

<code>tolower</code>	797
<code>toupper</code>	797
<code>towlower</code>	797
<code>toupper</code>	797
traditional C language	273
transactional memory extensions for x86	824
transparent unions	640
<code>trivial_abi</code> type attribute	1039
<code>trunc</code>	797
<code>truncf</code>	797
<code>trunc1</code>	797
two-stage name lookup	1109
type alignment	788
type-diff GCC_COLORS capability	90
<code>type_info</code>	1032
typedef names as function parameters	1104
<code>typeof</code>	786

U

<code>uhk</code> fixed-suffix	580
<code>UHK</code> fixed-suffix	580
<code>uhr</code> fixed-suffix	580
<code>UHR</code> fixed-suffix	580
<code>uk</code> fixed-suffix	580
<code>UK</code> fixed-suffix	580
<code>ulk</code> fixed-suffix	580
<code>ULK</code> fixed-suffix	580
ULL integer suffix	575
<code>ullk</code> fixed-suffix	580
<code>ULLK</code> fixed-suffix	580
<code>ullr</code> fixed-suffix	580
<code>ULLR</code> fixed-suffix	580
<code>ulr</code> fixed-suffix	580
<code>ULR</code> fixed-suffix	580
undefined behavior	1117
undefined function value	1117
underlying type of an <code>enum</code>	788
underscores in variables in macros	786
<code>union</code>	584
union, casting to a	585
unions	1105
unions with FAMs	584
unions with flexible array members	584
unknown pragmas, warning	130
unresolved references and <code>-nodefaultlibs</code>	278
unresolved references and <code>-nostdlib</code>	278
unused entities	642
unused function result	646
<code>ur</code> fixed-suffix	580
<code>UR</code> fixed-suffix	580
urls	90
used entities	642
User stack pointer in interrupts on the Blackfin	661

V

'V' in constraint	745
V850 Options	507
vague linkage	1032
valid GCC_COLORS capability	90
value after <code>longjmp</code>	775
variable addressability on the M32R/D	670
variable alignment	788
variable number of arguments	789
variable-length array in a structure	581
variable-length array scope	581
variable-length arrays	581
variables in arbitrary sections	631
variables in specified registers	774
variables that should not be initialized	621
variables, local, in macros	786
variadic function sentinel	631
variadic functions, pointer arguments	794
variadic macros	789
VAX options	510
<code>vec_blendv</code>	958
<code>vec_cfuge</code>	953
<code>vec_c1rl</code>	954
<code>vec_clrr</code>	954
<code>vec_cntlzm</code>	953
<code>vec_cnttzm</code>	954
<code>vec_extracth</code>	955
<code>vec_extractl</code>	954
<code>vec_genpcvm</code>	961
<code>vec_gnb</code>	954
<code>vec_inserth</code>	956
<code>vec_insertl</code>	955
<code>vec_pdep</code>	955
<code>vec_permx</code>	959
<code>vec_pext</code>	959
<code>vec_replace_element</code>	956
<code>vec_replace_unaligned</code>	957
<code>vec_sldb</code>	957
<code>vec_splati</code>	958
<code>vec_splati_ins</code>	958
<code>vec_splatid</code>	958
<code>vec_srdb</code>	958
<code>vec_stril</code>	959
<code>vec_stril_p</code>	959
<code>vec_strir</code>	959
<code>vec_strir_p</code>	960
<code>vec_ternarylogic</code>	960
<code>vec_test_lsbb_all_ones</code>	953
<code>vec_test_lsbb_all_zeros</code>	953
<code>vec_xst_trunc</code>	931, 932
vector types	642
vector types, using with x86 intrinsics	821
<code>vfprintf</code>	797
<code>vfscanf</code>	797
visibility of symbols	643
Visium options	511
VLAs	581
vold pointers, arithmetic	794

void, size of pointer to 794
 volatile access 720, 1031
 volatile applied to function 795
 volatile **asm** 725
 volatile read 720, 1031
 volatile write 720, 1031
vprintf 797
vscanf 797
vsnprintf 797
vsprintf 797
vsscanf 797
vsx_xl_sext 931
vsx_xl_zext 931
 vtable 1032
 VxWorks Options 512

W

w floating point suffix 577
W floating point suffix 577
warn_if_not_aligned attribute 645
 warning for comparison of signed and
 unsigned values 155
 warning for overloaded virtual function 75
 warning for reordering of member initializers ... 71
 warning for unknown pragmas 130
warning GCC_COLORS capability 89
 warning messages 101
 warnings from system headers 142
 warnings vs errors 1115
 weak symbols 646

whitespace 1104
 Windows Options for x86 549

X

x86 named address spaces 592
 x86 Options 512
 x86 transactional memory extensions 824
 x86 Windows Options 549
 ‘X’ in constraint 746
 X3.159-1989 3
 Xstormy16 Options 549
 Xtensa Options 549

Y

y0 797
y0f 797
y0l 797
y1 797
y1f 797
y1l 797
yn 797
ynf 797
ynl 797

Z

zero-length arrays 582
 zero-size structures 584
 zSeries options 552