

Using the GNU Compiler Collection

For GCC version 17.0.0 (pre-release)

(GCC)

Richard M. Stallman and the GCC Developer Community

Contributors to GCC	1145
----------------------------------	-------------

Appendix A Indices.....	1163
----------------------------------	-------------

A.1 Option Index	1163
--------------------------	------

A.2 Attribute Index.....	1201
----------------------------	------

A.3 Concept and Symbol Index	1205
--------------------------------------	------

K	An unsigned 12-bit constant (for logic instructions).
M	A constant that cannot be loaded using <code>lui</code> , <code>addiu</code> or <code>ori</code> .
N	A constant in the range -65535 to -1 (inclusive).
O	A signed 15-bit constant.
P	A constant in the range 1 to 65535 (inclusive).
R	An address that can be used in a non-macro load or store.
ZB	An address that is held in a general-purpose register. The offset is zero.
ZC	A memory operand whose address is formed by a base register and offset that is suitable for use in instructions with the same addressing mode as <code>ll.w</code> and <code>sc.w</code> .

MicroBlaze—`config/microblaze/constraints.md`

d	A general register (<code>r0</code> to <code>r31</code>).
z	A status register (<code>rmsr</code> , <code>\$fcc1</code> to <code>\$fcc7</code>).

MIPS—`config/mips/constraints.md`

d	A general-purpose register. This is equivalent to <code>r</code> unless generating MIPS16 code, in which case the MIPS16 register set is used.
f	A floating-point register (if available).
h	Formerly the <code>hi</code> register. This constraint is no longer supported.
l	The <code>lo</code> register. Use this register to store values that are no bigger than a word.
x	The concatenated <code>hi</code> and <code>lo</code> registers. Use this register to store doubleword values.
c	A register suitable for use in an indirect jump. This will always be <code>\$25</code> for <code>-mabicalls</code> .
v	Register <code>\$3</code> . Do not use this constraint in new code; it is retained only for compatibility with <code>glibc</code> .
y	Equivalent to <code>r</code> ; retained for backwards compatibility.
z	A floating-point condition code register.
I	A signed 16-bit constant (for arithmetic instructions).
J	Integer zero.
K	An unsigned 16-bit constant (for logic instructions).
L	A signed 32-bit constant in which the lower 16 bits are zero. Such constants can be loaded using <code>lui</code> .
M	A constant that cannot be loaded using <code>lui</code> , <code>addiu</code> or <code>ori</code> .
N	A constant in the range -65535 to -1 (inclusive).

O	A signed 15-bit constant.
P	A constant in the range 1 to 65535 (inclusive).
G	Floating-point zero.
R	An address that can be used in a non-macro load or store.
ZC	A memory operand whose address is formed by a base register and offset that is suitable for use in instructions with the same addressing mode as <code>ll</code> and <code>sc</code> .
ZD	An address suitable for a <code>prefetch</code> instruction, or for any other instruction with the same addressing mode as <code>prefetch</code> .

Motorola 680x0—`config/m68k/constraints.md`

a	Address register
d	Data register
f	68881 floating-point register, if available
I	Integer in the range 1 to 8
J	16-bit signed number
K	Signed number whose magnitude is greater than 0x80
L	Integer in the range -8 to -1
M	Signed number whose magnitude is greater than 0x100
N	Range 24 to 31, rotate:SI 8 to 1 expressed as rotate
O	16 (for rotate using swap)
P	Range 8 to 15, rotate:HI 8 to 1 expressed as rotate
R	Numbers that <code>mov3q</code> can handle
G	Floating point constant that is not a 68881 constant
S	Operands that satisfy 'm' when <code>-mpcrel</code> is in effect
T	Operands that satisfy 's' when <code>-mpcrel</code> is not in effect
Q	Address register indirect addressing mode
U	Register offset addressing
W	<code>const_call_operand</code>
Cs	<code>symbol_ref</code> or <code>const</code>
Ci	<code>const_int</code>
C0	<code>const_int 0</code>
Cj	Range of signed numbers that don't fit in 16 bits
Cmvq	Integers valid for <code>mvq</code>
Capsw	Integers valid for a <code>moveq</code> followed by a swap

Cmvz	Integers valid for mvz
Cmvs	Integers valid for mvs
Ap	push_operand
Ac	Non-register operands allowed in clr

Moxie—config/moxie/constraints.md

A	An absolute address
B	An offset address
W	A register indirect memory operand
I	A constant in the range of 0 to 255.
N	A constant in the range of 0 to −255.

MSP430—config/msp430/constraints.md

R12	Register R12.
R13	Register R13.
K	Integer constant 1.
L	Integer constant -1^{19} .. 1^{19} .
M	Integer constant 1-4.
Ya	Memory references which do not require an extended MOVX instruction.
Yl	Memory reference, labels only.
Ys	Memory reference, stack only.

NDS32—config/nds32/constraints.md

w	LOW register class \$r0 to \$r7 constraint for V3/V3M ISA.
l	LOW register class \$r0 to \$r7.
d	MIDDLE register class \$r0 to \$r11, \$r16 to \$r19.
h	HIGH register class \$r12 to \$r14, \$r20 to \$r31.
t	Temporary assist register \$ta (i.e. \$r15).
k	Stack register \$sp.
Iu03	Unsigned immediate 3-bit value.
In03	Negative immediate 3-bit value in the range of −7−0.
Iu04	Unsigned immediate 4-bit value.
Is05	Signed immediate 5-bit value.
Iu05	Unsigned immediate 5-bit value.
In05	Negative immediate 5-bit value in the range of −31−0.

Ip05	Unsigned immediate 5-bit value for movpi45 instruction with range 16–47.
Iu06	Unsigned immediate 6-bit value constraint for addri36.sp instruction.
Iu08	Unsigned immediate 8-bit value.
Iu09	Unsigned immediate 9-bit value.
Is10	Signed immediate 10-bit value.
Is11	Signed immediate 11-bit value.
Is15	Signed immediate 15-bit value.
Iu15	Unsigned immediate 15-bit value.
Ic15	A constant which is not in the range of imm15u but ok for bclr instruction.
Ie15	A constant which is not in the range of imm15u but ok for bset instruction.
It15	A constant which is not in the range of imm15u but ok for btgl instruction.
Ii15	A constant whose compliment value is in the range of imm15u and ok for bitci instruction.
Is16	Signed immediate 16-bit value.
Is17	Signed immediate 17-bit value.
Is19	Signed immediate 19-bit value.
Is20	Signed immediate 20-bit value.
Ihig	The immediate value that can be simply set high 20-bit.
Izeb	The immediate value 0xff.
Izeh	The immediate value 0xffff.
Ix1s	The immediate value 0x01.
Ix11	The immediate value 0x7ff.
Ibms	The immediate value with power of 2.
Ifex	The immediate value with power of 2 minus 1.
U33	Memory constraint for 333 format.
U45	Memory constraint for 45 format.
U37	Memory constraint for 37 format.

OpenRISC—config/or1k/constraints.md

I	Integer that is valid as an immediate operand in an instruction taking a signed 16-bit number. Range –32768 to 32767.
---	---

K	Integer that is valid as an immediate operand in an instruction taking an unsigned 16-bit number. Range 0 to 65535.
M	Signed 16-bit constant shifted left 16 bits. (Used with <code>l.movhi</code>)
0	Zero

PDP-11—`config/pdp11/constraints.md`

a	Floating point registers AC0 through AC3. These can be loaded from/to memory with a single instruction.
d	Odd numbered general registers (R1, R3, R5). These are used for 16-bit multiply operations.
D	A memory reference that is encoded within the opcode, but not auto-increment or auto-decrement.
f	Any of the floating point registers (AC0 through AC5).
G	Floating point constant 0.
h	Floating point registers AC4 and AC5. These cannot be loaded from/to memory with a single instruction.
I	An integer constant that fits in 16 bits.
J	An integer constant whose low order 16 bits are zero.
K	An integer constant that does not meet the constraints for codes ‘I’ or ‘J’.
L	The integer constant 1.
M	The integer constant -1 .
N	The integer constant 0.
0	Integer constants 0 through 3; shifts by these amounts are handled as multiple single-bit shifts rather than a single variable-length shift.
Q	A memory reference which requires an additional word (address or offset) after the opcode.
R	A memory reference that is encoded within the opcode.

PowerPC and IBM RS6000—`config/rs6000/constraints.md`

r	A general purpose register (GPR), <code>r0...r31</code> .
b	A base register. Like <code>r</code> , but <code>r0</code> is not allowed, so <code>r1...r31</code> .
f	A floating point register (FPR), <code>f0...f31</code> .
d	A floating point register. This is the same as <code>f</code> nowadays; historically <code>f</code> was for single-precision and <code>d</code> was for double-precision floating point.
v	An AltiVec vector register (VR), <code>v0...v31</code> .

wa A VSX register (VSR), `vs0...vs63`. This is either an FPR (`vs0...vs31` are `f0...f31`) or a VR (`vs32...vs63` are `v0...v31`). When using **wa**, you should use the `%x` output modifier, so that the correct register number is printed. For example:

```
asm ("xvadddp %x0,%x1,%x2"
    : "=wa" (v1)
    : "wa" (v2), "wa" (v3));
```

You should not use `%x` for `v` operands:

```
asm ("xsaddqp %0,%1,%2"
    : "=v" (v1)
    : "v" (v2), "v" (v3));
```

c The count register, `ctr`.

l The link register, `lr`.

x Condition register field 0, `cr0`.

y Any condition register field, `cr0...cr7`.

I A signed 16-bit constant.

J An unsigned 16-bit constant shifted left 16 bits (use **L** instead for **SImode** constants).

K An unsigned 16-bit constant.

L A signed 16-bit constant shifted left 16 bits.

eI A signed 34-bit integer constant if prefixed instructions are supported.

eP A scalar floating point constant or a vector constant that can be loaded to a VSX register with one prefixed instruction.

eQ An IEEE 128-bit constant that can be loaded into a VSX register with the `lxvkq` instruction.

m A memory operand. Normally, **m** does not allow addresses that update the base register. If the `<` or `>` constraint is also used, they are allowed and therefore on PowerPC targets in that case it is only safe to use **m<>** in an `asm` statement if that `asm` statement accesses the operand exactly once. The `asm` statement must also use `%U<opno>` as a placeholder for the “update” flag in the corresponding load or store instruction. For example:

```
asm ("st%U0 %1,%0" : "=m<>" (mem) : "r" (val));
```

is correct but:

```
asm ("st %1,%0" : "=m<>" (mem) : "r" (val));
```

is not.

Q A memory operand addressed by just a base register.

Z A memory operand accessed with indexed or indirect addressing.

a An indexed or indirect address.

PRU—config/pru/constraints.md

I	An unsigned 8-bit integer constant.
J	An unsigned 16-bit integer constant.
L	An unsigned 5-bit integer constant (for shift counts).
T	A text segment (program memory) constant label.
Z	Integer constant zero.

RL78—config/rl78/constraints.md

Int3	An integer constant in the range 1 . . . 7.
Int8	An integer constant in the range 0 . . . 255.
J	An integer constant in the range -255 . . . 0
K	The integer constant 1.
L	The integer constant -1.
M	The integer constant 0.
N	The integer constant 2.
O	The integer constant -2.
P	An integer constant in the range 1 . . . 15.
Qbi	The built-in compare types—eq, ne, gtu, ltu, geu, and leu.
Qsc	The synthetic compare types—gt, lt, ge, and le.
Wab	A memory reference with an absolute address.
Wbc	A memory reference using BC as a base register, with an optional offset.
Wca	A memory reference using AX, BC, DE, or HL for the address, for calls.
Wcv	A memory reference using any 16-bit register pair for the address, for calls.
Wd2	A memory reference using DE as a base register, with an optional offset.
Wde	A memory reference using DE as a base register, without any offset.
Wfr	Any memory reference to an address in the far address space.
Wh1	A memory reference using HL as a base register, with an optional one-byte offset.
Whb	A memory reference using HL as a base register, with B or C as the index register.
Whl	A memory reference using HL as a base register, without any offset.
Ws1	A memory reference using SP as a base register, with an optional one-byte offset.

Y	Any memory reference to an address in the near address space.
A	The AX register.
B	The BC register.
D	The DE register.
R	A through L registers.
S	The SP register.
T	The HL register.
Z08W	The 16-bit R8 register.
Z10W	The 16-bit R10 register.
Zint	The registers reserved for interrupts (R24 to R31).
a	The A register.
b	The B register.
c	The C register.
d	The D register.
e	The E register.
h	The H register.
l	The L register.
v	The virtual registers.
w	The PSW register.
x	The X register.

RISC-V—`config/riscv/constraints.md`

f	A floating-point register (if available).
I	An I-type 12-bit signed immediate.
J	Integer zero.
K	A 5-bit unsigned immediate for CSR access instructions.
A	An address that is held in a general-purpose register.
S	A constraint that matches an absolute symbolic address.
vr	A vector register (if available)..
vd	A vector register, excluding v0 (if available).
vm	A vector register, only v0 (if available).
cr	RVC general purpose register (x8-x15).
cf	RVC floating-point registers (f8-f15), if available, reuse GPR as FPR when use zfinx.

cR	Even-odd RVC general purpose register pair.
R	Even-odd general purpose register pair.

RX—config/rx/constraints.md

Q	An address which does not involve register indirect addressing or pre/post increment/decrement addressing.
Symbol	A symbol reference.
Int08	A constant in the range -256 to 255 , inclusive.
Sint08	A constant in the range -128 to 127 , inclusive.
Sint16	A constant in the range -32768 to 32767 , inclusive.
Sint24	A constant in the range -8388608 to 8388607 , inclusive.
Uint04	A constant in the range 0 to 15 , inclusive.

S/390 and zSeries—config/s390/s390.h

a	Address register (general purpose register except r0)
c	Condition code register
d	Data register (arbitrary general purpose register)
f	Floating-point register
I	Unsigned 8-bit constant ($0-255$)
J	Unsigned 12-bit constant ($0-4095$)
K	Signed 16-bit constant ($-32768-32767$)
L	Value appropriate as displacement. ($0..4095$) for short displacement ($-524288..524287$) for long displacement
M	Constant integer with a value of $0x7fffffff$.
N	Multiple letter constraint followed by 4 parameter letters. 0..9: number of the part counting from most to least significant H,Q: mode of the part D,S,H: mode of the containing operand O,F: value of the other parts (F—all bits set) The constraint matches if the specified part of a constant has a value different from its other parts.
Q	Memory reference without index register and with short displacement.

R	Memory reference with index register and short displacement.
S	Memory reference without index register but with long displacement.
T	Memory reference with index register and long displacement.
U	Pointer with short displacement.
W	Pointer with long displacement.
Y	Shift count operand.

SPARC—`config/sparc/sparc.h`

f	Floating-point register on the SPARC-V8 architecture and lower floating-point register on the SPARC-V9 architecture.
e	Floating-point register. It is equivalent to ‘f’ on the SPARC-V8 architecture and contains both lower and upper floating-point registers on the SPARC-V9 architecture.
c	Floating-point condition code register.
d	Lower floating-point register. It is only valid on the SPARC-V9 architecture when the Visual Instruction Set is available.
b	Floating-point register. It is only valid on the SPARC-V9 architecture when the Visual Instruction Set is available.
h	64-bit global or out register for the SPARC-V8+ architecture.
C	The constant all-ones, for floating-point.
A	Signed 5-bit constant
D	A vector constant
I	Signed 13-bit constant
J	Zero
K	32-bit constant with the low 12 bits clear (a constant that can be loaded with the <code>sethi</code> instruction)
L	A constant in the range supported by <code>movcc</code> instructions (11-bit signed immediate)
M	A constant in the range supported by <code>movrcc</code> instructions (10-bit signed immediate)
N	Same as ‘K’, except that it verifies that bits that are not in the lower 32-bit range are all zero. Must be used instead of ‘K’ for modes wider than <code>SImode</code>
O	The constant 4096
G	Floating-point zero
H	Signed 13-bit constant, sign-extended to 32 or 64 bits

P	The constant -1
Q	Floating-point constant whose integral representation can be moved into an integer register using a single sethi instruction
R	Floating-point constant whose integral representation can be moved into an integer register using a single mov instruction
S	Floating-point constant whose integral representation can be moved into an integer register using a high/lo_sum instruction sequence
T	Memory address aligned to an 8-byte boundary
W	Memory address for ‘e’ constraint registers
w	Memory address with only a base register
Y	Vector zero

TI C6X family—`config/c6x/constraints.md`

a	Register file A (A0–A31).
b	Register file B (B0–B31).
A	Predicate registers in register file A (A0–A2 on C64X and higher, A1 and A2 otherwise).
B	Predicate registers in register file B (B0–B2).
C	A call-used register in register file B (B0–B9, B16–B31).
Da	Register file A, excluding predicate registers (A3–A31, plus A0 if not C64X or higher).
Db	Register file B, excluding predicate registers (B3–B31).
Iu4	Integer constant in the range 0 . . . 15.
Iu5	Integer constant in the range 0 . . . 31.
In5	Integer constant in the range –31 . . . 0.
Is5	Integer constant in the range –16 . . . 15.
I5x	Integer constant that can be the operand of an ADDA or a SUBA insn.
IuB	Integer constant in the range 0 . . . 65535.
IsB	Integer constant in the range –32768 . . . 32767.
IsC	Integer constant in the range -2^{20} . . . $2^{20} - 1$.
Jc	Integer constant that is a valid mask for the clr instruction.
Js	Integer constant that is a valid mask for the set instruction.
Q	Memory location with A base register.
R	Memory location with B base register.
Z	Register B14 (aka DP).

Visium—config/visium/constraints.md

b	EAM register <code>mdb</code>
c	EAM register <code>mdc</code>
f	Floating point register
l	General register, but not <code>r29</code> , <code>r30</code> and <code>r31</code>
t	Register <code>r1</code>
u	Register <code>r2</code>
v	Register <code>r3</code>
G	Floating-point constant 0.0
J	Integer constant in the range 0 .. 65535 (16-bit immediate)
K	Integer constant in the range 1 .. 31 (5-bit immediate)
L	Integer constant in the range -65535 .. -1 (16-bit negative immediate)
M	Integer constant -1
O	Integer constant 0
P	Integer constant 32

x86 family—config/i386/constraints.md

R	Legacy register—the eight integer registers available on all i386 processors (<code>a</code> , <code>b</code> , <code>c</code> , <code>d</code> , <code>si</code> , <code>di</code> , <code>bp</code> , <code>sp</code>).
q	Any register accessible as <code>r1</code> . In 32-bit mode, <code>a</code> , <code>b</code> , <code>c</code> , and <code>d</code> ; in 64-bit mode, any integer register.
Q	Any register accessible as <code>rh</code> : <code>a</code> , <code>b</code> , <code>c</code> , and <code>d</code> .
a	The <code>a</code> register.
b	The <code>b</code> register.
c	The <code>c</code> register.
d	The <code>d</code> register.
S	The <code>si</code> register.
D	The <code>di</code> register.
A	The <code>a</code> and <code>d</code> registers. This class is used for instructions that return double word results in the <code>ax:dx</code> register pair. Single word values will be allocated either in <code>ax</code> or <code>dx</code> . For example on i386 the following implements <code>rdtsc</code> :

```

unsigned long long rdtsc (void)
{
    unsigned long long tick;
    __asm__ __volatile__ ("rdtsc":"=A"(tick));
    return tick;
}

```

```
    }
```

This is not correct on x86-64 as it would allocate tick in either **ax** or **dx**. You have to use the following variant instead:

```
unsigned long long rdtsc (void)
{
    unsigned int tickl, tickh;
    __asm__ __volatile__ ("rdtsc":"=a"(tickl),"=d"(tickh));
    return ((unsigned long long)tickh << 32)|tickl;
}
```

U	The call-clobbered integer registers.
f	Any 80387 floating-point (stack) register.
t	Top of 80387 floating-point stack (%st(0)).
u	Second from top of 80387 floating-point stack (%st(1)).
y	Any MMX register.
x	Any SSE register.
v	Any EVEX encodable SSE register (%xmm0–%xmm31).
Yz	First SSE register (%xmm0).
I	Integer constant in the range 0 . . . 31, for 32-bit shifts.
J	Integer constant in the range 0 . . . 63, for 64-bit shifts.
K	Signed 8-bit integer constant.
L	0xFF or 0xFFFF, for andsi as a zero-extending move.
M	0, 1, 2, or 3 (shifts for the lea instruction).
N	Unsigned 8-bit integer constant (for in and out instructions).
G	Standard 80387 floating point constant.
C	SSE constant zero operand.
e	32-bit signed integer constant, or a symbolic reference known to fit that range (for immediate operands in sign-extending x86-64 instructions).
We	32-bit signed integer constant, or a symbolic reference known to fit that range (for sign-extending conversion operations that require non-VOIDmode immediate operands).
Wz	32-bit unsigned integer constant, or a symbolic reference known to fit that range (for zero-extending conversion operations that require non-VOIDmode immediate operands).
Wd	128-bit integer constant where both the high and low 64-bit word satisfy the e constraint.
Ws	A symbolic reference or label reference. You can use the %p modifier to print the raw symbol.

Z	32-bit unsigned integer constant, or a symbolic reference known to fit that range (for immediate operands in zero-extending x86-64 instructions).
Tv	VSIB address operand.
Ts	Address operand without segment register.

Xstormy16—`config/stormy16/stormy16.h`

a	Register r0.
b	Register r1.
c	Register r2.
d	Register r8.
e	Registers r0 through r7.
t	Registers r0 and r1.
y	The carry register.
z	Registers r8 and r9.
I	A constant between 0 and 3 inclusive.
J	A constant that has exactly one bit set.
K	A constant that has exactly one bit clear.
L	A constant between 0 and 255 inclusive.
M	A constant between -255 and 0 inclusive.
N	A constant between -3 and 0 inclusive.
O	A constant between 1 and 4 inclusive.
P	A constant between -4 and -1 inclusive.
Q	A memory reference that is a stack push.
R	A memory reference that is a stack pop.
S	A memory reference that refers to a constant address of known value.
T	The register indicated by Rx (not implemented yet).
U	A constant that is not between 2 and 15 inclusive.
Z	The constant 0.

Xtensa—`config/xtensa/constraints.md`

a	General-purpose 32-bit register
b	One-bit boolean register
A	MAC16 40-bit accumulator register
I	Signed 12-bit integer constant, for use in MOVI instructions
J	Signed 8-bit integer constant, for use in ADDI instructions
K	Integer constant valid for BccI instructions
L	Unsigned constant valid for BccUI instructions

6.11.4 C++11 Constant Expressions instead of String Literals

In C++ with `-std=gnu++11` or later, strings that appear in asm syntax—specifically, the assembler template, constraints, and clobbers—can be specified as parenthesized compile-time constant expressions as well as by string literals. The parentheses around such an expression are a required part of the syntax. The constant expression can return a container with `data ()` and `size ()` member functions, following similar rules as the C++26 `static_assert` message. Any string is converted to the character set of the source code. When this feature is available the `__GXX_CONSTEXPR_ASM__` preprocessor macro is predefined.

This extension is supported for both the basic and extended asm syntax.

```
#include <string>
constexpr std::string_view genfoo() { return "foo"; }

void function()
{
    asm((genfoo()));
}
```

6.11.5 Controlling Names Used in Assembler Code

You can specify the name to be used in the assembler code for a C function or variable by writing the `asm` (or `__asm__`) keyword after the declarator. It is up to you to make sure that the assembler names you choose do not conflict with any other assembler symbols, or reference registers.

Assembler names for data

This sample shows how to specify the assembler name for data:

```
int foo asm ("myfoo") = 2;
```

This specifies that the name to be used for the variable `foo` in the assembler code should be `'myfoo'` rather than the usual `'_foo'`.

On systems where an underscore is normally prepended to the name of a C variable, this feature allows you to define names for the linker that do not start with an underscore.

GCC does not support using this feature with a non-static local variable since such variables do not have assembler names. If you are trying to put the variable in a particular register, see Section 6.11.6 [Explicit Register Variables], page 774.

Assembler names for functions

To specify the assembler name for functions, write a declaration for the function before its definition and put `asm` there, like this:

```
int func (int x, int y) asm ("MYFUNC");

int func (int x, int y)
{
    /* ... */
}
```

This specifies that the name to be used for the function `func` in the assembler code should be `MYFUNC`.

6.11.6 Variables in Specified Registers

GNU C allows you to associate specific hardware registers with C variables. In almost all cases, allowing the compiler to assign registers produces the best code. However under certain unusual circumstances, more precise control over the variable storage is required.

Both global and local variables can be associated with a register. The consequences of performing this association are very different between the two, as explained in the sections below.

6.11.6.1 Defining Global Register Variables

You can define a global register variable and associate it with a specified register like this:

```
register int *foo asm ("r12");
```

Here `r12` is the name of the register that should be used. Note that this is the same syntax used for defining local register variables, but for a global variable the declaration appears outside a function. The `register` keyword is required, and cannot be combined with `static`. The register name must be a valid register name for the target platform.

Do not use type qualifiers such as `const` and `volatile`, as the outcome may be contrary to expectations. In particular, using the `volatile` qualifier does not fully prevent the compiler from optimizing accesses to the register.

Registers are a scarce resource on most systems and allowing the compiler to manage their usage usually results in the best code. However, under special circumstances it can make sense to reserve some globally. For example this may be useful in programs such as programming language interpreters that have a couple of global variables that are accessed very often.

After defining a global register variable, for the current compilation unit:

- If the register is a call-saved register, call ABI is affected: the register will not be restored in function epilogue sequences after the variable has been assigned. Therefore, functions cannot safely return to callers that assume standard ABI.
- Conversely, if the register is a call-clobbered register, making calls to functions that use standard ABI may lose contents of the variable. Such calls may be created by the compiler even if none are evident in the original program, for example when `libgcc` functions are used to make up for unavailable instructions.
- Accesses to the variable may be optimized as usual and the register remains available for allocation and use in any computations, provided that observable values of the variable are not affected.
- If the variable is referenced in inline assembly, the type of access must be provided to the compiler via constraints (see Section 6.11.3 [Constraints], page 744). Accesses from basic asm's are not supported.

Note that these points *only* apply to code that is compiled with the definition. The behavior of code that is merely linked in (for example code from libraries) is not affected.

If you want to recompile source files that do not actually use your global register variable so they do not use the specified register for any other purpose, you need not actually add the global register declaration to their source code. It suffices to specify the compiler option `-ffixed-reg` (see Section 3.18 [Code Gen Options], page 286) to reserve the register.

Declaring the variable

Global register variables cannot have initial values, because an executable file has no means to supply initial contents for a register.

When selecting a register, choose one that is normally saved and restored by function calls on your machine. This ensures that code which is unaware of this reservation (such as library routines) will restore it before returning.

On machines with register windows, be sure to choose a global register that is not affected magically by the function call mechanism.

Using the variable

When calling routines that are not aware of the reservation, be cautious if those routines call back into code which uses them. As an example, if you call the system library version of `qsort`, it may clobber your registers during execution, but (if you have selected appropriate registers) it will restore them before returning. However it will *not* restore them before calling `qsort`'s comparison function. As a result, global values will not reliably be available to the comparison function unless the `qsort` function itself is rebuilt.

Similarly, it is not safe to access the global register variables from signal handlers or from more than one thread of control. Unless you recompile them specially for the task at hand, the system library routines may temporarily use the register for other things. Furthermore, since the register is not reserved exclusively for the variable, accessing it from handlers of asynchronous signals may observe unrelated temporary values residing in the register.

On most machines, `longjmp` restores to each global register variable the value it had at the time of the `setjmp`. On some machines, however, `longjmp` does not change the value of global register variables. To be portable, the function that called `setjmp` should make other arrangements to save the values of the global register variables, and to restore them in a `longjmp`. This way, the same thing happens regardless of what `longjmp` does.

6.11.6.2 Specifying Registers for Local Variables

You can define a local register variable and associate it with a specified register like this:

```
register int *foo asm ("r12");
```

Here `r12` is the name of the register that should be used. Note that this is the same syntax used for defining global register variables, but for a local variable the declaration appears within a function. The `register` keyword is required, and cannot be combined with `static`. The register name must be a valid register name for the target platform.

Do not use type qualifiers such as `const` and `volatile`, as the outcome may be contrary to expectations. In particular, when the `const` qualifier is used, the compiler may substitute the variable with its initializer in `asm` statements, which may cause the corresponding operand to appear in a different register.

As with global register variables, it is recommended that you choose a register that is normally saved and restored by function calls on your machine, so that calls to library routines will not clobber it.

The only supported use for this feature is to specify registers for input and output operands when calling Extended `asm` (see Section 6.11.2 [Extended Asm], page 723). This may be necessary if the constraints for a particular machine don't provide sufficient control

to select the desired register. To force an operand into a register, create a local variable and specify the register name after the variable's declaration. Then use the local variable for the `asm` operand and specify any constraint letter that matches the register:

```
register int *p1 asm ("r0") = ...;
register int *p2 asm ("r1") = ...;
register int *result asm ("r0");
asm ("sysint" : "=r" (result) : "0" (p1), "r" (p2));
```

Warning: In the above example, be aware that a register (for example `r0`) can be clobbered by subsequent code, including function calls and library calls for arithmetic operators on other variables (for example the initialization of `p2`). In this case, use temporary variables for expressions between the register assignments:

```
int t1 = ...;
register int *p1 asm ("r0") = ...;
register int *p2 asm ("r1") = t1;
register int *result asm ("r0");
asm ("sysint" : "=r" (result) : "0" (p1), "r" (p2));
```

Defining a register variable does not reserve the register. Other than when invoking the Extended `asm`, the contents of the specified register are not guaranteed. For this reason, the following uses are explicitly *not* supported. If they appear to work, it is only happenstance, and may stop working as intended due to (seemingly) unrelated changes in surrounding code, or even minor changes in the optimization of a future version of gcc:

- Passing parameters to or from Basic `asm`
- Passing parameters to or from Extended `asm` without using input or output operands.
- Passing parameters to or from routines written in assembler (or other languages) using non-standard calling conventions.

Some developers use Local Register Variables in an attempt to improve gcc's allocation of registers, especially in large functions. In this case the register name is essentially a hint to the register allocator. While in some instances this can generate better code, improvements are subject to the whims of the allocator/optimizers. Since there are no guarantees that your improvements won't be lost, this usage of Local Register Variables is discouraged.

On the MIPS platform, there is related use for local register variables with slightly different characteristics (see Section "Defining coprocessor specifics for MIPS targets" in *GNU Compiler Collection (GCC) Internals*).

6.11.6.3 Hard Register Constraints

Similar to register `asm` but still distinct, hard register constraints are another way to force operands of inline `asm` into specific machine registers. In contrast to register `asm` where a variable is bound to a machine register, a hard register constraint binds an `asm` operand to a machine register. Assume in the following that `r4` is a general-purpose register, `f5` a floating-point register, and `v6` a vector register for some target.

```
int x;
int y __attribute__((vector_size (16)));
...
asm ("some instructions"
    : "{r4}" (x)
    : "{f5}" (42.0), "{v6}" (y));
```

For the inline `asm`, variable `x` is bound to register `r4`, and `y` is loaded to `v6`. Furthermore, constant `42.0` is loaded into floating-point register `f5`.

A key difference between register `asm` and hard register constraints is that the latter are specified at the point where they are supposed to materialize, namely at inline `asm`, which may lead to more readable code.

Usage

Each input operand is loaded into the register specified by its corresponding hard register constraint. Furthermore, each hard register must be used at most once among an alternative for inputs. This renders hard register constraints more strict compared to register `asm` where multiple inputs may share a register as for example in

```
int x;
register int y asm ("0") = ...;
asm ("": "=r" (x) : "r" (y), "r" (y));
```

or even

```
register int x asm ("0") = 42;
register int y asm ("0") = 24;
asm ("": "=r" (x) : "r" (x), "r" (y));
```

The analogue for hard register constraints is invalid in order to prevent subtle bugs.

Likewise, two outputs must not share a register among an alternative. That means, the following example is invalid

```
int x, y;
asm ("": "={r4}" (x), "={r4}" (y)); // invalid
```

which also aligns with register `asm`. Despite that, each output must refer to a distinct object if a hard register constraint is involved. For example, in the following, object `x` is assigned two registers.

```
int x;
asm ("": "=r" (x), "=r" (x));
```

This is not allowed for hard register constraints in order to prevent subtle bugs. Even if only one output operand has a hard register constraint, the code is rejected since the allocation for the object is still ambiguous.

```
int x;
asm ("": "=r" (x), "={1}" (x)); // invalid
```

The type of an operand must be supported by the corresponding machine register.

A hard register constraint may refer to any general, floating-point, or vector register except a fixed register as e.g. the stack-pointer register. The set of allowed registers is target dependent analogue to register `asm`. Furthermore, the referenced register must be a valid register name of the target. Note, on some targets, a single register may be referred to by different names where each name specifies the length of the register. For example, on x86_64 the register names `rcx`, `ecx`, and `cx` all refer to the same register but in different sizes. If any of those names is used for a hard register constraint, the actual size of a register is determined by its corresponding operand. For example

```
long x;
asm ("mov\t$42, %0" : "={ecx}" (x));
```

Although the hard register constraint refers to register `ecx`, the actual register will be `rcx` since on x86_64 a `long` is 8 byte in total. This aligns with register `asm` where you could have

```
register long x asm ("ecx");
```

Interaction with Register asm

A mixture of both constructs as for example

```
register int x asm ("r4") = 42;
int y;
asm (" : "={r5}" (y) : "r" (x));
```

is valid.

If an operand is a register `asm` and the corresponding constraint a hard register, then both must refer to the same register. That means

```
register int x asm ("r4");
asm (" : "={r4}" (x));
```

is valid and

```
register int x asm ("r4");
asm (" : "={r5}" (x)); // invalid
```

is invalid.

Note, register `asm` may not only be clobbered by function calls but also by inline `asm` in conjunction with hard register constraints. For example, in the following

```
register int x asm ("r5") = 42;
int y;
asm (" : "={r5}" (y));
asm (" : "+r" (x));
```

variable `x` materializes before the very first inline `asm` which writes to register `r5` and therefore clobbers `x` which in turn is read by the subsequent inline `asm`.

Limitations

At the moment fixed registers are not supported for hard register constraints. Thus, idioms like

```
register void *x asm ("rsp");
asm (" : "=r" (x));
```

are not supported for hard register constraints. This might be lifted.

Multiple hard register constraints in one alternative are also not supported. Furthermore, combinations of hard register constraints and regular register constraints in one alternative are not supported, too.

6.11.7 Size of an asm

Some targets require that GCC track the size of each instruction used in order to generate correct code. Because the final length of the code produced by an `asm` statement is only known by the assembler, GCC must make an estimate as to how big it will be. It does this by counting the number of instructions in the pattern of the `asm` and multiplying that by the length of the longest instruction supported by that processor. (When working out the number of instructions, it assumes that any occurrence of a newline or of whatever statement separator character is supported by the assembler — typically ‘;’ — indicates the end of an instruction.)

Normally, GCC’s estimate is adequate to ensure that correct code is generated, but it is possible to confuse the compiler if you use pseudo instructions or assembler macros that expand into multiple real instructions, or if you use assembler directives that expand to

more space in the object file than is needed for a single instruction. If this happens then the assembler may produce a diagnostic saying that a label is unreachable.

This size is also used for inlining decisions. If you use `asm inline` instead of just `asm`, then for inlining purposes the size of the `asm` is taken as the minimum size, ignoring how many instructions GCC thinks it is.

6.12 Other Extensions to C Syntax

GNU C has traditionally supported numerous extensions to standard C syntax. Some of these features were originally intended for compatibility with other compilers or to ease traditional C compatibility, some have been adopted into subsequent versions of the C and/or C++ standards, while others remain specific to GNU C.

6.12.1 Statements and Declarations in Expressions

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

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

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

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

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

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

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

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

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

Note that introducing variable declarations (as we do in `maxint`) can cause variable shadowing, so while this example using the `max` macro produces correct results:

```
int _a = 1, _b = 2, c;
c = max (_a, _b);
```

this example using `maxint` will not:

```
int _a = 1, _b = 2, c;
c = maxint (_a, _b);
```

This problem may for instance occur when we use this pattern recursively, like so:

```
#define maxint3(a, b, c) \
```

```
{int _a = (a), _b = (b), _c = (c); maxint (maxint (_a, _b), _c); }
```

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

If you don't know the type of the operand, you can still do this, but you must use `typeof` or `__auto_type` (see Section 6.12.5 [Typeof], page 784).

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

```
A a;

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

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

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

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

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

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

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

Jumping into a statement expression with `goto` or using a `switch` statement outside the statement expression with a `case` or `default` label inside the statement expression is not permitted. Jumping into a statement expression with a computed `goto` (see Section 6.12.3 [Labels as Values], page 782) has undefined behavior. Jumping out of a statement expression is permitted, but if the statement expression is part of a larger expression then it is unspecified which other subexpressions of that expression have been evaluated except where the language definition requires certain subexpressions to be evaluated before or after the statement expression. A `break` or `continue` statement inside of a statement expression used in `while`, `do` or `for` loop or `switch` statement condition or `for` statement init or increment expressions jumps to an outer loop or `switch` statement if any (otherwise it is an error), rather than to the loop or `switch` statement in whose condition or init or increment expression it appears. In any case, as with a function call, the evaluation of a statement expression is not interleaved with the evaluation of other parts of the containing expression. For example,

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

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

6.12.2 Locally Declared Labels

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

A local label declaration looks like this:

```
__label__ label;
```

or

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

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

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

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

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

This could also be written using a statement expression:

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

```
} )
```

Local label declarations also make the labels they declare visible to nested functions, if there are any. See Section 6.12.4 [Nested Functions], page 783, for details.

6.12.3 Labels as Values

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

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

To use these values, you need to be able to jump to one. This is done with the computed `goto` statement², `goto *exp`;. For example,

```
goto *ptr;
```

Any expression of type `void *` is allowed.

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

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

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

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

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

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

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

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

An alternate way to write the above example is

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

This is more friendly to code living in shared libraries, as it reduces the number of dynamic relocations that are needed, and by consequence, allows the data to be read-only. This alternative with label differences is not supported for the AVR target, please use the first approach for AVR programs.

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

² The analogous feature in Fortran is called an assigned goto, but that name seems inappropriate in C, where one can do more than simply store label addresses in label variables.

Unlike a normal `goto`, in GNU C++ a computed `goto` will not call destructors for objects that go out of scope.

6.12.4 Nested Functions

A *nested function* is a function defined inside another function. Nested functions are supported as an extension in GNU C, but are not supported by GNU C++.

The nested function's name is local to the block where it is defined. For example, here we define a nested function named `square`, and call it twice:

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

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

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

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

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

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

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

    intermediate (store, size);
}
```

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

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

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

A nested function can jump to a label inherited from a containing function, provided the label is explicitly declared in the containing function (see Section 6.12.2 [Local Labels], page 781). Such a jump returns instantly to the containing function, exiting the nested function that did the `goto` and any intermediate functions as well. Here is an example:

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

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

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

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

6.12.5 Referring to a Type with `typeof`

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

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

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

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

Here is an example with a typename as the argument:

```
typeof (int *)
```

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

If you are writing a header file that must work when included in ISO C programs, write `__typeof__` instead of `typeof`. See Section 6.12.23 [Alternate Keywords], page 791.

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

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

`typeof` is often useful in conjunction with statement expressions (see Section 6.12.1 [Statement Exprs], page 779). Here is how the two together can be used to define a safe “maximum” macro which operates on any arithmetic type and evaluates each of its arguments exactly once:

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

The reason for using names that start with underscores for the local variables is to avoid conflicts with variable names that occur within the expressions that are substituted for `a` and `b`. Eventually we hope to design a new form of declaration syntax that allows you to declare variables whose scopes start only after their initializers; this will be a more reliable way to prevent such conflicts.

Some more examples of the use of `typeof`:

- This declares `y` with the type of what `x` points to.
`typeof (*x) y;`
- This declares `y` as an array of such values.
`typeof (*x) y[4];`
- This declares `y` as an array of pointers to characters:
`typeof (typeof (char *)[4]) y;`

It is equivalent to the following traditional C declaration:

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

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

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

Now the declaration can be rewritten this way:

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

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

The ISO C23 operator `typeof_unqual` is available in ISO C23 mode and its result is the non-atomic unqualified version of what `typeof` operator returns. Alternate spelling `__typeof_unqual__` is available in all C modes and provides non-atomic unqualified version of what `__typeof__` operator returns. See Section 6.12.23 [Alternate Keywords], page 791.

In GNU C, but not GNU C++, you may also declare the type of a variable as `__auto_type`. In that case, the declaration must declare only one variable, whose declarator must just be an identifier, the declaration must be initialized, and the type of the variable is determined by the initializer; the name of the variable is not in scope until after the initializer. (In C++, you should use C++11 `auto` for this purpose.) Using `__auto_type`, the “maximum” macro above could be written as:

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

Using `__auto_type` instead of `typeof` has two advantages:

- Each argument to the macro appears only once in the expansion of the macro. This prevents the size of the macro expansion growing exponentially when calls to such macros are nested inside arguments of such macros.
- If the argument to the macro has variably modified type, it is evaluated only once when using `__auto_type`, but twice if `typeof` is used.

6.12.6 Determining the Number of Elements of Arrays

The keyword `_Countof` determines the number of elements of an array operand. Its syntax is similar to `sizeof`. The operand must be a parenthesized complete array type name or an expression of such a type. For example:

```
int a[n];
_Countof (a); // returns n
_Countof (int [7][3]); // returns 7
```

The result of this operator is an integer constant expression, unless the array has a variable number of elements. The operand is only evaluated if the array has a variable number of elements. For example:

```
_Countof (int [7][n++]); // integer constant expression
_Countof (int [n++][7]); // run-time value; n++ is evaluated
```

6.12.7 The maximum and minimum representable values of a type

The keywords `_Maxof` and `_Minof` determine the maximum and minimum representable values of an integer type. Their syntax is similar to `sizeof`. The operand must be a parenthesized integer type. The result of these operators is an integer constant expression of the same type as the operand. For example:

```
_Maxof (int); // returns '(int) INT_MAX'
_Minof (short); // returns '(short) SHRT_MIN'
```

6.12.8 Support for `offsetof`

GCC implements for both C and C++ a syntactic extension to implement the `offsetof` macro.

```
primary:
    "__builtin_offsetof" "(" typename "," offsetof_member_designator ")"

offsetof_member_designator:
    identifier
    | offsetof_member_designator "." identifier
    | offsetof_member_designator "[" expr "]"
```

This extension is sufficient such that

```
#define offsetof(type, member) __builtin_offsetof (type, member)
```

is a suitable definition of the `offsetof` macro. In C++, *type* may be dependent. In either case, *member* may consist of a single identifier, or a sequence of member accesses and array references.

6.12.9 Determining the Alignment of Functions, Types or Variables

The keyword `__alignof__` determines the alignment requirement of a function, object, or a type, or the minimum alignment usually required by a type. Its syntax is just like `sizeof` and C11 `_Alignof`.

For example, if the target machine requires a `double` value to be aligned on an 8-byte boundary, then `__alignof__ (double)` is 8. This is true on many RISC machines. On more traditional machine designs, `__alignof__ (double)` is 4 or even 2.

Some machines never actually require alignment; they allow references to any data type even at an odd address. For these machines, `__alignof__` reports the smallest alignment that GCC gives the data type, usually as mandated by the target ABI.

If the operand of `__alignof__` is an lvalue rather than a type, its value is the required alignment for its type, taking into account any minimum alignment specified by attribute `aligned` (see Section 6.4.1 [Common Attributes], page 595). For example, after this declaration:

```
struct foo { int x; char y; } foo1;
```

the value of `__alignof__ (foo1.y)` is 1, even though its actual alignment is probably 2 or 4, the same as `__alignof__ (int)`. It is an error to ask for the alignment of an incomplete type other than `void`.

If the operand of the `__alignof__` expression is a function, the expression evaluates to the alignment of the function which may be specified by attribute `aligned` (see Section 6.4.1 [Common Attributes], page 595).

6.12.10 Extensions to enum Type Declarations

The C23 and C++11 standards added new syntax to specify the underlying type of an `enum` type. For example,

```
enum pet : unsigned char { CAT, DOG, ROCK };
```

In GCC, this feature is supported as an extension in all older dialects of C and C++ as well. For C++ dialects before C++11, use `-Wno-c++11-extensions` to silence the associated warnings.

You can also forward-declare an `enum` type, without specifying its possible values. The enumerators are supplied in a later redeclaration of the type, which must match the underlying type of the first declaration.

```
enum pet : unsigned char;
static enum pet my_pet;
...
enum pet : unsigned char { CAT, DOG, ROCK };
```

Forward declaration of `enum` types with an explicit underlying type is also a feature of C++11 that is supported as an extension by GCC for all C dialects. However, it's not available in C++ dialects prior to C++11.

The C++ standard refers to a forward declaration of an `enum` with an explicit underlying type as an *opaque type*. It is not considered an incomplete type, since its size is known. That means you can declare variables or allocate storage using the type before the redeclaration, not just use pointers of that type.

GCC has also traditionally supported forward declarations of `enum` types that don't include an explicit underlying type specification. This results in an incomplete type, much like what you get if you write `struct foo` without describing the elements. You cannot allocate variables or storage using the type while it is incomplete. However, you can work with pointers to that type.

Forward-declaring an incomplete `enum` type without an explicit underlying type is supported as an extension in all GNU C dialects, but is not supported at all in GNU C++.

6.12.11 Support for the `_Bool` Type

The C99 standard added `_Bool` as a C language keyword naming the boolean type. As an extension, GNU C also recognizes `_Bool` in C90 mode as well as with `-std=c99` and later.

C23 added `bool` as the preferred name of the boolean type, but `_Bool` also remains a standard keyword in the language and is supported as such by GCC with `-std=c23`.

GNU C++ does not support `_Bool` as a keyword, but including `<stdbool.h>` defines it as a macro in terms of standard C++'s `bool` type.

6.12.12 Macros with a Variable Number of Arguments.

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

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

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

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

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

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

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

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

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

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

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

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

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

6.12.13 Conditionals with Omitted Operands

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

Therefore, the expression

```
x ? : y
```

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

This example is perfectly equivalent to

```
x ? x : y
```

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

6.12.14 Case Ranges

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

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

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

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

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

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

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

rather than this:

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

6.12.15 Mixed Declarations, Labels and Code

ISO C99 and ISO C++ allow declarations and code to be freely mixed within compound statements. ISO C23 allows labels to be placed before declarations and at the end of a compound statement. As an extension, GNU C also allows all this in C90 mode. For example, you could do:

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

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

6.12.16 C++ Style Comments

In GNU C, you may use C++ style comments, which start with `/**` and continue until the end of the line. Many other C implementations allow such comments, and they are included in the 1999 C standard. However, C++ style comments are not recognized if you specify an `-std` option specifying a version of ISO C before C99, or `-ansi` (equivalent to `-std=c90`).

6.12.17 Slightly Looser Rules for Escaped Newlines

The preprocessor treatment of escaped newlines is more relaxed than that specified by the C90 standard, which requires the newline to immediately follow a backslash. GCC's implementation allows whitespace in the form of spaces, horizontal and vertical tabs, and form feeds between the backslash and the subsequent newline. The preprocessor issues a warning, but treats it as a valid escaped newline and combines the two lines to form a single logical line. This works within comments and tokens, as well as between tokens. Comments are *not* treated as whitespace for the purposes of this relaxation, since they have not yet been replaced with spaces.

6.12.18 Hex Floats

ISO C99 and ISO C++17 support floating-point numbers written not only in the usual decimal notation, such as `1.55e1`, but also numbers such as `0x1.fp3` written in hexadecimal format. As a GNU extension, GCC supports this in C90 mode (except in some cases when strictly conforming) and in C++98, C++11 and C++14 modes. In that format the `'0x'` hex introducer and the `'p'` or `'P'` exponent field are mandatory. The exponent is a decimal number that indicates the power of 2 by which the significant part is multiplied. Thus `'0x1.f'` is $1\frac{15}{16}$, `'p3'` multiplies it by 8, and the value of `0x1.fp3` is the same as `1.55e1`.

Unlike for floating-point numbers in the decimal notation the exponent is always required in the hexadecimal notation. Otherwise the compiler would not be able to resolve the ambiguity of, e.g., `0x1.f`. This could mean `1.0f` or `1.9375` since `'f'` is also the extension for floating-point constants of type `float`.

6.12.19 Binary Constants using the `'0b'` Prefix

Integer constants can be written as binary constants, consisting of a sequence of `'0'` and `'1'` digits, prefixed by `'0b'` or `'0B'`. This is particularly useful in environments that operate a lot on the bit level (like microcontrollers).

The following statements are identical:

```
i =      42;
i =     0x2a;
i =      052;
i = 0b101010;
```

The type of these constants follows the same rules as for octal or hexadecimal integer constants, so suffixes like `'L'` or `'UL'` can be applied.

6.12.20 Dollar Signs in Identifier Names

In GNU C, you may normally use dollar signs in identifier names. This is because many traditional C implementations allow such identifiers. However, dollar signs in identifiers are not supported on a few target machines, typically because the target assembler does not allow them.

6.12.21 The Character ESC in Constants

You can use the sequence ‘\e’ in a string or character constant to stand for the ASCII character ESC.

6.12.22 Raw String Literals

The C++11 standard added syntax for raw string literals prefixed with ‘R’. This syntax allows you to use an arbitrary delimiter sequence instead of escaping special characters within the string. For example, these string constants are all equivalent:

```
const char *s1 = "\\\";
const char *s2 = R"(\)";
const char *s3 = R"foo(\)foo";
```

As an extension, GCC also accepts raw string literals in C with `-std=gnu99` or later.

6.12.23 Alternate Keywords

`-ansi` and the various `-std` options disable certain keywords that are GNU C extensions. Specifically, the keywords `asm`, `typeof` and `inline` are not available in programs compiled with `-ansi` or a `-std=` option specifying an ISO standard that doesn’t define the keyword. This causes trouble when you want to use these extensions in a header file that can be included in programs that may be compiled with with such options.

The way to solve these problems is to put ‘__’ at the beginning and end of each problematical keyword. For example, use `__asm__` instead of `asm`, and `__inline__` instead of `inline`.

Other C compilers won’t accept these alternative keywords; if you want to compile with another compiler, you can define the alternate keywords as macros to replace them with the customary keywords. It looks like this:

```
#ifndef __GNUC__
#define __asm__ asm
#endif
```

`-pedantic` and other options cause warnings for many GNU C extensions. You can suppress such warnings using the keyword `__extension__`. Specifically:

- Writing `__extension__` before an expression prevents warnings about extensions within that expression.
- In C, writing:

```
[[__extension__ ...]]
```

suppresses warnings about using ‘[[]]’ attributes in C versions that predate C23.

`__extension__` has no effect aside from this.

6.12.24 Function Names as Strings

GCC provides three magic constants that hold the name of the current function as a string. In C++11 and later modes, all three are treated as constant expressions and can be used in `constexpr` contexts. The first of these constants is `__func__`, which is part of the C99 standard:

The identifier `__func__` is implicitly declared by the translator as if, immediately following the opening brace of each function definition, the declaration

```
static const char __func__[] = "function-name";
```

appeared, where function-name is the name of the lexically-enclosing function. This name is the unadorned name of the function. As an extension, at file (or, in C++, namespace scope), `__func__` evaluates to the empty string.

`__FUNCTION__` is another name for `__func__`, provided for backward compatibility with old versions of GCC.

In C, `__PRETTY_FUNCTION__` is yet another name for `__func__`, except that at file scope (or, in C++, namespace scope), it evaluates to the string `"top level"`. In addition, in C++, `__PRETTY_FUNCTION__` contains the signature of the function as well as its bare name. For example, this program:

```
extern "C" int printf (const char *, ...);

class a {
public:
    void sub (int i)
    {
        printf ("__FUNCTION__ = %s\n", __FUNCTION__);
        printf ("__PRETTY_FUNCTION__ = %s\n", __PRETTY_FUNCTION__);
    }
};

int
main (void)
{
    a ax;
    ax.sub (0);
    return 0;
}
```

gives this output:

```
__FUNCTION__ = sub
__PRETTY_FUNCTION__ = void a::sub(int)
```

These identifiers are variables, not preprocessor macros, and may not be used to initialize `char` arrays or be concatenated with string literals.

6.13 Extensions to C Semantics

GNU C defines useful behavior for some constructs that are not allowed or well-defined in standard C.

6.13.1 Prototypes and Old-Style Function Definitions

GNU C extends ISO C to allow a function prototype to override a later old-style non-prototype definition. Consider the following example:

```
/* Use prototypes unless the compiler is old-fashioned. */
#ifdef __STDC__
#define P(x) x
#else
#define P(x) ()
#endif

/* Prototype function declaration. */
int isroot P((uid_t));

/* Old-style function definition. */
```

```

int
isroot (x)    /* ??? lossage here ??? */
    uid_t x;
{
    return x == 0;
}

```

Suppose the type `uid_t` happens to be `short`. ISO C does not allow this example, because subword arguments in old-style non-prototype definitions are promoted. Therefore in this example the function definition's argument is really an `int`, which does not match the prototype argument type of `short`.

This restriction of ISO C makes it hard to write code that is portable to traditional C compilers, because the programmer does not know whether the `uid_t` type is `short`, `int`, or `long`. Therefore, in cases like these GNU C allows a prototype to override a later old-style definition. More precisely, in GNU C, a function prototype argument type overrides the argument type specified by a later old-style definition if the former type is the same as the latter type before promotion. Thus in GNU C the above example is equivalent to the following:

```

int isroot (uid_t);

int
isroot (uid_t x)
{
    return x == 0;
}

```

GNU C++ does not support old-style function definitions, so this extension is irrelevant.

6.13.2 Arithmetic on void- and Function-Pointers

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

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

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

6.13.3 Pointer Arguments in Variadic Functions

Standard C requires that pointer types used with `va_arg` in functions with variable argument lists either must be compatible with that of the actual argument, or that one type must be a pointer to `void` and the other a pointer to a character type. GNU C implements the POSIX XSI extension that additionally permits the use of `va_arg` with a pointer type to receive arguments of any other pointer type.

In particular, in GNU C `'va_arg (ap, void *)'` can safely be used to consume an argument of any pointer type.

6.13.4 Pointers to Arrays with Qualifiers Work as Expected

In GNU C, pointers to arrays with qualifiers work similar to pointers to other qualified types. For example, a value of type `int (*) [5]` can be used to initialize a variable of type `const int (*) [5]`. These types are incompatible in ISO C because the `const` qualifier is formally attached to the element type of the array and not the array itself.

```
extern void
```

```
transpose (int N, int M, double out[M][N], const double in[N][M]);
double x[3][2];
double y[2][3];
...
transpose(3, 2, y, x);
```

6.13.5 Const and Volatile Functions

The C standard explicitly leaves the behavior of the `const` and `volatile` type qualifiers applied to functions undefined; these constructs can only arise through the use of `typedef`. As an extension, GCC defines this use of the `const` qualifier to have the same meaning as the GCC `const` function attribute, and the `volatile` qualifier to be equivalent to the `noreturn` attribute. See Section 6.4.1 [Common Attributes], page 595, for more information.

As examples of this usage,

```
/* Equivalent to:
   void fatal () __attribute__ ((noreturn)); */
typedef void voidfn ();
volatile voidfn fatal;

/* Equivalent to:
   extern int square (int) __attribute__ ((const)); */
typedef int intfn (int);
extern const intfn square;
```

In general, using function attributes instead is preferred, since the attributes make both the intent of the code and its reliance on a GNU extension explicit. Additionally, using `const` and `volatile` in this way is specific to GNU C and does not work in GNU C++.

7 Built-in Functions Provided by GCC

GCC provides a very large number of implicitly-declared built-in functions that are typically inlined by the compiler. Some of these builtins directly correspond to standard library routines. Some are for internal use in the processing of exceptions or variable-length argument lists and are not documented here because they may change from time to time; we do not recommend general use of these functions.

The remaining functions are provided either for optimization purposes, or to expose low-level functionality needed to implement features provided by library functions or similar “glue” between GCC and other programming languages or libraries. Others are target-specific, providing direct access to instructions that have no direct C equivalents without the need to write assembly language. There are also builtins to support various kinds of runtime error checking.

Most builtins have names prefixed with ‘`__builtin_`’, although not all of them use this convention. Except as otherwise documented, all built-in functions are available from any of the C family languages supported by GCC.

With the exception of built-ins that have library equivalents such as the standard C library functions discussed below in Section 7.1 [Library Builtins], page 795, or that expand to library calls, GCC built-in functions are always expanded inline and thus do not have corresponding entry points and their address cannot be obtained. Attempting to use them in an expression other than a function call results in a compile-time error.

7.1 Builtins for C Library Functions

GCC includes built-in versions of many of the functions in the standard C library. These functions come in two forms: one whose names start with the `__builtin_` prefix, and the other without. Both forms have the same type (including prototype), the same address (when their address is taken), and the same meaning as the C library functions even if you specify the `-fno-builtin` option see Section 3.4 [C Dialect Options], page 45). Many of these functions are only optimized in certain cases; if they are not optimized in a particular case, a call to the library function is emitted.

Outside strict ISO C mode (`-ansi`, `-std=c90`, `-std=c99` or `-std=c11`), the functions `_exit`, `alloca`, `acospi`, `acospif`, `acospil`, `asinpi`, `asinpif`, `asinpil`, `atan2pi`, `atan2pif`, `atan2pil`, `atanpi`, `atanpif`, `atanpil`, `bcmp`, `bzero`, `cospi`, `cospif`, `cospil`, `dcgettext`, `dgettext`, `dremf`, `dreml`, `drem`, `exp10f`, `exp10l`, `exp10`, `ffsll`, `ffsl`, `ffs`, `fprintf_unlocked`, `fputs_unlocked`, `gammaf`, `gammal`, `gamma`, `gammaf_r`, `gammal_r`, `gamma_r`, `gettext`, `index`, `isascii`, `j0f`, `j0l`, `j0`, `j1f`, `j1l`, `j1`, `jnf`, `jnl`, `jn`, `lgammaf_r`, `lgammal_r`, `lgamma_r`, `mempcpy`, `pow10f`, `pow10l`, `pow10`, `printf_unlocked`, `rindex`, `roundeven`, `roundevenf`, `roundevenl`, `scalbf`, `scalbl`, `scalb`, `signbit`, `signbitf`, `signbitl`, `signbitd32`, `signbitd64`, `signbitd128`, `significandf`, `significandl`, `significand`, `sincosf`, `sincosl`, `sincos`, `sinpif`, `sinpil`, `sinpi`, `stpcpy`, `stpncpy`, `strcasecmp`, `strdup`, `strfmon`, `strncasecmp`, `strndup`, `strnlen`, `tanpif`, `tanpil`, `tanpi`, `toascii`, `y0f`, `y0l`, `y0`, `y1f`, `y1l`, `y1`, `ynf`, `ynl` and `yn` may be handled as built-in functions. All these functions have corresponding versions prefixed with `__builtin_`, which may be used even in strict C90 mode.

The ISO C99 functions `_Exit`, `acoshf`, `acoshl`, `acosh`, `asinhf`, `asinh`, `atanhf`, `atanhl`, `atanh`, `cabsf`, `cabsl`, `cabs`, `cacosf`, `cacoshf`, `cacoshl`, `cacosh`, `cacosl`, `cacos`, `cargf`, `cargl`, `carg`, `casinf`, `casinhf`, `casinh`, `casinl`, `casin`, `catanf`, `catanhf`, `catanh`, `catanl`, `catan`, `cbrtf`, `cbrtl`, `cbrt`, `ccosf`, `ccoshf`, `ccoshl`, `ccosh`, `ccosl`, `ccos`, `cexpf`, `cexpl`, `cexp`, `cimagf`, `cimagl`, `cimag`, `clogf`, `clogl`, `clog`, `conjf`, `conjl`, `conj`, `copysignf`, `copysignl`, `copysign`, `cpowf`, `cpowl`, `cpow`, `cprojf`, `cprojl`, `cproj`, `crealf`, `creall`, `creal`, `csinf`, `csinhf`, `csinh`, `csinl`, `csin`, `csqrtf`, `csqrtl`, `csqrt`, `ctanf`, `ctanhf`, `ctanh`, `ctanl`, `ctan`, `erfcf`, `erfcl`, `erfc`, `erff`, `erfl`, `erf`, `exp2f`, `exp2l`, `exp2`, `expm1f`, `expm1l`, `expm1`, `fdimf`, `fdiml`, `fdim`, `fmaf`, `fmalf`, `fmaxf`, `fmaxl`, `fmax`, `fma`, `fminf`, `fminl`, `fmin`, `hypotf`, `hypotl`, `hypot`, `ilogbf`, `ilogbl`, `ilogb`, `imaxabs`, `isblank`, `iswblank`, `lgammaf`, `lgammal`, `lgamma`, `llabs`, `llrintf`, `llrintl`, `llrint`, `llroundf`, `llroundl`, `llround`, `log1pf`, `log1pl`, `log1p`, `log2f`, `log2l`, `log2`, `logbf`, `logbl`, `logb`, `lrintf`, `lrintl`, `lrint`, `lroundf`, `lroundl`, `lround`, `nearbyintf`, `nearbyintl`, `nearbyint`, `nextafterf`, `nextafterl`, `nextafter`, `nexttowardf`, `nexttowardl`, `nexttoward`, `remainderf`, `remainderl`, `remainder`, `remquof`, `remquo`, `rintf`, `rintl`, `rint`, `roundf`, `roundl`, `round`, `scalblnf`, `scalblnl`, `scalbln`, `scalbnf`, `scalbnl`, `scalbn`, `snprintf`, `tgammaf`, `tgammal`, `tgamma`, `truncf`, `truncl`, `trunc`, `vscanf`, `vscanf`, `vsprintf` and `vsscanf` are handled as built-in functions except in strict ISO C90 mode (`-ansi` or `-std=c90`).

There are also built-in versions of the ISO C99 functions `acosf`, `acosl`, `asinf`, `asinl`, `atan2f`, `atan2l`, `atanf`, `atanl`, `ceilf`, `ceil`, `cosf`, `coshf`, `coshl`, `cosl`, `expf`, `expl`, `fabsf`, `fabsl`, `floorf`, `floorl`, `fmodf`, `fmodl`, `frexpf`, `frexpl`, `ldexpf`, `ldexpl`, `log10f`, `log10l`, `logf`, `logl`, `modfl`, `modff`, `powf`, `powl`, `sinf`, `sinhf`, `sinhl`, `sinl`, `sqrtf`, `sqrtl`, `tanf`, `tanhf`, `tanh` and `tanl` that are recognized in any mode since ISO C90 reserves these names for the purpose to which ISO C99 puts them. All these functions have corresponding versions prefixed with `__builtin_`.

There are also built-in functions `__builtin_fabsfn`, `__builtin_fabsfnx`, `__builtin_copysignfn` and `__builtin_copysignfnx`, corresponding to the TS 18661-3 functions `fabsfn`, `fabsfnx`, `copysignfn` and `copysignfnx`, for supported types `_Floatn` and `_Floatnx`.

There are also GNU extension functions `clog10`, `clog10f` and `clog10l` which names are reserved by ISO C99 for future use. All these functions have versions prefixed with `__builtin_`.

The ISO C94 functions `iswalnum`, `iswalp`, `iswcntrl`, `iswdigit`, `iswgraph`, `iswlower`, `iswprint`, `iswpunct`, `iswspace`, `iswupper`, `iswxdigit`, `towlower` and `toupper` are handled as built-in functions except in strict ISO C90 mode (`-ansi` or `-std=c90`).

The ISO C90 functions `abort`, `abs`, `acos`, `asin`, `atan2`, `atan`, `calloc`, `ceil`, `cosh`, `cos`, `exit`, `exp`, `fabs`, `floor`, `fmod`, `fprintf`, `fputs`, `free`, `frexp`, `fscanf`, `isalnum`, `isalpha`, `isctrl`, `isdigit`, `isgraph`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `isxdigit`, `tolower`, `toupper`, `labs`, `ldexp`, `log10`, `log`, `malloc`, `memchr`, `memcmp`, `memcpy`, `memset`, `modf`, `pow`, `printf`, `putchar`, `puts`, `realloc`, `scanf`, `sinh`, `sin`, `snprintf`, `sprintf`, `sqrt`, `sscanf`, `strcat`, `strchr`, `strcmp`, `strcpy`, `strcspn`, `strlen`, `strncat`, `strncmp`, `strncpy`, `strpbrk`, `strrchr`, `strspn`, `strstr`, `tanh`, `tan`, `vfprintf`, `vprintf` and `vsprintf` are all recognized as built-in functions unless `-fno-builtin` is specified (or `-fno-builtin-function` is specified for an individual function). All of these functions have corresponding versions prefixed with `__builtin_`.

GCC provides built-in versions of the ISO C99 floating-point comparison macros that avoid raising exceptions for unordered operands. They have the same names as the standard macros (`isgreater`, `isgreaterequal`, `isless`, `islessequal`, `islessgreater`, and `isunordered`), with `__builtin_` prefixed. We intend for a library implementor to be able to simply `#define` each standard macro to its built-in equivalent. In the same fashion, GCC provides `fpclassify`, `iseqsig`, `isfinite`, `isinf_sign`, `isnormal` and `signbit` built-ins used with `__builtin_` prefixed. The `isinf` and `isnan` built-in functions appear both with and without the `__builtin_` prefix. With `-ffinite-math-only` option the `isinf` and `isnan` built-in functions will always return 0.

GCC provides built-in versions of the ISO C99 floating-point rounding and exceptions handling functions `fegetround`, `feclearexcept` and `feraiseexcept`. They may not be available for all targets, and because they need close interaction with libc internal values, they may not be available for all target libcs, but in all cases they will gracefully fallback to libc calls. These built-in functions appear both with and without the `__builtin_` prefix.

7.2 Additional Builtins for Numeric Operations

GCC provides a large set of built-in functions for operating on GCC's extended set of floating-point and integer types (see Section 6.1 [Additional Numeric Types], page 575). The floating-point builtins include functions for building and testing infinities and NaNs. On integer types, there are additional bit manipulation functions, byte-swapping, and CRC functions.

Many of these builtins are type-generic and can operate on any floating-point or integer operand.

7.2.1 Floating-Point Format Builtins

`double __builtin_huge_val (void)` [Built-in Function]
Returns a positive infinity, if supported by the floating-point format, else `DBL_MAX`. This function is suitable for implementing the ISO C macro `HUGE_VAL`.

`float __builtin_huge_valf (void)` [Built-in Function]
Similar to `__builtin_huge_val`, except the return type is `float`.

`long double __builtin_huge_vall (void)` [Built-in Function]
Similar to `__builtin_huge_val`, except the return type is `long double`.

`_Floatn __builtin_huge_valfn (void)` [Built-in Function]
Similar to `__builtin_huge_val`, except the return type is `_Floatn`.

`_Floatn __builtin_huge_valfnx (void)` [Built-in Function]
Similar to `__builtin_huge_val`, except the return type is `_Floatn`.

`int __builtin_fpclassify (int, int, int, int, int, ...)` [Built-in Function]
This built-in implements the C99 `fpclassify` functionality. The first five `int` arguments should be the target library's notion of the possible FP classes and are used for return values. They must be constant values and they must appear in this order: `FP_NAN`, `FP_INFINITE`, `FP_NORMAL`, `FP_SUBNORMAL` and `FP_ZERO`. The ellipsis is for exactly one

floating-point value to classify. GCC treats the last argument as type-generic, which means it does not do default promotion from float to double.

`double __builtin_inf (void)` [Built-in Function]
 Similar to `__builtin_huge_val`, except a warning is generated if the target floating-point format does not support infinities.

`_Decimal32 __builtin_infd32 (void)` [Built-in Function]
 Similar to `__builtin_inf`, except the return type is `_Decimal32`.

`_Decimal64 __builtin_infd64 (void)` [Built-in Function]
 Similar to `__builtin_inf`, except the return type is `_Decimal64`.

`_Decimal128 __builtin_infd128 (void)` [Built-in Function]
 Similar to `__builtin_inf`, except the return type is `_Decimal128`.

`float __builtin_inff (void)` [Built-in Function]
 Similar to `__builtin_inf`, except the return type is `float`. This function is suitable for implementing the ISO C99 macro `INFINITY`.

`long double __builtin_infl (void)` [Built-in Function]
 Similar to `__builtin_inf`, except the return type is `long double`.

`_Floatn __builtin_inffn (void)` [Built-in Function]
 Similar to `__builtin_inf`, except the return type is `_Floatn`.

`_Floatn __builtin_inffnx (void)` [Built-in Function]
 Similar to `__builtin_inf`, except the return type is `_Floatnx`.

`int __builtin_isinf_sign (...)` [Built-in Function]
 Similar to `isinf`, except the return value is -1 for an argument of `-Inf` and 1 for an argument of `+Inf`. Note while the parameter list is an ellipsis, this function only accepts exactly one floating-point argument. GCC treats this parameter as type-generic, which means it does not do default promotion from float to double.

`double __builtin_nan (const char *str)` [Built-in Function]
 This is an implementation of the ISO C99 function `nan`.

Since ISO C99 defines this function in terms of `strtod`, which we do not implement, a description of the parsing is in order. The string is parsed as by `strtoul`; that is, the base is recognized by leading '0' or '0x' prefixes. The number parsed is placed in the significand such that the least significant bit of the number is at the least significant bit of the significand. The number is truncated to fit the significand field provided. The significand is forced to be a quiet NaN.

This function, if given a string literal all of which would have been consumed by `strtoul`, is evaluated early enough that it is considered a compile-time constant.

`_Decimal32 __builtin_nand32 (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the return type is `_Decimal32`.

`_Decimal64 __builtin_nand64 (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the return type is `_Decimal64`.

`_Decimal128 __builtin_nand128 (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the return type is `_Decimal128`.

`float __builtin_nanf (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the return type is `float`.

`long double __builtin_nanl (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the return type is `long double`.

`_Floatn __builtin_nanfn (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the return type is `_Floatn`.

`_Floatnx __builtin_nanfnx (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the return type is `_Floatnx`.

`double __builtin_nans (const char *str)` [Built-in Function]
 Similar to `__builtin_nan`, except the significand is forced to be a signaling NaN.
 The `nans` function is proposed by WG14 N965.

`_Decimal32 __builtin_nansd32 (const char *str)` [Built-in Function]
 Similar to `__builtin_nans`, except the return type is `_Decimal32`.

`_Decimal64 __builtin_nansd64 (const char *str)` [Built-in Function]
 Similar to `__builtin_nans`, except the return type is `_Decimal64`.

`_Decimal128 __builtin_nansd128 (const char *str)` [Built-in Function]
 Similar to `__builtin_nans`, except the return type is `_Decimal128`.

`float __builtin_nansf (const char *str)` [Built-in Function]
 Similar to `__builtin_nans`, except the return type is `float`.

`long double __builtin_nansl (const char *str)` [Built-in Function]
 Similar to `__builtin_nans`, except the return type is `long double`.

`_Floatn __builtin_nansfn (const char *str)` [Built-in Function]
 Similar to `__builtin_nans`, except the return type is `_Floatn`.

`_Floatnx __builtin_nansfnx (const char *str)` [Built-in Function]
 Similar to `__builtin_nans`, except the return type is `_Floatnx`.

`int __builtin_issignaling (...)` [Built-in Function]
 Return non-zero if the argument is a signaling NaN and zero otherwise. Note while the parameter list is an ellipsis, this function only accepts exactly one floating-point argument. GCC treats this parameter as type-generic, which means it does not do default promotion from `float` to `double`. This built-in function can work even without the non-default `-fsignaling-nans` option, although if a signaling NaN is computed, stored or passed as argument to some function other than this built-in in the current translation unit, it is safer to use `-fsignaling-nans`. With `-ffinite-math-only` option this built-in function will always return 0.

double `__builtin_powi` (double, int) [Built-in Function]
float `__builtin_powif` (float, int) [Built-in Function]
long double `__builtin_powil` (long double, int) [Built-in Function]
Returns the first argument raised to the power of the second. Unlike the `pow` function no guarantees about precision and rounding are made.

7.2.2 Bit Operation Builtins

int `__builtin_ffs` (int x) [Built-in Function]
Returns one plus the index of the least significant 1-bit of x, or if x is zero, returns zero.

int `__builtin_clz` (unsigned int x) [Built-in Function]
Returns the number of leading 0-bits in x, starting at the most significant bit position. If x is 0, the result is undefined.

int `__builtin_ctz` (unsigned int x) [Built-in Function]
Returns the number of trailing 0-bits in x, starting at the least significant bit position. If x is 0, the result is undefined.

int `__builtin_clrsb` (int x) [Built-in Function]
Returns the number of leading redundant sign bits in x, i.e. the number of bits following the most significant bit that are identical to it. There are no special cases for 0 or other values.

int `__builtin_popcount` (unsigned int x) [Built-in Function]
Returns the number of 1-bits in x.

int `__builtin_parity` (unsigned int x) [Built-in Function]
Returns the parity of x, i.e. the number of 1-bits in x modulo 2.

int `__builtin_ffsl` (long) [Built-in Function]
Similar to `__builtin_ffs`, except the argument type is long.

int `__builtin_clzl` (unsigned long) [Built-in Function]
Similar to `__builtin_clz`, except the argument type is unsigned long.

int `__builtin_ctzl` (unsigned long) [Built-in Function]
Similar to `__builtin_ctz`, except the argument type is unsigned long.

int `__builtin_clrsbl` (long) [Built-in Function]
Similar to `__builtin_clrsb`, except the argument type is long.

int `__builtin_popcountl` (unsigned long) [Built-in Function]
Similar to `__builtin_popcount`, except the argument type is unsigned long.

int `__builtin_parityl` (unsigned long) [Built-in Function]
Similar to `__builtin_parity`, except the argument type is unsigned long.

int `__builtin_ffsll` (long long) [Built-in Function]
Similar to `__builtin_ffs`, except the argument type is long long.

- `int __builtin_clzll (unsigned long long)` [Built-in Function]
Similar to `__builtin_clz`, except the argument type is `unsigned long long`.
- `int __builtin_ctzll (unsigned long long)` [Built-in Function]
Similar to `__builtin_ctz`, except the argument type is `unsigned long long`.
- `int __builtin_clrsbll (long long)` [Built-in Function]
Similar to `__builtin_clrsb`, except the argument type is `long long`.
- `int __builtin_popcountll (unsigned long long)` [Built-in Function]
Similar to `__builtin_popcount`, except the argument type is `unsigned long long`.
- `int __builtin_parityll (unsigned long long)` [Built-in Function]
Similar to `__builtin_parity`, except the argument type is `unsigned long long`.
- `int __builtin_ffsg (...)` [Built-in Function]
Similar to `__builtin_ffs`, except the argument is type-generic signed integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument.
- `int __builtin_clzg (...)` [Built-in Function]
Similar to `__builtin_clz`, except the argument is type-generic unsigned integer (standard, extended or bit-precise) and there is optional second argument with `int` type. No integral argument promotions are performed on the first argument. If two arguments are specified, and first argument is 0, the result is the second argument. If only one argument is specified and it is 0, the result is undefined.
- `int __builtin_ctzg (...)` [Built-in Function]
Similar to `__builtin_ctz`, except the argument is type-generic unsigned integer (standard, extended or bit-precise) and there is optional second argument with `int` type. No integral argument promotions are performed on the first argument. If two arguments are specified, and first argument is 0, the result is the second argument. If only one argument is specified and it is 0, the result is undefined.
- `int __builtin_clrsbg (...)` [Built-in Function]
Similar to `__builtin_clrsb`, except the argument is type-generic signed integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument.
- `int __builtin_popcountg (...)` [Built-in Function]
Similar to `__builtin_popcount`, except the argument is type-generic unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument.
- `int __builtin_parityg (...)` [Built-in Function]
Similar to `__builtin_parity`, except the argument is type-generic unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument.

`type __builtin_stdlib_bit_ceil (type arg)` [Built-in Function]

The `__builtin_stdlib_bit_ceil` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `arg <= 1 ? (type) 1 : (type) 2 << (prec - 1 - __builtin_clzg ((type) (arg - 1)))` where `prec` is bit width of `type`, except that side-effects in `arg` are evaluated just once.

`type __builtin_stdlib_bit_floor (type arg)` [Built-in Function]

The `__builtin_stdlib_bit_floor` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `arg == 0 ? (type) 0 : (type) 1 << (prec - 1 - __builtin_clzg (arg))` where `prec` is bit width of `type`, except that side-effects in `arg` are evaluated just once.

`unsigned int __builtin_stdlib_bit_width (type arg)` [Built-in Function]

The `__builtin_stdlib_bit_width` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(unsigned int) (prec - __builtin_clzg (arg, prec))` where `prec` is bit width of `type`.

`unsigned int __builtin_stdlib_count_ones (type arg)` [Built-in Function]

The `__builtin_stdlib_count_ones` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(unsigned int) __builtin_popcountg (arg)`

`unsigned int __builtin_stdlib_count_zeros (type arg)` [Built-in Function]

The `__builtin_stdlib_count_zeros` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(unsigned int) __builtin_popcountg ((type) ~arg)`

`unsigned int __builtin_stdlib_first_leading_one (type arg)` [Built-in Function]

The `__builtin_stdlib_first_leading_one` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `__builtin_clzg (arg, -1) + 1U`

`unsigned int __builtin_stdlib_first_leading_zero (type arg)` [Built-in Function]

The `__builtin_stdlib_first_leading_zero` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `__builtin_clzg ((type) ~arg, -1) + 1U`

`unsigned int __builtin_stdcl_first_trailing_one (type arg)` [Built-in Function]

The `__builtin_stdcl_first_trailing_one` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `__builtin_ctzg (arg, -1) + 1U`

`unsigned int __builtin_stdcl_first_trailing_zero (type arg)` [Built-in Function]

The `__builtin_stdcl_first_trailing_zero` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `__builtin_ctzg ((type) ~arg, -1) + 1U`

`unsigned int __builtin_stdcl_has_single_bit (type arg)` [Built-in Function]

The `__builtin_stdcl_has_single_bit` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(_Bool) (__builtin_popcountg (arg) == 1)`

`unsigned int __builtin_stdcl_leading_ones (type arg)` [Built-in Function]

The `__builtin_stdcl_leading_ones` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(unsigned int) __builtin_clzg ((type) ~arg, prec)`

`unsigned int __builtin_stdcl_leading_zeros (type arg)` [Built-in Function]

The `__builtin_stdcl_leading_zeros` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(unsigned int) __builtin_clzg (arg, prec)`

`unsigned int __builtin_stdcl_trailing_ones (type arg)` [Built-in Function]

The `__builtin_stdcl_trailing_ones` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(unsigned int) __builtin_ctzg ((type) ~arg, prec)`

`unsigned int __builtin_stdcl_trailing_zeros (type arg)` [Built-in Function]

The `__builtin_stdcl_trailing_zeros` function is available only in C. It is type-generic, the argument can be any unsigned integer (standard, extended or bit-precise). No integral argument promotions are performed on the argument. It is equivalent to `(unsigned int) __builtin_ctzg (arg, prec)`

`type1 __builtin_stdcl_rotate_left (type1 arg1, type2 arg2)` [Built-in Function]

The `__builtin_stdcl_rotate_left` function is available only in C. It is type-generic, the first argument can be any unsigned integer (standard, extended or bit-precise)

and second argument any signed or unsigned integer or `char`. No integral argument promotions are performed on the arguments. It is equivalent to `(type1)((arg1 << (arg2 % prec)) | (arg1 >> ((-(unsigned type2) arg2) % prec)))` where *prec* is bit width of *type1*, except that side-effects in *arg1* and *arg2* are evaluated just once. The behavior is undefined if *arg2* is negative.

***type1* __builtin_stdcrotate_right (*type1* *arg1*, *type2* *arg2*)** [Built-in Function]

The `__builtin_stdcrotate_right` function is available only in C. It is type-generic, the first argument can be any unsigned integer (standard, extended or bit-precise) and second argument any signed or unsigned integer or `char`. No integral argument promotions are performed on the arguments. It is equivalent to `(type1)((arg1 >> (arg2 % prec)) | (arg1 << ((-(unsigned type2) arg2) % prec)))` where *prec* is bit width of *type1*, except that side-effects in *arg1* and *arg2* are evaluated just once. The behavior is undefined if *arg2* is negative.

7.2.3 Byte-Swapping Builtins

`uint16_t` __builtin_bswap16 (`uint16_t` *x*) [Built-in Function]

Returns *x* with the order of the bytes reversed; for example, `0xabcd` becomes `0xcdab`. Byte here always means exactly 8 bits.

`uint32_t` __builtin_bswap32 (`uint32_t` *x*) [Built-in Function]

Similar to `__builtin_bswap16`, except the argument and return types are 32-bit.

`uint64_t` __builtin_bswap64 (`uint64_t` *x*) [Built-in Function]

Similar to `__builtin_bswap32`, except the argument and return types are 64-bit.

`uint128_t` __builtin_bswap128 (`uint128_t` *x*) [Built-in Function]

Similar to `__builtin_bswap64`, except the argument and return types are 128-bit. Only supported on targets when 128-bit types are supported.

`uint8_t` __builtin_bitreverse8 (`uint8_t` *x*) [Built-in Function]

Returns *x* with all bits reversed.

`uint16_t` __builtin_bitreverse16 (`uint16_t` *x*) [Built-in Function]

Similar to `__builtin_bitreverse8`, except the argument and return types are 16-bit.

`uint32_t` __builtin_bitreverse32 (`uint32_t` *x*) [Built-in Function]

Similar to `__builtin_bitreverse16`, except the argument and return types are 32-bit.

`uint64_t` __builtin_bitreverse64 (`uint64_t` *x*) [Built-in Function]

Similar to `__builtin_bitreverse32`, except the argument and return types are 64-bit.

`uint128_t` __builtin_bitreverse128 (`uint128_t` *x*) [Built-in Function]

Similar to `__builtin_bitreverse64`, except the argument and return types are 128-bit. Only supported on targets when 128-bit types are supported.

7.2.4 CRC Builtins

`uint8_t __builtin_rev_crc8_data8 (uint8_t crc, [Built-in Function]
uint8_t data, uint8_t poly)`

Returns the calculated 8-bit bit-reversed CRC using the initial CRC (8-bit), *data* (8-bit) and the polynomial (8-bit). *crc* is the initial CRC, *data* is the data and *poly* is the polynomial without leading 1. *poly* is required to be a compile-time constant. Table-based or clmul-based CRC may be used for the calculation, depending on the target architecture.

`uint16_t __builtin_rev_crc16_data16 (uint16_t crc, [Built-in Function]
uint16_t data, uint16_t poly)`

Similar to `__builtin_rev_crc8_data8`, except the argument and return types are 16-bit.

`uint16_t __builtin_rev_crc16_data8 (uint16_t crc, [Built-in Function]
uint8_t data, uint16_t poly)`

Similar to `__builtin_rev_crc16_data16`, except the *data* argument type is 8-bit.

`uint32_t __builtin_rev_crc32_data32 (uint32_t crc, [Built-in Function]
uint32_t data, uint32_t poly)`

Similar to `__builtin_rev_crc8_data8`, except the argument and return types are 32-bit and for the CRC calculation may be also used `crc*` machine instruction depending on the target and the polynomial.

`uint32_t __builtin_rev_crc32_data8 (uint32_t crc, [Built-in Function]
uint8_t data, uint32_t poly)`

Similar to `__builtin_rev_crc32_data32`, except the *data* argument type is 8-bit.

`uint32_t __builtin_rev_crc32_data16 (uint32_t crc, [Built-in Function]
uint16_t data, uint32_t poly)`

Similar to `__builtin_rev_crc32_data32`, except the *data* argument type is 16-bit.

`uint64_t __builtin_rev_crc64_data64 (uint64_t crc, [Built-in Function]
uint64_t data, uint64_t poly)`

Similar to `__builtin_rev_crc8_data8`, except the argument and return types are 64-bit.

`uint64_t __builtin_rev_crc64_data8 (uint64_t crc, [Built-in Function]
uint8_t data, uint64_t poly)`

Similar to `__builtin_rev_crc64_data64`, except the *data* argument type is 8-bit.

`uint64_t __builtin_rev_crc64_data16 (uint64_t crc, [Built-in Function]
uint16_t data, uint64_t poly)`

Similar to `__builtin_rev_crc64_data64`, except the *data* argument type is 16-bit.

`uint64_t __builtin_rev_crc64_data32 (uint64_t crc, [Built-in Function]
uint32_t data, uint64_t poly)`

Similar to `__builtin_rev_crc64_data64`, except the *data* argument type is 32-bit.

`uint8_t __builtin_crc8_data8 (uint8_t crc, uint8_t data, uint8_t poly)` [Built-in Function]

Returns the calculated 8-bit bit-forward CRC using the initial CRC (8-bit), *data* (8-bit) and the polynomial (8-bit). *crc* is the initial CRC, *data* is the data and *poly* is the polynomial without leading 1. *poly* is required to be a compile-time constant. Table-based or clmul-based CRC may be used for the calculation, depending on the target architecture.

`uint16_t __builtin_crc16_data16 (uint16_t crc, uint16_t data, uint16_t poly)` [Built-in Function]

Similar to `__builtin_crc8_data8`, except the argument and return types are 16-bit.

`uint16_t __builtin_crc16_data8 (uint16_t crc, uint8_t data, uint16_t poly)` [Built-in Function]

Similar to `__builtin_crc16_data16`, except the *data* argument type is 8-bit.

`uint32_t __builtin_crc32_data32 (uint32_t crc, uint32_t data, uint32_t poly)` [Built-in Function]

Similar to `__builtin_crc8_data8`, except the argument and return types are 32-bit.

`uint32_t __builtin_crc32_data8 (uint32_t crc, uint8_t data, uint32_t poly)` [Built-in Function]

Similar to `__builtin_crc32_data32`, except the *data* argument type is 8-bit.

`uint32_t __builtin_crc32_data16 (uint32_t crc, uint16_t data, uint32_t poly)` [Built-in Function]

Similar to `__builtin_crc32_data32`, except the *data* argument type is 16-bit.

`uint64_t __builtin_crc64_data64 (uint64_t crc, uint64_t data, uint64_t poly)` [Built-in Function]

Similar to `__builtin_crc8_data8`, except the argument and return types are 64-bit.

`uint64_t __builtin_crc64_data8 (uint64_t crc, uint8_t data, uint64_t poly)` [Built-in Function]

Similar to `__builtin_crc64_data64`, except the *data* argument type is 8-bit.

`uint64_t __builtin_crc64_data16 (uint64_t crc, uint16_t data, uint64_t poly)` [Built-in Function]

Similar to `__builtin_crc64_data64`, except the *data* argument type is 16-bit.

`uint64_t __builtin_crc64_data32 (uint64_t crc, uint32_t data, uint64_t poly)` [Built-in Function]

Similar to `__builtin_crc64_data64`, except the *data* argument type is 32-bit.

7.2.5 Built-in Functions to Perform Arithmetic with Overflow Checking

The following built-in functions allow performing simple arithmetic operations together with checking whether the operations overflowed.

```

bool __builtin_add_overflow (type1 a, type2 b, type3    [Built-in Function]
                           *res)
bool __builtin_sadd_overflow (int a, int b, int        [Built-in Function]
                           *res)
bool __builtin_saddl_overflow (long int a, long int    [Built-in Function]
                              b, long int *res)
bool __builtin_saddll_overflow (long long int a,       [Built-in Function]
                               long long int b, long long int *res)
bool __builtin_uadd_overflow (unsigned int a,          [Built-in Function]
                              unsigned int b, unsigned int *res)
bool __builtin_uaddl_overflow (unsigned long int a,    [Built-in Function]
                              unsigned long int b, unsigned long int *res)
bool __builtin_uaddll_overflow (unsigned long long    [Built-in Function]
                               int a, unsigned long long int b, unsigned long long int *res)

```

These built-in functions promote the first two operands into infinite precision signed type and perform addition on those promoted operands. The result is then cast to the type the third pointer argument points to and stored there. If the stored result is equal to the infinite precision result, the built-in functions return **false**, otherwise they return **true**. As the addition is performed in infinite signed precision, these built-in functions have fully defined behavior for all argument values.

The first built-in function allows arbitrary integral types for operands and the result type must be pointer to some integral type other than enumerated or boolean type, the rest of the built-in functions have explicit integer types.

The compiler will attempt to use hardware instructions to implement these built-in functions where possible, like conditional jump on overflow after addition, conditional jump on carry etc.

```

bool __builtin_sub_overflow (type1 a, type2 b, type3    [Built-in Function]
                           *res)
bool __builtin_ssub_overflow (int a, int b, int        [Built-in Function]
                           *res)
bool __builtin_ssubl_overflow (long int a, long int    [Built-in Function]
                              b, long int *res)
bool __builtin_ssubll_overflow (long long int a,       [Built-in Function]
                               long long int b, long long int *res)
bool __builtin_usub_overflow (unsigned int a,          [Built-in Function]
                              unsigned int b, unsigned int *res)
bool __builtin_usubl_overflow (unsigned long int a,    [Built-in Function]
                              unsigned long int b, unsigned long int *res)
bool __builtin_usubll_overflow (unsigned long long    [Built-in Function]
                               int a, unsigned long long int b, unsigned long long int *res)

```

These built-in functions are similar to the add overflow checking built-in functions above, except they perform subtraction, subtract the second argument from the first one, instead of addition.

```

bool __builtin_mul_overflow (type1 a, type2 b, type3 [Built-in Function]
                             *res)
bool __builtin_smul_overflow (int a, int b, int [Built-in Function]
                              *res)
bool __builtin_smull_overflow (long int a, long int [Built-in Function]
                               b, long int *res)
bool __builtin_smulll_overflow (long long int a, [Built-in Function]
                                long long int b, long long int *res)
bool __builtin_umul_overflow (unsigned int a, [Built-in Function]
                              unsigned int b, unsigned int *res)
bool __builtin_umull_overflow (unsigned long int a, [Built-in Function]
                               unsigned long int b, unsigned long int *res)
bool __builtin_umulll_overflow (unsigned long long [Built-in Function]
                                int a, unsigned long long int b, unsigned long long int *res)

```

These built-in functions are similar to the add overflow checking built-in functions above, except they perform multiplication, instead of addition.

The following built-in functions allow checking if simple arithmetic operation would overflow.

```

bool __builtin_add_overflow_p (type1 a, type2 b, [Built-in Function]
                               type3 c)
bool __builtin_sub_overflow_p (type1 a, type2 b, [Built-in Function]
                               type3 c)
bool __builtin_mul_overflow_p (type1 a, type2 b, [Built-in Function]
                               type3 c)

```

These built-in functions are similar to `__builtin_add_overflow`, `__builtin_sub_overflow`, or `__builtin_mul_overflow`, except that they don't store the result of the arithmetic operation anywhere and the last argument is not a pointer, but some expression with integral type other than enumerated or boolean type.

The built-in functions promote the first two operands into infinite precision signed type and perform the corresponding operation on those promoted operands. The result is then cast to the type of the third argument. If the cast result is equal to the infinite precision result, the built-in functions return **false**, otherwise they return **true**. The value of the third argument is ignored, just the side effects in the third argument are evaluated, and no integral argument promotions are performed on the last argument. If the third argument is a bit-field, the type used for the result cast has the precision and signedness of the given bit-field, rather than precision and signedness of the underlying type.

For example, the following macro can be used to portably check, at compile-time, whether or not adding two constant integers will overflow, and perform the addition only when it is known to be safe and not to trigger a `-Woverflow` warning.

```

#define INT_ADD_OVERFLOW_P(a, b) \
    __builtin_add_overflow_p (a, b, (__typeof__ ((a) + (b))) 0)

enum {
    A = INT_MAX, B = 3,
    C = INT_ADD_OVERFLOW_P (A, B) ? 0 : A + B,

```

```

    D = __builtin_add_overflow_p (1, SCHAR_MAX, (signed char) 0)
};

```

The compiler will attempt to use hardware instructions to implement these built-in functions where possible, like conditional jump on overflow after addition, conditional jump on carry etc.

```

unsigned int __builtin_addc (unsigned int a,           [Built-in Function]
                           unsigned int b, unsigned int carry_in, unsigned int
                           *carry_out)
unsigned long int __builtin_addcl (unsigned long int  [Built-in Function]
                                  a, unsigned long int b, unsigned int carry_in, unsigned long
                                  int *carry_out)
unsigned long long int __builtin_addc11 (unsigned      [Built-in Function]
                                         long long int a, unsigned long long int b, unsigned long
                                         long int carry_in, unsigned long long int *carry_out)

```

These built-in functions are equivalent to:

```

({ __typeof__ (a) s; \
   __typeof__ (a) c1 = __builtin_add_overflow (a, b, &s); \
   __typeof__ (a) c2 = __builtin_add_overflow (s, carry_in, &s); \
   *(carry_out) = c1 | c2; \
   s; })

```

i.e. they add 3 unsigned values, set what the last argument points to to 1 if any of the two additions overflowed (otherwise 0) and return the sum of those 3 unsigned values. Note, while all the first 3 arguments can have arbitrary values, better code will be emitted if one of them (preferably the third one) has only values 0 or 1 (i.e. carry-in).

```

unsigned int __builtin_subc (unsigned int a,           [Built-in Function]
                           unsigned int b, unsigned int carry_in, unsigned int
                           *carry_out)
unsigned long int __builtin_subcl (unsigned long int  [Built-in Function]
                                  a, unsigned long int b, unsigned int carry_in, unsigned long
                                  int *carry_out)
unsigned long long int __builtin_subc11 (unsigned      [Built-in Function]
                                         long long int a, unsigned long long int b, unsigned long
                                         long int carry_in, unsigned long long int *carry_out)

```

These built-in functions are equivalent to:

```

({ __typeof__ (a) s; \
   __typeof__ (a) c1 = __builtin_sub_overflow (a, b, &s); \
   __typeof__ (a) c2 = __builtin_sub_overflow (s, carry_in, &s); \
   *(carry_out) = c1 | c2; \
   s; })

```

i.e. they subtract 2 unsigned values from the first unsigned value, set what the last argument points to to 1 if any of the two subtractions overflowed (otherwise 0) and return the result of the subtractions. Note, while all the first 3 arguments can have arbitrary values, better code will be emitted if one of them (preferably the third one) has only values 0 or 1 (i.e. carry-in).

7.3 Builtins for Stack Allocation

`void * __builtin_alloca (size_t size)` [Built-in Function]

The `__builtin_alloca` function must be called at block scope. The function allocates an object *size* bytes large on the stack of the calling function. The object is aligned on the default stack alignment boundary for the target determined by the `__BIGGEST_ALIGNMENT__` macro. The `__builtin_alloca` function returns a pointer to the first byte of the allocated object. The lifetime of the allocated object ends just before the calling function returns to its caller. This is so even when `__builtin_alloca` is called within a nested block.

For example, the following function allocates eight objects of *n* bytes each on the stack, storing a pointer to each in consecutive elements of the array *a*. It then passes the array to function *g* which can safely use the storage pointed to by each of the array elements.

```
void f (unsigned n)
{
    void *a [8];
    for (int i = 0; i != 8; ++i)
        a [i] = __builtin_alloca (n);

    g (a, n);    // safe
}
```

Since the `__builtin_alloca` function doesn't validate its argument it is the responsibility of its caller to make sure the argument doesn't cause it to exceed the stack size limit. The `__builtin_alloca` function is provided to make it possible to allocate on the stack arrays of bytes with an upper bound that may be computed at run time. Since C99 Variable Length Arrays offer similar functionality under a portable, more convenient, and safer interface they are recommended instead, in both C99 and C++ programs where GCC provides them as an extension. See Section 6.2.1 [Variable Length], page 581, for details.

`void * __builtin_alloca_with_align (size_t size, size_t alignment)` [Built-in Function]

The `__builtin_alloca_with_align` function must be called at block scope. The function allocates an object *size* bytes large on the stack of the calling function. The allocated object is aligned on the boundary specified by the argument *alignment* whose unit is given in bits (not bytes). The *size* argument must be positive and not exceed the stack size limit. The *alignment* argument must be a constant integer expression that evaluates to a power of 2 greater than or equal to `CHAR_BIT` and less than some unspecified maximum. Invocations with other values are rejected with an error indicating the valid bounds. The function returns a pointer to the first byte of the allocated object. The lifetime of the allocated object ends at the end of the block in which the function was called. The allocated storage is released no later than just before the calling function returns to its caller, but may be released at the end of the block in which the function was called.

For example, in the following function the call to *g* is unsafe because when *overallign* is non-zero, the space allocated by `__builtin_alloca_with_align` may have been released at the end of the *if* statement in which it was called.

```

void f (unsigned n, bool overalign)
{
    void *p;
    if (overalign)
        p = __builtin_alloca_with_align (n, 64 /* bits */);
    else
        p = __builtin_alloc (n);

    g (p, n);    // unsafe
}

```

Since the `__builtin_alloca_with_align` function doesn't validate its *size* argument it is the responsibility of its caller to make sure the argument doesn't cause it to exceed the stack size limit. The `__builtin_alloca_with_align` function is provided to make it possible to allocate on the stack overaligned arrays of bytes with an upper bound that may be computed at run time. Since C99 Variable Length Arrays offer the same functionality under a portable, more convenient, and safer interface they are recommended instead, in both C99 and C++ programs where GCC provides them as an extension. See Section 6.2.1 [Variable Length], page 581, for details.

```

void * __builtin_alloca_with_align_and_max (size_t      [Built-in Function]
      size, size_t alignment, size_t max_size)

```

Similar to `__builtin_alloca_with_align` but takes an extra argument specifying an upper bound for *size* in case its value cannot be computed at compile time, for use by `-fstack-usage`, `-Wstack-usage` and `-Walloca-larger-than`. *max_size* must be a constant integer expression, it has no effect on code generation and no attempt is made to check its compatibility with *size*.

7.4 Nonlocal Gotos

GCC provides the built-in functions `__builtin_setjmp` and `__builtin_longjmp` which are similar to, but not interchangeable with, the C library functions `setjmp` and `longjmp`. The built-in versions are used internally by GCC's libraries to implement exception handling on some targets. You should use the standard C library functions declared in `<setjmp.h>` in user code instead of the builtins.

The built-in versions of these functions use GCC's normal mechanisms to save and restore registers using the stack on function entry and exit. The jump buffer argument *buf* holds only the information needed to restore the stack frame, rather than the entire set of saved register values.

An important caveat is that GCC arranges to save and restore only those registers known to the specific architecture variant being compiled for. This can make `__builtin_setjmp` and `__builtin_longjmp` more efficient than their library counterparts in some cases, but it can also cause incorrect and mysterious behavior when mixing with code that uses the full register set.

You should declare the jump buffer argument *buf* to the built-in functions as:

```

#include <stdint.h>
intptr_t buf[5];

```

int __builtin_setjmp (intptr_t *buf) [Built-in Function]

This function saves the current stack context in *buf*. `__builtin_setjmp` returns 0 when returning directly, and 1 when returning from `__builtin_longjmp` using the same *buf*.

void __builtin_longjmp (intptr_t *buf, int val) [Built-in Function]

This function restores the stack context in *buf*, saved by a previous call to `__builtin_setjmp`. After `__builtin_longjmp` is finished, the program resumes execution as if the matching `__builtin_setjmp` returns the value *val*, which must be 1.

Because `__builtin_longjmp` depends on the function return mechanism to restore the stack context, it cannot be called from the same function calling `__builtin_setjmp` to initialize *buf*. It can only be called from a function called (directly or indirectly) from the function calling `__builtin_setjmp`.

7.5 Constructing Function Calls

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

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

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

void * __builtin_apply_args () [Built-in Function]

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

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

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

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

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

This function returns a pointer to data describing how to return whatever value is returned by *function*. The data is saved in a block of memory allocated on the stack. It is not always simple to compute the proper value for *size*. The value is used by `__builtin_apply` to compute the amount of data that should be pushed on the stack and copied from the incoming argument area.

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

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

`__builtin_va_arg_pack ()` [Built-in Function]

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

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

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

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

```
#ifdef __OPTIMIZE__
extern inline __attribute__ ((__gnu_inline__)) int
myopen (const char *path, int oflag, ...)
{
    if (__builtin_va_arg_pack_len () > 1)
        warn_open_too_many_arguments ();

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

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

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

`type __builtin_call_with_static_chain (call_exp, [Built-in Function]
pointer_exp)`

The *call_exp* expression must be a function call, and the *pointer_exp* expression must be a pointer. The *pointer_exp* is passed to the function call in the target's static chain location. The result of builtin is the result of the function call.

Note: This builtin is only available for C. This builtin can be used to call Go closures from C.

7.6 Getting the Return or Frame Address of a Function

These functions may be used to get information about the callers of a function.

`void * __builtin_return_address (unsigned int level) [Built-in Function]`

This function returns the return address of the current function, or of one of its callers. The *level* argument is number of frames to scan up the call stack. A value of 0 yields the return address of the current function, a value of 1 yields the return address of the caller of the current function, and so forth. When inlining the expected behavior is that the function returns the address of the function that is returned to. To work around this behavior use the `noinline` function attribute.

The *level* argument must be a constant integer.

On some machines it may be impossible to determine the return address of any function other than the current one; in such cases, or when the top of the stack has been reached, this function returns an unspecified value. In addition, `__builtin_frame_address` may be used to determine if the top of the stack has been reached.

Additional post-processing of the returned value may be needed, see `__builtin_extract_return_addr`.

The stored representation of the return address in memory may be different from the address returned by `__builtin_return_address`. For example, on AArch64 the stored address may be mangled with return address signing whereas the address returned by `__builtin_return_address` is not.

Calling this function with a nonzero argument can have unpredictable effects, including crashing the calling program. As a result, calls that are considered unsafe are diagnosed when the `-Wframe-address` option is in effect. Such calls should only be made in debugging situations.

On targets where code addresses are representable as `void *`,

```
void *addr = __builtin_extract_return_addr (__builtin_return_address (0));
```

gives the code address where the current function would return. For example, such an address may be used with `dladdr` or other interfaces that work with code addresses.

`void * __builtin_extract_return_addr (void *addr) [Built-in Function]`

The address as returned by `__builtin_return_address` may have to be fed through this function to get the actual encoded address. For example, on the 31-bit S/390 platform the highest bit has to be masked out, or on SPARC platforms an offset has to be added for the true next instruction to be executed.

If no fixup is needed, this function simply passes through *addr*.

`void * __builtin_frob_return_addr (void *addr)` [Built-in Function]
 This function does the reverse of `__builtin_extract_return_addr`.

`void * __builtin_frame_address (unsigned int level)` [Built-in Function]
 This function is similar to `__builtin_return_address`, but it returns the address of the function frame rather than the return address of the function. Calling `__builtin_frame_address` with a value of 0 yields the frame address of the current function, a value of 1 yields the frame address of the caller of the current function, and so forth. The frame is the area on the stack that holds local variables and saved registers. The frame address is normally the address of the first word pushed on to the stack by the function. However, the exact definition depends upon the processor and the calling convention. If the processor has a dedicated frame pointer register, and the function has a frame, then `__builtin_frame_address` returns the value of the frame pointer register.

On some machines it may be impossible to determine the frame address of any function other than the current one; in such cases, or when the top of the stack has been reached, this function returns 0 if the first frame pointer is properly initialized by the startup code.

Calling this function with a nonzero argument can have unpredictable effects, including crashing the calling program. As a result, calls that are considered unsafe are diagnosed when the `-Wframe-address` option is in effect. Such calls should only be made in debugging situations.

`void * __builtin_stack_address ()` [Built-in Function]
 This function returns the stack pointer register, offset by `STACK_ADDRESS_OFFSET` if that's defined.

Conceptually, the returned address returned by this built-in function is the boundary between the stack area allocated for use by its caller, and the area that could be modified by a function call, that the caller could safely zero-out before or after (but not during) the call sequence.

Arguments for a callee may be preallocated as part of the caller's stack frame, or allocated on a per-call basis, depending on the target, so they may be on either side of this boundary.

Even if the stack pointer is biased, the result is not. The register save area on SPARC is regarded as modifiable by calls, rather than as allocated for use by the caller function, since it is never in use while the caller function itself is running.

Red zones that only leaf functions could use are also regarded as modifiable by calls, rather than as allocated for use by the caller. This is only theoretical, since leaf functions do not issue calls, but a constant offset makes this built-in function more predictable.

7.7 Stack scrubbing internal interfaces

Stack scrubbing involves cooperation between a `strub` context, i.e., a function whose stack frame is to be zeroed-out, and its callers. The caller initializes a stack watermark, the `strub` context updates the watermark according to its stack use, and the caller zeroes it out once it regains control, whether by the callee's returning or by an exception.

Each of these steps is performed by a different builtin function call. Calls to these builtins are introduced automatically, in response to **strub** attributes and command-line options; they are not expected to be explicitly called by source code.

The functions that implement the builtins are available in `libgcc` but, depending on optimization levels, they are expanded internally, adjusted to account for inlining, and sometimes combined/deferred (e.g. passing the caller-supplied watermark on to callees, refraining from erasing stack areas that the caller will) to enable tail calls and to optimize for code size.

void __builtin__strub_enter (void **wmptr) [Built-in Function]

This function initializes a stack *watermark* variable with the current top of the stack. A call to this builtin function is introduced before entering a **strub** context. It remains as a function call if optimization is not enabled.

void __builtin__strub_update (void **wmptr) [Built-in Function]

This function updates a stack *watermark* variable with the current top of the stack, if it tops the previous watermark. A call to this builtin function is inserted within **strub** contexts, whenever additional stack space may have been used. It remains as a function call at optimization levels lower than 2.

void __builtin__strub_leave (void **wmptr) [Built-in Function]

This function overwrites the memory area between the current top of the stack, and the *watermarked* address. A call to this builtin function is inserted after leaving a **strub** context. It remains as a function call at optimization levels lower than 3, and it is guarded by a condition at level 2.

7.8 Using Vector Instructions through Built-in Functions

On some targets, the instruction set contains SIMD vector instructions which operate on multiple values contained in one large register at the same time. For example, on the x86 the MMX, 3DNow! and SSE extensions can be used this way.

The first step in using these extensions is to provide the necessary data types. This should be done using an appropriate **typedef**:

```
typedef int v4si __attribute__((vector_size (16)));
```

The **int** type specifies the *base type* (which can be a **typedef**), while the attribute specifies the vector size for the variable, measured in bytes. For example, the declaration above causes the compiler to set the mode for the **v4si** type to be 16 bytes wide and divided into **int** sized units. For a 32-bit **int** this means a vector of 4 units of 4 bytes, and the corresponding mode of **foo** is **V4SI**.

The **vector_size** attribute is only applicable to integral and floating scalars, although arrays, pointers, and function return values are allowed in conjunction with this construct. Only sizes that are positive power-of-two multiples of the base type size are currently allowed.

All the basic integer types can be used as base types, both as signed and as unsigned: **char**, **short**, **int**, **long**, **long long**. In addition, **float** and **double** can be used to build floating-point vector types.

Specifying a combination that is not valid for the current architecture causes GCC to synthesize the instructions using a narrower mode. For example, if you specify a variable of type `V4SI` and your architecture does not allow for this specific SIMD type, GCC produces code that uses 4 `SIs`.

The types defined in this manner can be used with a subset of normal C operations. Currently, GCC allows using the following operators on these types: `+`, `-`, `*`, `/`, unary minus, `^`, `|`, `&`, `~`, `%`.

The operations behave like C++ `valarrays`. Addition is defined as the addition of the corresponding elements of the operands. For example, in the code below, each of the 4 elements in `a` is added to the corresponding 4 elements in `b` and the resulting vector is stored in `c`.

```
typedef int v4si __attribute__((vector_size (16)));

v4si a, b, c;

c = a + b;
```

Subtraction, multiplication, division, and the logical operations operate in a similar manner. Likewise, the result of using the unary minus or complement operators on a vector type is a vector whose elements are the negative or complemented values of the corresponding elements in the operand.

It is possible to use shifting operators `<<`, `>>` on integer-type vectors. The operation is defined as following: `{a0, a1, ..., an} >> {b0, b1, ..., bn} == {a0 >> b0, a1 >> b1, ..., an >> bn}`. Unlike OpenCL, values of `b` are not implicitly taken modulo bit width of the base type `B`, and the behavior is undefined if any `bi` is greater than or equal to `B`.

In contrast to scalar operations in C and C++, operands of integer vector operations do not undergo integer promotions.

Operands of binary vector operations must have the same number of elements.

For convenience, it is allowed to use a binary vector operation where one operand is a scalar. In that case the compiler transforms the scalar operand into a vector where each element is the scalar from the operation. The transformation happens only if the scalar could be safely converted to the vector-element type. Consider the following code.

```
typedef int v4si __attribute__((vector_size (16)));

v4si a, b, c;
long l;

a = b + 1;    /* a = b + {1,1,1,1}; */
a = 2 * b;    /* a = {2,2,2,2} * b; */

a = l + a;    /* Error, cannot convert long to int. */
```

Vectors can be subscripted as if the vector were an array with the same number of elements and base type. Out of bound accesses invoke undefined behavior at run time. Warnings for out of bound accesses for vector subscription can be enabled with `-Warray-bounds`.

Vector comparison is supported with standard comparison operators: `==`, `!=`, `<`, `<=`, `>`, and `>=`. Comparison operands can be vector expressions of integer-type or real-type. Comparison between integer-type vectors and real-type vectors are not supported. The result of the comparison is a vector of the same width and number of elements as the comparison operands with a signed integral element type.

Vectors are compared elementwise producing 0 when the comparison is false and -1 (a constant of the appropriate type where all bits are set) otherwise. Consider the following example:

```
typedef int v4si __attribute__((vector_size (16)));

v4si a = {1,2,3,4};
v4si b = {3,2,1,4};
v4si c;

c = a > b;    /* The result would be {0, 0,-1, 0} */
c = a == b;   /* The result would be {0,-1, 0,-1} */
```

In C++, the ternary operator `?:` is available. `a?b:c`, where `b` and `c` are vectors of the same type and `a` is an integer vector with the same number of elements of the same size as `b` and `c`, computes all three arguments and creates a vector `{a[0]?b[0]:c[0], a[1]?b[1]:c[1], ...}`. Note that unlike in OpenCL, `a` is thus interpreted as `a != 0` and not `a < 0`. As in the case of binary operations, this syntax is also accepted when one of `b` or `c` is a scalar that is then transformed into a vector. If both `b` and `c` are scalars and the type of `true?b:c` has the same size as the element type of `a`, then `b` and `c` are converted to a vector type whose elements have this type and with the same number of elements as `a`.

In C++, the logic operators `!`, `&&`, `||` are available for vectors. `!v` is equivalent to `v == 0`, `a && b` is equivalent to `a!=0 & b!=0` and `a || b` is equivalent to `a!=0 | b!=0`. For mixed operations between a scalar `s` and a vector `v`, `s && v` is equivalent to `s?v!=0:0` (the evaluation is short-circuit) and `v && s` is equivalent to `v!=0 & (s?-1:0)`.

Vector shuffling is available using functions `__builtin_shuffle (vec, mask)` and `__builtin_shuffle (vec0, vec1, mask)`. Both functions construct a permutation of elements from one or two vectors and return a vector of the same type as the input vector(s). The `mask` is an integral vector with the same width (`W`) and element count (`N`) as the output vector.

The elements of the input vectors are numbered in memory ordering of `vec0` beginning at 0 and `vec1` beginning at `N`. The elements of `mask` are considered modulo `N` in the single-operand case and modulo `2 * N` in the two-operand case.

Consider the following example,

```
typedef int v4si __attribute__((vector_size (16)));

v4si a = {1,2,3,4};
v4si b = {5,6,7,8};
v4si mask1 = {0,1,1,3};
v4si mask2 = {0,4,2,5};
v4si res;

res = __builtin_shuffle (a, mask1);    /* res is {1,2,2,4} */
res = __builtin_shuffle (a, b, mask2); /* res is {1,5,3,6} */
```

Note that `__builtin_shuffle` is intentionally semantically compatible with the OpenCL `shuffle` and `shuffle2` functions.

You can declare variables and use them in function calls and returns, as well as in assignments and some casts. You can specify a vector type as a return type for a function. Vector types can also be used as function arguments. It is possible to cast from one vector type to another, provided they are of the same size (in fact, you can also cast vectors to and from other data types of the same size).

You cannot operate between vectors of different lengths or different signedness without a cast.

Vector shuffling is available using the `__builtin_shufflevector (vec1, vec2, index...)` function. `vec1` and `vec2` must be expressions with vector type with a compatible element type. The result of `__builtin_shufflevector` is a vector with the same element type as `vec1` and `vec2` but that has an element count equal to the number of indices specified.

The *index* arguments are a list of integers that specify the elements indices of the first two vectors that should be extracted and returned in a new vector. These element indices are numbered sequentially starting with the first vector, continuing into the second vector. An index of -1 can be used to indicate that the corresponding element in the returned vector is a don't care and can be freely chosen to optimized the generated code sequence performing the shuffle operation.

Consider the following example,

```
typedef int v4si __attribute__((vector_size (16)));
typedef int v8si __attribute__((vector_size (32)));

v8si a = {1,-2,3,-4,5,-6,7,-8};
v4si b = __builtin_shufflevector (a, a, 0, 2, 4, 6); /* b is {1,3,5,7} */
v4si c = {-2,-4,-6,-8};
v8si d = __builtin_shufflevector (c, b, 4, 0, 5, 1, 6, 2, 7, 3); /* d is a */
```

Vector conversion is available using the `__builtin_convertvector (vec, vectype)` function. `vec` must be an expression with integral or floating vector type and *vectype* an integral or floating vector type with the same number of elements. The result has *vectype* type and value of a C cast of every element of `vec` to the element type of *vectype*.

Consider the following example,

```
typedef int v4si __attribute__((vector_size (16)));
typedef float v4sf __attribute__((vector_size (16)));
typedef double v4df __attribute__((vector_size (32)));
typedef unsigned long long v4di __attribute__((vector_size (32)));

v4si a = {1,-2,3,-4};
v4sf b = {1.5f,-2.5f,3.f,7.f};
v4di c = {1ULL,5ULL,0ULL,10ULL};
v4sf d = __builtin_convertvector (a, v4sf); /* d is {1.f,-2.f,3.f,-4.f} */
/* Equivalent of:
    v4sf d = { (float)a[0], (float)a[1], (float)a[2], (float)a[3] }; */
v4df e = __builtin_convertvector (a, v4df); /* e is {1.,-2.,3.,-4.} */
v4df f = __builtin_convertvector (b, v4df); /* f is {1.5,-2.5,3.,7.} */
v4si g = __builtin_convertvector (f, v4si); /* g is {1,-2,3,7} */
v4si h = __builtin_convertvector (c, v4si); /* h is {1,5,0,10} */
```

Sometimes it is desirable to write code using a mix of generic vector operations (for clarity) and machine-specific vector intrinsics (to access vector instructions that are not exposed via generic built-ins). On x86, intrinsic functions for integer vectors typically use the same vector type `__m128i` irrespective of how they interpret the vector, making it necessary to cast their arguments and return values from/to other vector types. In C, you can make use of a union type:

```
#include <immintrin.h>

typedef unsigned char u8x16 __attribute__((vector_size (16)));
```

```
typedef unsigned int  u32x4 __attribute__((vector_size (16)));

typedef union {
    __m128i mm;
    u8x16   u8;
    u32x4   u32;
} v128;
```

for variables that can be used with both built-in operators and x86 intrinsics:

```
v128 x, y = { 0 };
memcpy (&x, ptr, sizeof x);
y.u8  += 0x80;
x.mm  = _mm_adds_epu8 (x.mm, y.mm);
x.u32 &= 0xffffffff;

/* Instead of a variable, a compound literal may be used to pass the
   return value of an intrinsic call to a function expecting the union: */
v128 foo (v128);
x = foo ((v128) {_mm_adds_epu8 (x.mm, y.mm)});
```

7.9 Builtins for Atomic Memory Access

GCC supports two sets of builtins for atomic memory access primitives. The `__atomic` builtins provide the underlying support for the C++11 atomic operations library, and are the currently-recommended interface when the C++11 library functions cannot be used directly. The `__sync` builtins implement the specification from the Intel IA64 pSABI and are supported primarily for use in legacy code.

7.9.1 Built-in Functions for Memory Model Aware Atomic Operations

The following built-in functions approximately match the requirements for the C++11 memory model. They are all identified by being prefixed with `__atomic` and most are overloaded so that they work with multiple types.

These functions are intended to replace the legacy `__sync` builtins. The main difference is that the memory order that is requested is a parameter to the functions. New code should always use the `__atomic` builtins rather than the `__sync` builtins.

Note that the `__atomic` builtins assume that programs will conform to the C++11 memory model. In particular, they assume that programs are free of data races. See the C++11 standard for detailed requirements.

The `__atomic` builtins can be used with any integral scalar or pointer type that is 1, 2, 4, or 8 bytes in length. 16-byte integral types are also allowed if `__int128` (see Section 6.1.1 [__int128], page 575) is supported by the architecture.

The four non-arithmetic functions (load, store, exchange, and compare_exchange) all have a generic version as well. This generic version works on any data type. It uses the lock-free built-in function if the specific data type size makes that possible; otherwise, an external call is left to be resolved at run time. This external call is the same format with the addition of a `size_t` parameter inserted as the first parameter indicating the size of the object being pointed to. All objects must be the same size.

There are 6 different memory orders that can be specified. These map to the C++11 memory orders with the same names, see the C++11 standard or the GCC wiki on atomic syn-

chronization (<https://gcc.gnu.org/wiki/Atomic/GCCMM/AtomicSync>) for detailed definitions. Individual targets may also support additional memory orders for use on specific architectures. Refer to the target documentation for details of these.

An atomic operation can both constrain code motion and be mapped to hardware instructions for synchronization between threads (e.g., a fence). To which extent this happens is controlled by the memory orders, which are listed here in approximately ascending order of strength. The description of each memory order is only meant to roughly illustrate the effects and is not a specification; see the C++11 memory model for precise semantics.

`__ATOMIC_RELAXED`

Implies no inter-thread ordering constraints.

`__ATOMIC_CONSUME`

This is currently implemented using the stronger `__ATOMIC_ACQUIRE` memory order because of a deficiency in C++11's semantics for `memory_order_consume`.

`__ATOMIC_ACQUIRE`

Creates an inter-thread happens-before constraint from the release (or stronger) semantic store to this acquire load. Can prevent hoisting of code to before the operation.

`__ATOMIC_RELEASE`

Creates an inter-thread happens-before constraint to acquire (or stronger) semantic loads that read from this release store. Can prevent sinking of code to after the operation.

`__ATOMIC_ACQ_REL`

Combines the effects of both `__ATOMIC_ACQUIRE` and `__ATOMIC_RELEASE`.

`__ATOMIC_SEQ_CST`

Enforces total ordering with all other `__ATOMIC_SEQ_CST` operations.

Note that in the C++11 memory model, *fences* (e.g., `'__atomic_thread_fence'`) take effect in combination with other atomic operations on specific memory locations (e.g., atomic loads); operations on specific memory locations do not necessarily affect other operations in the same way.

Target architectures are encouraged to provide their own patterns for each of the atomic built-in functions. If no target is provided, the original non-memory model set of `'__sync'` atomic built-in functions are used, along with any required synchronization fences surrounding it in order to achieve the proper behavior. Execution in this case is subject to the same restrictions as those built-in functions.

If there is no pattern or mechanism to provide a lock-free instruction sequence, a call is made to an external routine with the same parameters to be resolved at run time.

When implementing patterns for these built-in functions, the memory order parameter can be ignored as long as the pattern implements the most restrictive `__ATOMIC_SEQ_CST` memory order. Any of the other memory orders execute correctly with this memory order but they may not execute as efficiently as they could with a more appropriate implementation of the relaxed requirements.

Note that the C++11 standard allows for the memory order parameter to be determined at run time rather than at compile time. These built-in functions map any run-time value

to `__ATOMIC_SEQ_CST` rather than invoke a runtime library call or inline a switch statement. This is standard compliant, safe, and the simplest approach for now.

The memory order parameter is a signed int, but only the lower 16 bits are reserved for the memory order. The remainder of the signed int is reserved for target use and should be 0. Use of the predefined atomic values ensures proper usage.

The x86 architecture supports additional memory ordering modifiers to mark critical sections for hardware lock elision. These modifiers can be bitwise or'ed with a standard memory order to atomic intrinsics.

`__ATOMIC_HLE_ACQUIRE`

Start lock elision on a lock variable. Memory order must be `__ATOMIC_ACQUIRE` or stronger.

`__ATOMIC_HLE_RELEASE`

End lock elision on a lock variable. Memory order must be `__ATOMIC_RELEASE` or stronger.

When a lock acquire fails, it is required for good performance to abort the transaction quickly. This can be done with a `_mm_pause`.

```
#include <immintrin.h> // For _mm_pause

int lockvar;

/* Acquire lock with lock elision */
while (__atomic_exchange_n(&lockvar, 1, __ATOMIC_ACQUIRE|__ATOMIC_HLE_ACQUIRE))
    _mm_pause(); /* Abort failed transaction */
...
/* Free lock with lock elision */
__atomic_store_n(&lockvar, 0, __ATOMIC_RELEASE|__ATOMIC_HLE_RELEASE);
```

`type __atomic_load_n (type *ptr, int memorder)` [Built-in Function]
This built-in function implements an atomic load operation. It returns the contents of *ptr*.

The valid memory order variants are `__ATOMIC_RELAXED`, `__ATOMIC_SEQ_CST`, `__ATOMIC_ACQUIRE`, and `__ATOMIC_CONSUME`.

`void __atomic_load (type *ptr, type *ret, int memorder)` [Built-in Function]

This is the generic version of an atomic load. It returns the contents of *ptr* in *ret*.

`void __atomic_store_n (type *ptr, type val, int memorder)` [Built-in Function]

This built-in function implements an atomic store operation. It writes *val* into *ptr*.

The valid memory order variants are `__ATOMIC_RELAXED`, `__ATOMIC_SEQ_CST`, and `__ATOMIC_RELEASE`.

`void __atomic_store (type *ptr, type *val, int memorder)` [Built-in Function]

This is the generic version of an atomic store. It stores the value of *val* into *ptr*.

```
type __atomic_exchange_n (type *ptr, type val, int memorder) [Built-in Function]
```

This built-in function implements an atomic exchange operation. It writes *val* into **ptr*, and returns the previous contents of **ptr*.

All memory order variants are valid.

```
void __atomic_exchange (type *ptr, type *val, type *ret, int memorder) [Built-in Function]
```

This is the generic version of an atomic exchange. It stores the contents of **val* into **ptr*. The original value of **ptr* is copied into **ret*.

```
bool __atomic_compare_exchange_n (type *ptr, type *expected, type desired, bool weak, int success_memorder, int failure_memorder) [Built-in Function]
```

This built-in function implements an atomic compare and exchange operation. This compares the contents of **ptr* with the contents of **expected*. If equal, the operation is a *read-modify-write* operation that writes *desired* into **ptr*. If they are not equal, the operation is a *read* and the current contents of **ptr* are written into **expected*. *weak* is **true** for weak compare_exchange, which may fail spuriously, and **false** for the strong variation, which never fails spuriously. Many targets only offer the strong variation and ignore the parameter. When in doubt, use the strong variation.

If *desired* is written into **ptr* then **true** is returned and memory is affected according to the memory order specified by *success_memorder*. There are no restrictions on what memory order can be used here.

Otherwise, **false** is returned and memory is affected according to *failure_memorder*. This memory order cannot be `__ATOMIC_RELEASE` nor `__ATOMIC_ACQ_REL`. It also cannot be a stronger order than that specified by *success_memorder*.

```
bool __atomic_compare_exchange (type *ptr, type *expected, type *desired, bool weak, int success_memorder, int failure_memorder) [Built-in Function]
```

This built-in function implements the generic version of `__atomic_compare_exchange`. The function is virtually identical to `__atomic_compare_exchange_n`, except the desired value is also a pointer.

```
type __atomic_add_fetch (type *ptr, type val, int memorder) [Built-in Function]
```

```
type __atomic_sub_fetch (type *ptr, type val, int memorder) [Built-in Function]
```

```
type __atomic_and_fetch (type *ptr, type val, int memorder) [Built-in Function]
```

```
type __atomic_xor_fetch (type *ptr, type val, int memorder) [Built-in Function]
```

```
type __atomic_or_fetch (type *ptr, type val, int memorder) [Built-in Function]
```

`type __atomic_nand_fetch (type *ptr, type val, int memorder)` [Built-in Function]

These built-in functions perform the operation suggested by the name, and return the result of the operation. Operations on pointer arguments are performed as if the operands were of the `uintptr_t` type. That is, they are not scaled by the size of the type to which the pointer points.

```
{ *ptr op= val; return *ptr; }
{ *ptr = ~(*ptr & val); return *ptr; } // nand
```

The object pointed to by the first argument must be of integer or pointer type. It must not be a boolean type. All memory orders are valid.

`type __atomic_fetch_add (type *ptr, type val, int memorder)` [Built-in Function]

`type __atomic_fetch_sub (type *ptr, type val, int memorder)` [Built-in Function]

`type __atomic_fetch_and (type *ptr, type val, int memorder)` [Built-in Function]

`type __atomic_fetch_xor (type *ptr, type val, int memorder)` [Built-in Function]

`type __atomic_fetch_or (type *ptr, type val, int memorder)` [Built-in Function]

`type __atomic_fetch_nand (type *ptr, type val, int memorder)` [Built-in Function]

These built-in functions perform the operation suggested by the name, and return the value that had previously been in `*ptr`. Operations on pointer arguments are performed as if the operands were of the `uintptr_t` type. That is, they are not scaled by the size of the type to which the pointer points.

```
{ tmp = *ptr; *ptr op= val; return tmp; }
{ tmp = *ptr; *ptr = ~(*ptr & val); return tmp; } // nand
```

The same constraints on arguments apply as for the corresponding `__atomic_op_fetch` built-in functions. All memory orders are valid.

`bool __atomic_test_and_set (void *ptr, int memorder)` [Built-in Function]

This built-in function performs an atomic test-and-set operation on the byte at `*ptr`. The byte is set to some implementation defined nonzero “set” value and the return value is `true` if and only if the previous contents were “set”. It should be only used for operands of type `bool` or `char`. For other types only part of the value may be set.

All memory orders are valid.

`void __atomic_clear (bool *ptr, int memorder)` [Built-in Function]

This built-in function performs an atomic clear operation on `*ptr`. After the operation, `*ptr` contains 0. It should be only used for operands of type `bool` or `char` and in conjunction with `__atomic_test_and_set`. For other types it may only clear partially. If the type is not `bool` prefer using `__atomic_store`.

The valid memory order variants are `__ATOMIC_RELAXED`, `__ATOMIC_SEQ_CST`, and `__ATOMIC_RELEASE`.

void __atomic_thread_fence (int memorder) [Built-in Function]

This built-in function acts as a synchronization fence between threads based on the specified memory order.

All memory orders are valid.

void __atomic_signal_fence (int memorder) [Built-in Function]

This built-in function acts as a synchronization fence between a thread and signal handlers based in the same thread.

All memory orders are valid.

bool __atomic_always_lock_free (size_t size, void *ptr) [Built-in Function]

This built-in function returns **true** if objects of *size* bytes always generate lock-free atomic instructions for the target architecture. *size* must resolve to a compile-time constant and the result also resolves to a compile-time constant.

ptr is an optional pointer to the object that may be used to determine alignment. A value of 0 indicates typical alignment should be used. The compiler may also ignore this parameter.

```
if (__atomic_always_lock_free (sizeof (long long), 0))
```

bool __atomic_is_lock_free (size_t size, void *ptr) [Built-in Function]

This built-in function returns **true** if objects of *size* bytes always generate lock-free atomic instructions for the target architecture. If the built-in function is not known to be lock-free, a call is made to a runtime routine named **__atomic_is_lock_free**.

ptr is an optional pointer to the object that may be used to determine alignment. A value of 0 indicates typical alignment should be used. The compiler may also ignore this parameter.

7.9.2 Legacy __sync Built-in Functions for Atomic Memory Access

The following built-in functions are intended to be compatible with those described in the *Intel Itanium Processor-specific Application Binary Interface*, section 7.4. As such, they depart from normal GCC practice by not using the ‘**__builtin_**’ prefix and also by being overloaded so that they work on multiple types.

The definition given in the Intel documentation allows only for the use of the types **int**, **long**, **long long** or their unsigned counterparts. GCC allows any scalar type that is 1, 2, 4 or 8 bytes in size other than the C type **_Bool** or the C++ type **bool**. Operations on pointer arguments are performed as if the operands were of the **uintptr_t** type. That is, they are not scaled by the size of the type to which the pointer points.

These functions are implemented in terms of the ‘**__atomic**’ builtins (see Section 7.9.1 [‘**__atomic** Builtins], page 820). They should not be used for new code which should use the ‘**__atomic**’ builtins instead.

Not all operations are supported by all target processors. If a particular operation cannot be implemented on the target processor, a call to an external function is generated. The external function carries the same name as the built-in version, with an additional suffix ‘**_n**’ where *n* is the size of the data type.

In most cases, these built-in functions are considered a *full barrier*. That is, no memory operand is moved across the operation, either forward or backward. Further, instructions are issued as necessary to prevent the processor from speculating loads across the operation and from queuing stores after the operation.

All of the routines are described in the Intel documentation to take “an optional list of variables protected by the memory barrier”. It’s not clear what is meant by that; it could mean that *only* the listed variables are protected, or it could mean a list of additional variables to be protected. The list is ignored by GCC which treats it as empty. GCC interprets an empty list as meaning that all globally accessible variables should be protected.

```

type __sync_fetch_and_add (type *ptr, type value,      [Built-in Function]
    ...)
type __sync_fetch_and_sub (type *ptr, type value,      [Built-in Function]
    ...)
type __sync_fetch_and_or (type *ptr, type value,       [Built-in Function]
    ...)
type __sync_fetch_and_and (type *ptr, type value,      [Built-in Function]
    ...)
type __sync_fetch_and_xor (type *ptr, type value,      [Built-in Function]
    ...)
type __sync_fetch_and_nand (type *ptr, type value,     [Built-in Function]
    ...)

```

These built-in functions perform the operation suggested by the name, and returns the value that had previously been in memory. That is, operations on integer operands have the following semantics. Operations on pointer arguments are performed as if the operands were of the `uintptr_t` type. That is, they are not scaled by the size of the type to which the pointer points.

```

{ tmp = *ptr; *ptr op= value; return tmp; }
{ tmp = *ptr; *ptr = ~(tmp & value); return tmp; } // nand

```

The object pointed to by the first argument must be of integer or pointer type. It must not be a boolean type.

Note: GCC 4.4 and later implement `__sync_fetch_and_nand` as `*ptr = ~(tmp & value)` instead of `*ptr = ~tmp & value`.

```

type __sync_add_and_fetch (type *ptr, type value,     [Built-in Function]
    ...)
type __sync_sub_and_fetch (type *ptr, type value,     [Built-in Function]
    ...)
type __sync_or_and_fetch (type *ptr, type value,      [Built-in Function]
    ...)
type __sync_and_and_fetch (type *ptr, type value,     [Built-in Function]
    ...)
type __sync_xor_and_fetch (type *ptr, type value,     [Built-in Function]
    ...)

```

```
type __sync_nand_and_fetch (type *ptr, type value,      [Built-in Function]
    ...)
```

These built-in functions perform the operation suggested by the name, and return the new value. That is, operations on integer operands have the following semantics. Operations on pointer operands are performed as if the operand's type were `uintptr_t`.

```
{ *ptr op= value; return *ptr; }
{ *ptr = ~(*ptr & value); return *ptr; }    // nand
```

The same constraints on arguments apply as for the corresponding `__sync_op_and_fetch` built-in functions.

Note: GCC 4.4 and later implement `__sync_nand_and_fetch` as `*ptr = ~(*ptr & value)` instead of `*ptr = ~*ptr & value`.

```
bool __sync_bool_compare_and_swap (type *ptr, type      [Built-in Function]
    oldval, type newval, ...)
```

```
type __sync_val_compare_and_swap (type *ptr, type      [Built-in Function]
    oldval, type newval, ...)
```

These built-in functions perform an atomic compare and swap. That is, if the current value of `*ptr` is `oldval`, then write `newval` into `*ptr`.

The “bool” version returns `true` if the comparison is successful and `newval` is written. The “val” version returns the contents of `*ptr` before the operation.

```
void __sync_synchronize (...)                        [Built-in Function]
```

This built-in function issues a full memory barrier.

```
type __sync_lock_test_and_set (type *ptr, type      [Built-in Function]
    value, ...)
```

This built-in function, as described by Intel, is not a traditional test-and-set operation, but rather an atomic exchange operation. It writes `value` into `*ptr`, and returns the previous contents of `*ptr`.

Many targets have only minimal support for such locks, and do not support a full exchange operation. In this case, a target may support reduced functionality here by which the *only* valid value to store is the immediate constant 1. The exact value actually stored in `*ptr` is implementation defined.

This built-in function is not a full barrier, but rather an *acquire barrier*. This means that references after the operation cannot move to (or be speculated to) before the operation, but previous memory stores may not be globally visible yet, and previous memory loads may not yet be satisfied.

```
void __sync_lock_release (type *ptr, ...)           [Built-in Function]
```

This built-in function releases the lock acquired by `__sync_lock_test_and_set`. Normally this means writing the constant 0 to `*ptr`.

This built-in function is not a full barrier, but rather a *release barrier*. This means that all previous memory stores are globally visible, and all previous memory loads have been satisfied, but following memory reads are not prevented from being speculated to before the barrier.

7.10 Object Size Checking

7.10.1 Object Size Checking Built-in Functions

GCC implements a limited buffer overflow protection mechanism that can prevent some buffer overflow attacks by determining the sizes of objects into which data is about to be written and preventing the writes when the size isn't sufficient. The built-in functions described below yield the best results when used together and when optimization is enabled. For example, to detect object sizes across function boundaries or to follow pointer assignments through non-trivial control flow they rely on various optimization passes enabled with `-O2`. However, to a limited extent, they can be used without optimization as well.

`size_t __builtin_object_size (const void * ptr, int type)` [Built-in Function]

This built-in construct returns a constant number of bytes from *ptr* to the end of the object *ptr* pointer points to (if known at compile time). To determine the sizes of dynamically allocated objects the function relies on the allocation functions called to obtain the storage to be declared with the `alloc_size` attribute (see Section 6.4.1 [Common Attributes], page 595). `__builtin_object_size` never evaluates its arguments for side effects. If there are any side effects in them, it returns `(size_t) -1` for *type* 0 or 1 and `(size_t) 0` for *type* 2 or 3. If there are multiple objects *ptr* can point to and all of them are known at compile time, the returned number is the maximum of remaining byte counts in those objects if *type* & 2 is 0 and minimum if nonzero. If it is not possible to determine which objects *ptr* points to at compile time, `__builtin_object_size` should return `(size_t) -1` for *type* 0 or 1 and `(size_t) 0` for *type* 2 or 3.

type is an integer constant from 0 to 3. If the least significant bit is clear, objects are whole variables, if it is set, a closest surrounding subobject is considered the object a pointer points to. The second bit determines if maximum or minimum of remaining bytes is computed.

```
struct V { char buf1[10]; int b; char buf2[10]; } var;
char *p = &var.buf1[1];
int *q = &var.b;

/* Here the object p points to is var. */
assert (__builtin_object_size (p, 0) == sizeof (var) - 1);
/* The subobject p points to is var.buf1. */
assert (__builtin_object_size (p, 1) == sizeof (var.buf1) - 1);
/* The object q points to is var. */
assert (__builtin_object_size (q, 0)
        == (char *) (&var + 1) - (char *) &var.b);
/* The subobject q points to is var.b. */
assert (__builtin_object_size (q, 1) == sizeof (var.b));
```

`size_t __builtin_dynamic_object_size (const void * ptr, int type)` [Built-in Function]

This builtin is similar to `__builtin_object_size` in that it returns a number of bytes from *ptr* to the end of the object *ptr* pointer points to, except that the size returned may not be a constant. This results in successful evaluation of object size estimates in a wider range of use cases and can be more precise than `__builtin_object_size`,

but it incurs a performance penalty since it may add a runtime overhead on size computation. Semantics of *type* as well as return values in case it is not possible to determine which objects *ptr* points to at compile time are the same as in the case of `__builtin_object_size`.

7.10.2 Object Size Checking and Source Fortification

Hardening of function calls using the `_FORTIFY_SOURCE` macro is one of the key uses of the object size checking built-in functions. To make implementation of these features more convenient and improve optimization and diagnostics, there are built-in functions added for many common string operation functions, e.g., for `memcpy` `__builtin___memcpy_chk` built-in is provided. This built-in has an additional last argument, which is the number of bytes remaining in the object the *dest* argument points to or (`size_t`) `-1` if the size is not known.

The built-in functions are optimized into the normal string functions like `memcpy` if the last argument is (`size_t`) `-1` or if it is known at compile time that the destination object will not be overflowed. If the compiler can determine at compile time that the object will always be overflowed, it issues a warning.

The intended use can be e.g.

```
#undef memcpy
#define bos0(dest) __builtin_object_size (dest, 0)
#define memcpy(dest, src, n) \
    __builtin___memcpy_chk (dest, src, n, bos0 (dest))

char *volatile p;
char buf[10];
/* It is unknown what object p points to, so this is optimized
   into plain memcpy - no checking is possible. */
memcpy (p, "abcde", n);
/* Destination is known and length too. It is known at compile
   time there will be no overflow. */
memcpy (&buf[5], "abcde", 5);
/* Destination is known, but the length is not known at compile time.
   This will result in __memcpy_chk call that can check for overflow
   at run time. */
memcpy (&buf[5], "abcde", n);
/* Destination is known and it is known at compile time there will
   be overflow. There will be a warning and __memcpy_chk call that
   will abort the program at run time. */
memcpy (&buf[6], "abcde", 5);
```

Such built-in functions are provided for `memcpy`, `memcpy`, `memmove`, `memset`, `strcpy`, `stpcpy`, `strncpy`, `strcat` and `strncat`.

7.10.2.1 Formatted Output Function Checking

<code>int __builtin___sprintf_chk (char *s, int flag,</code>	[Built-in Function]
<code>size_t os, const char *fmt, ...)</code>	
<code>int __builtin___snprintf_chk (char *s, size_t</code>	[Built-in Function]
<code>maxlen, int flag, size_t os, const char *fmt, ...)</code>	
<code>int __builtin___vsprintf_chk (char *s, int flag,</code>	[Built-in Function]
<code>size_t os, const char *fmt, va_list ap)</code>	

```
int __builtin__vsnprintf_chk (char *s, size_t [Built-in Function]
                             maxlen, int flag, size_t os, const char *fmt, va_list ap)
```

The added *flag* argument is passed unchanged to `__sprintf_chk` etc. functions and can contain implementation specific flags on what additional security measures the checking function might take, such as handling `%n` differently.

The *os* argument is the object size *s* points to, like in the other built-in functions. There is a small difference in the behavior though, if *os* is `(size_t) -1`, the built-in functions are optimized into the non-checking functions only if *flag* is 0, otherwise the checking function is called with *os* argument set to `(size_t) -1`.

In addition to this, there are checking built-in functions `__builtin__printf_chk`, `__builtin__vprintf_chk`, `__builtin__fprintf_chk` and `__builtin__vfprintf_chk`. These have just one additional argument, *flag*, right before format string *fmt*. If the compiler is able to optimize them to `fputc` etc. functions, it does, otherwise the checking function is called and the *flag* argument passed to it.

7.11 Built-in functions for C++ allocations and deallocations

GNU C++ provides builtins that are equivalent to calling `::operator new` or `::operator delete` with the same arguments. It is an error if the selected `::operator new` or `::operator delete` overload is not a replaceable global operator. For optimization purposes, calls to pairs of these builtins can be omitted if access to the allocation is optimized out, or could be replaced with an implementation-provided buffer on the stack, or multiple allocation calls can be merged into a single allocation. In C++ such optimizations are normally allowed just for calls to such replaceable global operators from `new` and `delete` expressions.

```
void foo () {
    int *a = new int;
    delete a; // This pair of allocation/deallocation operators can be omitted
              // or replaced with int _temp; int *a = &_temp; etc.
    void *b = ::operator new (32);
    ::operator delete (b); // This one cannot.
    void *c = __builtin_operator_new (32);
    __builtin_operator_delete (c); // This one can.
}
```

These built-ins are only available in C++.

```
void * __builtin_operator_new (std::size_t size, [Built-in Function]
                             ...)
```

This is the built-in form of `operator new`. It accepts the same argument forms as a “usual allocation function”, as described in the C++ standard.

```
void __builtin_operator_delete (void * ptr, ...) [Built-in Function]
```

This is the built-in form of `operator delete`. It accepts the same argument forms as a “usual deallocation function”, as described in the C++ standard.

7.12 Other Built-in Functions Provided by GCC

This section documents miscellaneous built-in functions available in GCC.

bool `__builtin_has_attribute` (*type-or-expression*, *attribute*) [Built-in Function]

The `__builtin_has_attribute` function evaluates to an integer constant expression equal to `true` if the symbol or type referenced by the *type-or-expression* argument has been declared with the *attribute* referenced by the second argument. For an *type-or-expression* argument that does not reference a symbol, since attributes do not apply to expressions the built-in consider the type of the argument. Neither argument is evaluated. The *type-or-expression* argument is subject to the same restrictions as the argument to `typeof` (see Section 6.12.5 [Typeof], page 784). The *attribute* argument is an attribute name optionally followed by a comma-separated list of arguments enclosed in parentheses. Both forms of attribute names—with and without double leading and trailing underscores—are recognized. See Section 6.4.3 [Attribute Syntax], page 703, for details. When no attribute arguments are specified for an attribute that expects one or more arguments the function returns `true` if *type-or-expression* has been declared with the attribute regardless of the attribute argument values. Arguments provided for an attribute that expects some are validated and matched up to the provided number. The function returns `true` if all provided arguments match. For example, the first call to the function below evaluates to `true` because `x` is declared with the `aligned` attribute but the second call evaluates to `false` because `x` is declared `aligned (8)` and not `aligned (4)`.

```
__attribute__((aligned (8))) int x;
_Static_assert (__builtin_has_attribute (x, aligned), "aligned");
_Static_assert (!__builtin_has_attribute (x, aligned (4)), "aligned (4)");
```

Due to a limitation the `__builtin_has_attribute` function returns `false` for the `mode` attribute even if the type or variable referenced by the *type-or-expression* argument was declared with one. The function is also not supported with labels, and in C with enumerators.

Note that unlike the `__has_attribute` preprocessor operator which is suitable for use in `#if` preprocessing directives `__builtin_has_attribute` is an intrinsic function that is not recognized in such contexts.

type `__builtin_speculation_safe_value` (*type val*, *type failval*) [Built-in Function]

This built-in function can be used to help mitigate against unsafe speculative execution. *type* may be any integral type or any pointer type.

1. If the CPU is not speculatively executing the code, then *val* is returned.
2. If the CPU is executing speculatively then either:
 - The function may cause execution to pause until it is known that the code is no-longer being executed speculatively (in which case *val* can be returned, as above); or
 - The function may use target-dependent speculation tracking state to cause *failval* to be returned when it is known that speculative execution has incorrectly predicted a conditional branch operation.

The second argument, *failval*, is optional and defaults to zero if omitted.

GCC defines the preprocessor macro `__HAVE_BUILTIN_SPECULATION_SAFE_VALUE` for targets that have been updated to support this builtin.

The built-in function can be used where a variable appears to be used in a safe way, but the CPU, due to speculative execution may temporarily ignore the bounds checks. Consider, for example, the following function:

```
int array[500];
int f (unsigned untrusted_index)
{
    if (untrusted_index < 500)
        return array[untrusted_index];
    return 0;
}
```

If the function is called repeatedly with `untrusted_index` less than the limit of 500, then a branch predictor will learn that the block of code that returns a value stored in `array` will be executed. If the function is subsequently called with an out-of-range value it will still try to execute that block of code first until the CPU determines that the prediction was incorrect (the CPU will unwind any incorrect operations at that point). However, depending on how the result of the function is used, it might be possible to leave traces in the cache that can reveal what was stored at the out-of-bounds location. The built-in function can be used to provide some protection against leaking data in this way by changing the code to:

```
int array[500];
int f (unsigned untrusted_index)
{
    if (untrusted_index < 500)
        return array[__builtin_speculation_safe_value (untrusted_index)];
    return 0;
}
```

The built-in function will either cause execution to stall until the conditional branch has been fully resolved, or it may permit speculative execution to continue, but using 0 instead of `untrusted_value` if that exceeds the limit.

If accessing any memory location is potentially unsafe when speculative execution is incorrect, then the code can be rewritten as

```
int array[500];
int f (unsigned untrusted_index)
{
    if (untrusted_index < 500)
        return *__builtin_speculation_safe_value (&array[untrusted_index], NULL);
    return 0;
}
```

which will cause a NULL pointer to be used for the unsafe case.

int `__builtin_types_compatible_p` (*type1*, *type2*) [Built-in Function]

You can use the built-in function `__builtin_types_compatible_p` to determine whether two types are the same.

This built-in function returns 1 if the unqualified versions of the types *type1* and *type2* (which are types, not expressions) are compatible, 0 otherwise. The result of this built-in function can be used in integer constant expressions.

This built-in function ignores top level qualifiers (e.g., `const`, `volatile`). For example, `int` is equivalent to `const int`.

The type `int[]` and `int[5]` are compatible. On the other hand, `int` and `char *` are not compatible, even if the size of their types, on the particular architecture are the

same. Also, the amount of pointer indirection is taken into account when determining similarity. Consequently, `short *` is not similar to `short **`. Furthermore, two types that are typedefed are considered compatible if their underlying types are compatible. An `enum` type is not considered to be compatible with another `enum` type even if both are compatible with the same integer type; this is what the C standard specifies. For example, `enum {foo, bar}` is not similar to `enum {hot, dog}`.

You typically use this function in code whose execution varies depending on the arguments' types. For example:

```
#define foo(x) \
({ \
    typeof (x) tmp = (x); \
    if (__builtin_types_compatible_p (typeof (x), long double)) \
        tmp = foo_long_double (tmp); \
    else if (__builtin_types_compatible_p (typeof (x), double)) \
        tmp = foo_double (tmp); \
    else if (__builtin_types_compatible_p (typeof (x), float)) \
        tmp = foo_float (tmp); \
    else \
        abort (); \
    tmp; \
})
```

Note: This construct is only available for C.

type `__builtin_choose_expr (const_exp, exp1, exp2)` [Built-in Function]

You can use the built-in function `__builtin_choose_expr` to evaluate code depending on the value of a constant expression. This built-in function returns `exp1` if `const_exp`, which is an integer constant expression, is nonzero. Otherwise it returns `exp2`.

Like the `'? :'` operator, this built-in function does not evaluate the expression that is not chosen. For example, if `const_exp` evaluates to `true`, `exp2` is not evaluated even if it has side effects. On the other hand, `__builtin_choose_expr` differs from `'? :'` in that the first operand must be a compile-time constant, and the other operands are not subject to the `'? :'` type constraints and promotions.

This built-in function can return an lvalue if the chosen argument is an lvalue.

If `exp1` is returned, the return type is the same as `exp1`'s type. Similarly, if `exp2` is returned, its return type is the same as `exp2`.

Example:

```
#define foo(x) \
__builtin_choose_expr ( \
    __builtin_types_compatible_p (typeof (x), double), \
    foo_double (x), \
    __builtin_choose_expr ( \
        __builtin_types_compatible_p (typeof (x), float), \
        foo_float (x), \
        /* The void expression results in a compile-time error \
           when assigning the result to something. */ \
        (void)0))
```

Note: This construct is only available for C. Furthermore, the unused expression (`exp1` or `exp2` depending on the value of `const_exp`) may still generate syntax errors. This may change in future revisions.

type `__builtin_tgmath (functions, arguments)` [Built-in Function]

The built-in function `__builtin_tgmath`, available only for C and Objective-C, calls a function determined according to the rules of `<tgmath.h>` macros. It is intended to be used in implementations of that header, so that expansions of macros from that header only expand each of their arguments once, to avoid problems when calls to such macros are nested inside the arguments of other calls to such macros; in addition, it results in better diagnostics for invalid calls to `<tgmath.h>` macros than implementations using other GNU C language features. For example, the `pow` type-generic macro might be defined as:

```
#define pow(a, b) __builtin_tgmath (powf, pow, powl, \
                                   cpowf, cpow, cpowl, a, b)
```

The arguments to `__builtin_tgmath` are at least two pointers to functions, followed by the arguments to the type-generic macro (which will be passed as arguments to the selected function). All the pointers to functions must be pointers to prototyped functions, none of which may have variable arguments, and all of which must have the same number of parameters; the number of parameters of the first function determines how many arguments to `__builtin_tgmath` are interpreted as function pointers, and how many as the arguments to the called function.

The types of the specified functions must all be different, but related to each other in the same way as a set of functions that may be selected between by a macro in `<tgmath.h>`. This means that the functions are parameterized by a floating-point type t , different for each such function. The function return types may all be the same type, or they may be t for each function, or they may be the real type corresponding to t for each function (if some of the types t are complex). Likewise, for each parameter position, the type of the parameter in that position may always be the same type, or may be t for each function (this case must apply for at least one parameter position), or may be the real type corresponding to t for each function.

The standard rules for `<tgmath.h>` macros are used to find a common type u from the types of the arguments for parameters whose types vary between the functions; complex integer types (a GNU extension) are treated like the complex type corresponding to the real floating type that would be chosen for the corresponding real integer type. If the function return types vary, or are all the same integer type, the function called is the one for which t is u , and it is an error if there is no such function. If the function return types are all the same floating-point type, the type-generic macro is taken to be one of those from TS 18661 that rounds the result to a narrower type; if there is a function for which t is u , it is called, and otherwise the first function, if any, for which t has at least the range and precision of u is called, and it is an error if there is no such function.

int `__builtin_constant_p (exp)` [Built-in Function]

You can use the built-in function `__builtin_constant_p` to determine if the expression `exp` is known to be constant at compile time and hence that GCC can perform constant-folding on expressions involving that value. The argument of the function is the expression to test. The expression is not evaluated, side-effects are discarded. The function returns the integer 1 if the argument is known to be a compile-time constant and 0 if it is not known to be a compile-time constant. Any expression that has side-effects makes the function return 0. A return of 0 does not indicate that

the expression is *not* a constant, but merely that GCC cannot prove it is a constant within the constraints of the active set of optimization options.

You typically use this function in an embedded application where memory is a critical resource. If you have some complex calculation, you may want it to be folded if it involves constants, but need to call a function if it does not. For example:

```
#define Scale_Value(X)      \
    (__builtin_constant_p (X) \
     ? ((X) * SCALE + OFFSET) : Scale (X))
```

You may use this built-in function in either a macro or an inline function. However, if you use it in an inlined function and pass an argument of the function as the argument to the built-in, GCC never returns 1 when you call the inline function with a string constant or compound literal (see Section 6.2.10 [Compound Literals], page 587) and does not return 1 when you pass a constant numeric value to the inline function unless you specify the `-O` option.

You may also use `__builtin_constant_p` in initializers for static data. For instance, you can write

```
static const int table[] = {
    __builtin_constant_p (EXPRESSION) ? (EXPRESSION) : -1,
    /* ... */
};
```

This is an acceptable initializer even if *EXPRESSION* is not a constant expression, including the case where `__builtin_constant_p` returns 1 because *EXPRESSION* can be folded to a constant but *EXPRESSION* contains operands that are not otherwise permitted in a static initializer (for example, `0 && foo ()`). GCC must be more conservative about evaluating the built-in in this case, because it has no opportunity to perform optimization.

bool `__builtin_is_constant_evaluated` (void) [Built-in Function]

The `__builtin_is_constant_evaluated` function is available only in C++. The built-in is intended to be used by implementations of the `std::is_constant_evaluated` C++ function. Programs should make use of the latter function rather than invoking the built-in directly.

The main use case of the built-in is to determine whether a `constexpr` function is being called in a `constexpr` context. A call to the function evaluates to a core constant expression with the value `true` if and only if it occurs within the evaluation of an expression or conversion that is manifestly constant-evaluated as defined in the C++ standard. Manifestly constant-evaluated contexts include constant-expressions, the conditions of `constexpr` if statements, constraint-expressions, and initializers of variables usable in constant expressions. For more details refer to the latest revision of the C++ standard.

type `__builtin_counted_by_ref` (ptr) [Built-in Function]

The built-in function `__builtin_counted_by_ref` checks whether the array object pointed by the pointer *ptr* has another object associated with it that represents the number of elements in the array object through the `counted_by` attribute (i.e. the counted-by object). If so, returns a pointer to the corresponding counted-by object. If such counted-by object does not exist, returns a null pointer.

This built-in function is only available in C for now.

The argument *ptr* must be a pointer to an flexible array or a pointer inside a structure. The *type* of the returned value is a pointer type pointing to the corresponding type of the counted-by object or a void pointer type in case of a null pointer being returned.

For example:

```
struct foo1 {
    int counter;
    struct bar1 array[] __attribute__((counted_by (counter)));
} *p;

struct foo2 {
    struct bar2 *pointer __attribute__((counted_by (counter2)));
    int counter2;
} *p2;

struct foo3 {
    int other;
    struct bar3 array[];
} *p3;
```

the following call to the built-in

```
__builtin_counted_by_ref (p->array)
```

returns:

```
&p->counter with type int *.
```

```
__builtin_counted_by_ref (p2->pointer)
```

returns:

```
&p2->counter2 with type int *.
```

However, the following call to the built-in

```
__builtin_counted_by_ref (p3->array)
```

returns a null pointer to void.

void __builtin_clear_padding (*ptr*) [Built-in Function]

The built-in function `__builtin_clear_padding` function clears padding bits inside of the object representation of object pointed by *ptr*, which has to be a pointer. The value representation of the object is not affected. The type of the object is assumed to be the type the pointer points to. Inside of a union, the only cleared bits are bits that are padding bits for all the union members.

This built-in-function is useful if the padding bits of an object might have indeterminate values and the object representation needs to be bitwise compared to some other object, for example for atomic operations.

For C++, *ptr* argument type should be pointer to trivially-copyable type, unless the argument is address of a variable or parameter, because otherwise it isn't known if the type isn't just a base class whose padding bits are reused or laid out differently in a derived class.

type __builtin_bit_cast (*type*, *arg*) [Built-in Function]

The `__builtin_bit_cast` function is available only in C++. The built-in is intended to be used by implementations of the `std::bit_cast` C++ template function. Programs should make use of the latter function rather than invoking the built-in directly.

This built-in function allows reinterpreting the bits of the *arg* argument as if it had type *type*. *type* and the type of the *arg* argument need to be trivially copyable types with the same size. When manifestly constant-evaluated, it performs extra diagnostics required for `std::bit_cast` and returns a constant expression if *arg* is a constant expression. For more details refer to the latest revision of the C++ standard.

long __builtin_expect (long *exp*, long *c*) [Built-in Function]

You may use `__builtin_expect` to provide the compiler with branch prediction information. In general, you should prefer to use actual profile feedback for this (`-fprofile-arcs`), as programmers are notoriously bad at predicting how their programs actually perform. However, there are applications in which this data is hard to collect.

The return value is the value of *exp*, which should be an integral expression. The semantics of the built-in are that it is expected that *exp* == *c*. For example:

```
if (__builtin_expect (x, 0))
    foo ();
```

indicates that we do not expect to call `foo`, since we expect *x* to be zero. Since you are limited to integral expressions for *exp*, you should use constructions such as

```
if (__builtin_expect (ptr != NULL, 1))
    foo (*ptr);
```

when testing pointer or floating-point values.

For the purposes of branch prediction optimizations, the probability that a `__builtin_expect` expression is true is controlled by GCC's `builtin-expect-probability` parameter, which defaults to 90%.

You can also use `__builtin_expect_with_probability` to explicitly assign a probability value to individual expressions. If the built-in is used in a loop construct, the provided probability will influence the expected number of iterations made by loop optimizations.

long __builtin_expect_with_probability [Built-in Function]
(long *exp*, long *c*, double *probability*)

This function has the same semantics as `__builtin_expect`, but the caller provides the expected probability that *exp* == *c*. The last argument, *probability*, is a floating-point value in the range 0.0 to 1.0, inclusive. The *probability* argument must be a constant floating-point expression.

void __builtin_trap (void) [Built-in Function]

This function causes the program to exit abnormally. GCC implements this function by using a target-dependent mechanism (such as intentionally executing an illegal instruction) or by calling `abort`. The mechanism used may vary from release to release so you should not rely on any particular implementation.

void __builtin_unreachable (void) [Built-in Function]

If control flow reaches the point of the `__builtin_unreachable`, the program is undefined. It is useful in situations where the compiler cannot deduce the unreachability of the code.

One such case is immediately following an `asm` statement that either never terminates, or one that transfers control elsewhere and never returns. In this example, without the `__builtin_unreachable`, GCC issues a warning that control reaches the end of a non-void function. It also generates code to return after the `asm`.

```
int f (int c, int v)
{
    if (c)
    {
        return v;
    }
    else
    {
        asm("jmp error_handler");
        __builtin_unreachable ();
    }
}
```

Because the `asm` statement unconditionally transfers control out of the function, control never reaches the end of the function body. The `__builtin_unreachable` is in fact unreachable and communicates this fact to the compiler.

Another use for `__builtin_unreachable` is following a call a function that never returns but that is not declared `__attribute__((noreturn))`, as in this example:

```
void function_that_never_returns (void);

int g (int c)
{
    if (c)
    {
        return 1;
    }
    else
    {
        function_that_never_returns ();
        __builtin_unreachable ();
    }
}
```

type `__builtin_assoc_barrier (type expr)` [Built-in Function]

This built-in inhibits re-association of the floating-point expression *expr* with expressions consuming the return value of the built-in. The expression *expr* itself can be reordered, and the whole expression *expr* can be reordered with operands after the barrier. The barrier is relevant when `-fassociative-math` is active.

```
float x0 = a + b - b;
float x1 = __builtin_assoc_barrier(a + b) - b;
```

means that, with `-fassociative-math`, `x0` can be optimized to `x0 = a` but `x1` cannot.

It is also relevant when `-ffp-contract=fast` is active; it will prevent contraction between expressions.

```
float x0 = a * b + c;
float x1 = __builtin_assoc_barrier (a * b) + c;
```

means that, with `-ffp-contract=fast`, `x0` may be optimized to use a fused multiply-add instruction but `x1` cannot.

`void * __builtin_assume_aligned (const void *exp, [Built-in Function]
size_t align, ...)`

This function returns its first argument, and allows the compiler to assume that the returned pointer is at least *align* bytes aligned. This built-in can have either two or three arguments, if it has three, the third argument should have integer type, and if it is nonzero means misalignment offset. For example:

```
void *x = __builtin_assume_aligned (arg, 16);
```

means that the compiler can assume *x*, set to *arg*, is at least 16-byte aligned, while:

```
void *x = __builtin_assume_aligned (arg, 32, 8);
```

means that the compiler can assume for *x*, set to *arg*, that `(char *) x - 8` is 32-byte aligned.

`int __builtin_LINE () [Built-in Function]`

This function is the equivalent of the preprocessor `__LINE__` macro and returns a constant integer expression that evaluates to the line number of the invocation of the built-in. When used as a C++ default argument for a function *F*, it returns the line number of the call to *F*.

`const char * __builtin_FUNCTION () [Built-in Function]`

This function is the equivalent of the `__FUNCTION__` symbol and returns an address constant pointing to the name of the function from which the built-in was invoked, or the empty string if the invocation is not at function scope. When used as a C++ default argument for a function *F*, it returns the name of *F*'s caller or the empty string if the call was not made at function scope.

`const char * __builtin_FILE () [Built-in Function]`

This function is the equivalent of the preprocessor `__FILE__` macro and returns an address constant pointing to the file name containing the invocation of the built-in, or the empty string if the invocation is not at function scope. When used as a C++ default argument for a function *F*, it returns the file name of the call to *F* or the empty string if the call was not made at function scope.

For example, in the following, each call to function `foo` will print a line similar to `"file.c:123: foo: message"` with the name of the file and the line number of the `printf` call, the name of the function `foo`, followed by the word `message`.

```
const char*
function (const char *func = __builtin_FUNCTION ())
{
    return func;
}

void foo (void)
{
    printf ("%s:%i: %s: message\n", file (), line (), function ());
}
```

`void __builtin___clear_cache (void *begin, void [Built-in Function]
*end)`

This function is used to flush the processor's instruction cache for the region of memory between *begin* inclusive and *end* exclusive. Some targets require that the instruc-

tion cache be flushed, after modifying memory containing code, in order to obtain deterministic behavior.

If the target does not require instruction cache flushes, `__builtin___clear_cache` has no effect. Otherwise either instructions are emitted in-line to clear the instruction cache or a call to the `__clear_cache` function in `libgcc` is made.

void `__builtin_prefetch` (**const void** **addr*, ...) [Built-in Function]

This function is used to minimize cache-miss latency by moving data into a cache before it is accessed. You can insert calls to `__builtin_prefetch` into code for which you know addresses of data in memory that is likely to be accessed soon. If the target supports them, data prefetch instructions are generated. If the prefetch is done early enough before the access then the data will be in the cache by the time it is accessed.

The value of *addr* is the address of the memory to prefetch. There are two optional arguments, *rw* and *locality*. The value of *rw* is a compile-time constant zero, one or two; one means that the prefetch is preparing for a write to the memory address, two means that the prefetch is preparing for a shared read (expected to be read by at least one other processor before it is written if written at all) and zero, the default, means that the prefetch is preparing for a read. The value *locality* must be a compile-time constant integer between zero and three. A value of zero means that the data has no temporal locality, so it need not be left in the cache after the access. A value of three means that the data has a high degree of temporal locality and should be left in all levels of cache possible. Values of one and two mean, respectively, a low or moderate degree of temporal locality. The default is three.

```
for (i = 0; i < n; i++)
{
    a[i] = a[i] + b[i];
    __builtin_prefetch (&a[i+j], 1, 1);
    __builtin_prefetch (&b[i+j], 0, 1);
    /* ... */
}
```

Data prefetch does not generate faults if *addr* is invalid, but the address expression itself must be valid. For example, a prefetch of `p->next` does not fault if `p->next` is not a valid address, but evaluation faults if `p` is not a valid address.

If the target does not support data prefetch, the address expression is evaluated if it includes side effects but no other code is generated and GCC does not issue a warning.

int `__builtin_classify_type` (*arg*) [Built-in Function]

int `__builtin_classify_type` (*type*) [Built-in Function]

The `__builtin_classify_type` returns a small integer with a category of *arg* argument's type, like void type, integer type, enumerals type, boolean type, pointer type, reference type, offset type, real type, complex type, function type, method type, record type, union type, array type, string type, bit-precise integer type, vector type, etc. When the argument is an expression, for backwards compatibility reason the argument is promoted like arguments passed to ... in `varargs` function, so some classes are never returned in certain languages. Alternatively, the argument of the built-in function can be a typename, such as the `typeof` specifier.

```
int a[2];
```

```
__builtin_classify_type (a) == __builtin_classify_type (int[5]);
__builtin_classify_type (a) == __builtin_classify_type (void*);
__builtin_classify_type (typeof (a)) == __builtin_classify_type (int[5]);
```

The first comparison will never be true, as *a* is implicitly converted to pointer. The last two comparisons will be true as they classify pointers in the second case and arrays in the last case.

Pmode `__builtin_extend_pointer (void * x)` [Built-in Function]

On targets where the user visible pointer size is smaller than the size of an actual hardware address this function returns the extended user pointer. Targets where this is true included ILP32 mode on x86_64 or Aarch64. This function is mainly useful when writing inline assembly code.

int `__builtin_goacc_parlevel_id (int x)` [Built-in Function]

Returns the openacc gang, worker or vector id depending on whether *x* is 0, 1 or 2.

int `__builtin_goacc_parlevel_size (int x)` [Built-in Function]

Returns the openacc gang, worker or vector size depending on whether *x* is 0, 1 or 2.

7.13 Built-in Functions Specific to Particular Target Machines

On some target machines, GCC supports many built-in functions specific to those machines. Generally these generate calls to specific machine instructions, but allow the compiler to schedule those calls.

7.13.1 AArch64 Built-in Functions

These built-in functions are available for the AArch64 family of processors.

```
unsigned int __builtin_aarch64_get_fpcr ();
void __builtin_aarch64_set_fpcr (unsigned int);
unsigned int __builtin_aarch64_get_fpsr ();
void __builtin_aarch64_set_fpsr (unsigned int);

unsigned long long __builtin_aarch64_get_fpcr64 ();
void __builtin_aarch64_set_fpcr64 (unsigned long long);
unsigned long long __builtin_aarch64_get_fpsr64 ();
void __builtin_aarch64_set_fpsr64 (unsigned long long);
```

7.13.2 Alpha Built-in Functions

These built-in functions are available for the Alpha family of processors, depending on the command-line switches used.

The following built-in functions are always available. They all generate the machine instruction that is part of the name.

```
long __builtin_alpha_implver (void);
long __builtin_alpha_rpcc (void);
long __builtin_alpha_amask (long);
long __builtin_alpha_cmpbge (long, long);
long __builtin_alpha_extbl (long, long);
long __builtin_alpha_extwl (long, long);
long __builtin_alpha_extll (long, long);
long __builtin_alpha_extql (long, long);
```

```

long __builtin_alpha_extwh (long, long);
long __builtin_alpha_extlh (long, long);
long __builtin_alpha_extqh (long, long);
long __builtin_alpha_insb1 (long, long);
long __builtin_alpha_insw1 (long, long);
long __builtin_alpha_insl1 (long, long);
long __builtin_alpha_insq1 (long, long);
long __builtin_alpha_insw4 (long, long);
long __builtin_alpha_inslh (long, long);
long __builtin_alpha_insqh (long, long);
long __builtin_alpha_mskb1 (long, long);
long __builtin_alpha_mskw1 (long, long);
long __builtin_alpha_mskl1 (long, long);
long __builtin_alpha_mskq1 (long, long);
long __builtin_alpha_mskwh (long, long);
long __builtin_alpha_msklh (long, long);
long __builtin_alpha_mskqh (long, long);
long __builtin_alpha_umulh (long, long);
long __builtin_alpha_zap (long, long);
long __builtin_alpha_zapnot (long, long);

```

The following built-in functions are always with `-mmax` or `-mcpu=cpu` where *cpu* is *pca56* or later. They all generate the machine instruction that is part of the name.

```

long __builtin_alpha_pklb (long);
long __builtin_alpha_pkwb (long);
long __builtin_alpha_unpkb1 (long);
long __builtin_alpha_unpkbw (long);
long __builtin_alpha_minub8 (long, long);
long __builtin_alpha_minsb8 (long, long);
long __builtin_alpha_minuw4 (long, long);
long __builtin_alpha_minsw4 (long, long);
long __builtin_alpha_maxub8 (long, long);
long __builtin_alpha_maxsb8 (long, long);
long __builtin_alpha_maxuw4 (long, long);
long __builtin_alpha_maxsw4 (long, long);
long __builtin_alpha_perr (long, long);

```

The following built-in functions are always with `-mcix` or `-mcpu=cpu` where *cpu* is *ev67* or later. They all generate the machine instruction that is part of the name.

```

long __builtin_alpha_cttz (long);
long __builtin_alpha_ctlz (long);
long __builtin_alpha_ctpop (long);

```

The following built-in functions are available on systems that use the OSF/1 PALcode. Normally they invoke the `rduniq` and `wruniq` PAL calls, but when invoked with `-mtls-kernel`, they invoke `rdval` and `wrval`.

```

void *__builtin_thread_pointer (void);
void __builtin_set_thread_pointer (void *);

```

7.13.3 ARC Built-in Functions

The following built-in functions are provided for ARC targets. The built-ins generate the corresponding assembly instructions. In the examples given below, the generated code often requires an operand or result to be in a register. Where necessary further code will be generated to ensure this is true, but for brevity this is not described in each case.

Note: Using a built-in to generate an instruction not supported by a target may cause problems. At present the compiler is not guaranteed to detect such misuse, and as a result an internal compiler error may be generated.

int __builtin_arc_aligned (void **val*, int *alignval*) [Built-in Function]
Return 1 if *val* is known to have the byte alignment given by *alignval*, otherwise return 0. Note that this is different from

`__alignof__(*(char *)val) >= alignval`

because `__alignof__` sees only the type of the dereference, whereas `__builtin_arc_align` uses alignment information from the pointer as well as from the pointed-to type. The information available will depend on optimization level.

void __builtin_arc_brk (void) [Built-in Function]
Generates

`brk`

unsigned int __builtin_arc_core_read (unsigned int *regno*) [Built-in Function]
The operand is the number of a register to be read. Generates:

`mov dest, rregno`

where the value in *dest* will be the result returned from the built-in.

void __builtin_arc_core_write (unsigned int *regno*, unsigned int *val*) [Built-in Function]
The first operand is the number of a register to be written, the second operand is a

compile time constant to write into that register. Generates:

`mov rregno, val`

int __builtin_arc_divaw (int *a*, int *b*) [Built-in Function]
Only available if either `-mcpu=ARC700` or `-meA` is set. Generates:

`divaw dest, a, b`

where the value in *dest* will be the result returned from the built-in.

void __builtin_arc_flag (unsigned int *a*) [Built-in Function]
Generates

`flag a`

unsigned int __builtin_arc_lr (unsigned int *auxr*) [Built-in Function]
The operand, *auxv*, is the address of an auxiliary register and must be a compile time constant. Generates:

`lr dest, [auxr]`

Where the value in *dest* will be the result returned from the built-in.

void __builtin_arc_mul64 (int *a*, int *b*) [Built-in Function]
Only available with `-mmul64`. Generates:

`mul64 a, b`

`void __builtin_arc_mulu64 (unsigned int a, unsigned int b)` [Built-in Function]

Only available with `-mmul64`. Generates:

`mulu64 a, b`

`void __builtin_arc_nop (void)` [Built-in Function]

Generates:

`nop`

`int __builtin_arc_norm (int src)` [Built-in Function]

Only valid if the ‘norm’ instruction is available through the `-mnorm` option or by default with `-mcpu=ARC700`. Generates:

`norm dest, src`

Where the value in *dest* will be the result returned from the built-in.

`short int __builtin_arc_normw (short int src)` [Built-in Function]

Only valid if the ‘normw’ instruction is available through the `-mnorm` option or by default with `-mcpu=ARC700`. Generates:

`normw dest, src`

Where the value in *dest* will be the result returned from the built-in.

`void __builtin_arc_rtie (void)` [Built-in Function]

Generates:

`rtie`

`void __builtin_arc_sleep (int a)` [Built-in Function]

Generates:

`sleep a`

`void __builtin_arc_sr (unsigned int val, unsigned int auxr)` [Built-in Function]

The first argument, *val*, is a compile time constant to be written to the register, the second argument, *auxr*, is the address of an auxiliary register. Generates:

`sr val, [auxr]`

`int __builtin_arc_swap (int src)` [Built-in Function]

Only valid with `-mswap`. Generates:

`swap dest, src`

Where the value in *dest* will be the result returned from the built-in.

`void __builtin_arc_swi (void)` [Built-in Function]

Generates:

`swi`

`void __builtin_arc_sync (void)` [Built-in Function]

Only available with `-mcpu=ARC700`. Generates:

`sync`

`void __builtin_arc_trap_s (unsigned int c)` [Built-in Function]

Only available with `-mcpu=ARC700`. Generates:

`trap_s c`

`void __builtin_arc_unimp_s (void)` [Built-in Function]

Only available with `-mcpu=ARC700`. Generates:

`unimp_s`

The instructions generated by the following builtins are not considered as candidates for scheduling. They are not moved around by the compiler during scheduling, and thus can be expected to appear where they are put in the C code:

```
__builtin_arc_brk()
__builtin_arc_core_read()
__builtin_arc_core_write()
__builtin_arc_flag()
__builtin_arc_lr()
__builtin_arc_sleep()
__builtin_arc_sr()
__builtin_arc_swi()
```

The following built-in functions are available for the ARCV2 family of processors.

```
int __builtin_arc_clri ();
void __builtin_arc_kflag (unsigned);
void __builtin_arc_seti (int);
```

The following built-in functions are available for the ARCV2 family and uses `-mnorm`.

```
int __builtin_arc_ffs (int);
int __builtin_arc_fls (int);
```

7.13.4 ARC SIMD Built-in Functions

SIMD builtins provided by the compiler can be used to generate the vector instructions. This section describes the available builtins and their usage in programs. With the `-msimd` option, the compiler provides 128-bit vector types, which can be specified using the `vector_size` attribute. The header file `arc-simd.h` can be included to use the following predefined types:

```
typedef int __v4si __attribute__((vector_size(16)));
typedef short __v8hi __attribute__((vector_size(16)));
```

These types can be used to define 128-bit variables. The built-in functions listed in the following section can be used on these variables to generate the vector operations.

For all builtins, `__builtin_arc_someinsn`, the header file `arc-simd.h` also provides equivalent macros called `_someinsn` that can be used for programming ease and improved readability. The following macros for DMA control are also provided:

```
#define _setup_dma_in_channel_reg _vdiwr
#define _setup_dma_out_channel_reg _vdowr
```

The following is a complete list of all the SIMD built-ins provided for ARC, grouped by calling signature.

The following take two `__v8hi` arguments and return a `__v8hi` result:

```
__v8hi __builtin_arc_vaddaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vaddw (__v8hi, __v8hi);
__v8hi __builtin_arc_vand (__v8hi, __v8hi);
__v8hi __builtin_arc_vandaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vavb (__v8hi, __v8hi);
__v8hi __builtin_arc_vavrb (__v8hi, __v8hi);
__v8hi __builtin_arc_vbic (__v8hi, __v8hi);
__v8hi __builtin_arc_vbicaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vdifaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vdifw (__v8hi, __v8hi);
__v8hi __builtin_arc_veqw (__v8hi, __v8hi);
__v8hi __builtin_arc_vh264f (__v8hi, __v8hi);
__v8hi __builtin_arc_vh264ft (__v8hi, __v8hi);
__v8hi __builtin_arc_vh264fw (__v8hi, __v8hi);
__v8hi __builtin_arc_vlew (__v8hi, __v8hi);
__v8hi __builtin_arc_vltw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmaxaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmaxw (__v8hi, __v8hi);
__v8hi __builtin_arc_vminaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vminw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr1aw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr1w (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr2aw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr2w (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr3aw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr3w (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr4aw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr4w (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr5aw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr5w (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr6aw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr6w (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr7aw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmr7w (__v8hi, __v8hi);
__v8hi __builtin_arc_vmrw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmulaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmulfaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmulfw (__v8hi, __v8hi);
__v8hi __builtin_arc_vmulw (__v8hi, __v8hi);
__v8hi __builtin_arc_vnew (__v8hi, __v8hi);
__v8hi __builtin_arc_vor (__v8hi, __v8hi);
__v8hi __builtin_arc_vsubaw (__v8hi, __v8hi);
__v8hi __builtin_arc_vsubw (__v8hi, __v8hi);
__v8hi __builtin_arc_vsummw (__v8hi, __v8hi);
__v8hi __builtin_arc_vvc1f (__v8hi, __v8hi);
```

```
__v8hi __builtin_arc_vvc1ft (__v8hi, __v8hi);
__v8hi __builtin_arc_vxor (__v8hi, __v8hi);
__v8hi __builtin_arc_vxoraw (__v8hi, __v8hi);
```

The following take one `__v8hi` and one `int` argument and return a `__v8hi` result:

```
__v8hi __builtin_arc_vbaddw (__v8hi, int);
__v8hi __builtin_arc_vbmaxw (__v8hi, int);
__v8hi __builtin_arc_vbminw (__v8hi, int);
__v8hi __builtin_arc_vbmulaw (__v8hi, int);
__v8hi __builtin_arc_vbmulfw (__v8hi, int);
__v8hi __builtin_arc_vbmulw (__v8hi, int);
__v8hi __builtin_arc_vbrsubw (__v8hi, int);
__v8hi __builtin_arc_vbsubw (__v8hi, int);
```

The following take one `__v8hi` argument and one `int` argument which must be a 3-bit compile time constant indicating a register number I0-I7. They return a `__v8hi` result.

```
__v8hi __builtin_arc_vasrw (__v8hi, const int);
__v8hi __builtin_arc_vsr8 (__v8hi, const int);
__v8hi __builtin_arc_vsr8aw (__v8hi, const int);
```

The following take one `__v8hi` argument and one `int` argument which must be a 6-bit compile time constant. They return a `__v8hi` result.

```
__v8hi __builtin_arc_vasrpwbi (__v8hi, const int);
__v8hi __builtin_arc_vasrrpwbi (__v8hi, const int);
__v8hi __builtin_arc_vasrrwi (__v8hi, const int);
__v8hi __builtin_arc_vasrsrwi (__v8hi, const int);
__v8hi __builtin_arc_vasrwi (__v8hi, const int);
__v8hi __builtin_arc_vsr8awi (__v8hi, const int);
__v8hi __builtin_arc_vsr8i (__v8hi, const int);
```

The following take one `__v8hi` argument and one `int` argument which must be a 8-bit compile time constant. They return a `__v8hi` result.

```
__v8hi __builtin_arc_vd6tapf (__v8hi, const int);
__v8hi __builtin_arc_vmvaw (__v8hi, const int);
__v8hi __builtin_arc_vmvw (__v8hi, const int);
__v8hi __builtin_arc_vmvzw (__v8hi, const int);
```

The following take two `int` arguments, the second of which which must be a 8-bit compile time constant. They return a `__v8hi` result:

```
__v8hi __builtin_arc_vmovaw (int, const int);
__v8hi __builtin_arc_vmovw (int, const int);
__v8hi __builtin_arc_vmovzw (int, const int);
```

The following take a single `__v8hi` argument and return a `__v8hi` result:

```
__v8hi __builtin_arc_vabsaw (__v8hi);
__v8hi __builtin_arc_vabsw (__v8hi);
__v8hi __builtin_arc_vaddsuw (__v8hi);
__v8hi __builtin_arc_vexch1 (__v8hi);
__v8hi __builtin_arc_vexch2 (__v8hi);
__v8hi __builtin_arc_vexch4 (__v8hi);
```

```
__v8hi __builtin_arc_vsignw (__v8hi);
__v8hi __builtin_arc_vupbaw (__v8hi);
__v8hi __builtin_arc_vupbw (__v8hi);
__v8hi __builtin_arc_vupsbaw (__v8hi);
__v8hi __builtin_arc_vupsbw (__v8hi);
```

The following take two `int` arguments and return no result:

```
void __builtin_arc_vdirun (int, int);
void __builtin_arc_vdoron (int, int);
```

The following take two `int` arguments and return no result. The first argument must be a 3-bit compile time constant indicating one of the DR0-DR7 DMA setup channels:

```
void __builtin_arc_vdiwr (const int, int);
void __builtin_arc_vdowr (const int, int);
```

The following take an `int` argument and return no result:

```
void __builtin_arc_vendrec (int);
void __builtin_arc_vrec (int);
void __builtin_arc_vrecrun (int);
void __builtin_arc_vrun (int);
```

The following take a `__v8hi` argument and two `int` arguments and return a `__v8hi` result. The second argument must be a 3-bit compile time constants, indicating one the registers I0-I7, and the third argument must be an 8-bit compile time constant.

Note: Although the equivalent hardware instructions do not take an SIMD register as an operand, these builtins overwrite the relevant bits of the `__v8hi` register provided as the first argument with the value loaded from the [Ib, u8] location in the SDM.

```
__v8hi __builtin_arc_vld32 (__v8hi, const int, const int);
__v8hi __builtin_arc_vld32wh (__v8hi, const int, const int);
__v8hi __builtin_arc_vld32wl (__v8hi, const int, const int);
__v8hi __builtin_arc_vld64 (__v8hi, const int, const int);
```

The following take two `int` arguments and return a `__v8hi` result. The first argument must be a 3-bit compile time constants, indicating one the registers I0-I7, and the second argument must be an 8-bit compile time constant.

```
__v8hi __builtin_arc_vld128 (const int, const int);
__v8hi __builtin_arc_vld64w (const int, const int);
```

The following take a `__v8hi` argument and two `int` arguments and return no result. The second argument must be a 3-bit compile time constants, indicating one the registers I0-I7, and the third argument must be an 8-bit compile time constant.

```
void __builtin_arc_vst128 (__v8hi, const int, const int);
void __builtin_arc_vst64 (__v8hi, const int, const int);
```

The following take a `__v8hi` argument and three `int` arguments and return no result. The second argument must be a 3-bit compile-time constant, identifying the 16-bit sub-register to be stored, the third argument must be a 3-bit compile time constants, indicating one the registers I0-I7, and the fourth argument must be an 8-bit compile time constant.

```
void __builtin_arc_vst16_n (__v8hi, const int, const int, const int);
void __builtin_arc_vst32_n (__v8hi, const int, const int, const int);
```

The following built-in functions are available on systems that uses `-mmpy-option=6` or higher.

```
__v2hi __builtin_arc_dmach (__v2hi, __v2hi);
__v2hi __builtin_arc_dmachu (__v2hi, __v2hi);
__v2hi __builtin_arc_dmpyh (__v2hi, __v2hi);
__v2hi __builtin_arc_dmpyhu (__v2hi, __v2hi);
__v2hi __builtin_arc_vaddsub2h (__v2hi, __v2hi);
__v2hi __builtin_arc_vsubadd2h (__v2hi, __v2hi);
```

The following built-in functions are available on systems that uses `-mmpy-option=7` or higher.

```
__v2si __builtin_arc_vmac2h (__v2hi, __v2hi);
__v2si __builtin_arc_vmac2hu (__v2hi, __v2hi);
__v2si __builtin_arc_vmpy2h (__v2hi, __v2hi);
__v2si __builtin_arc_vmpy2hu (__v2hi, __v2hi);
```

The following built-in functions are available on systems that uses `-mmpy-option=8` or higher.

```
long long __builtin_arc_qmach (__v4hi, __v4hi);
long long __builtin_arc_qmachu (__v4hi, __v4hi);
long long __builtin_arc_qmpyh (__v4hi, __v4hi);
long long __builtin_arc_qmpyhu (__v4hi, __v4hi);
long long __builtin_arc_dmacwh (__v2si, __v2hi);
long long __builtin_arc_dmacwhu (__v2si, __v2hi);
_v2si __builtin_arc_vaddsub (__v2si, __v2si);
_v2si __builtin_arc_vsubadd (__v2si, __v2si);
_v4hi __builtin_arc_vaddsub4h (__v4hi, __v4hi);
_v4hi __builtin_arc_vsubadd4h (__v4hi, __v4hi);
```

7.13.5 Arm C Language Extensions (ACLE)

GCC implements extensions for C and C++ as described in the Arm C Language Extensions (ACLE) specification, which can be found at <https://arm-software.github.io/acle/main/>.

As a part of ACLE, GCC implements extensions for Arm Vector extensions as described in the Arm C Language Extensions Specification. The complete list of Arm Vector extension intrinsics is available at <https://arm-software.github.io/acle/main/>. The built-in intrinsics for the Arm vector extensions are available when the respective extensions are enabled.

Not all aspects of ACLE are supported. Support for each feature of the ACLE is determined with the `__ARM_FEATURE_X` macros.

See Section 3.20.5 [ARM Options], page 341, and Section 3.20.1 [AArch64 Options], page 317, for more information on the availability of extensions.

7.13.6 ARM Floating Point Status and Control Intrinsics

These built-in functions are available for the ARM family of processors with floating-point unit.

```
unsigned int __builtin_arm_get_fpscr ();
void __builtin_arm_set_fpscr (unsigned int);
```

7.13.7 ARM ARMv8-M Security Extensions

GCC implements the ARMv8-M Security Extensions as described in the ARMv8-M Security Extensions: Requirements on Development Tools Engineering Specification, which can be found at <https://developer.arm.com/documentation/ecn0359818/latest/>.

As part of the Security Extensions GCC implements two new function attributes: `cmse_nonsecure_entry` and `cmse_nonsecure_call`.

As part of the Security Extensions GCC implements the intrinsics below. `FPTR` is used here to mean any function pointer type.

```
cmse_address_info_t cmse_TT (void *);
cmse_address_info_t cmse_TT_fptr (FPTR);
cmse_address_info_t cmse_TTT (void *);
cmse_address_info_t cmse_TTT_fptr (FPTR);
cmse_address_info_t cmse_TTA (void *);
cmse_address_info_t cmse_TTA_fptr (FPTR);
cmse_address_info_t cmse_TTAT (void *);
cmse_address_info_t cmse_TTAT_fptr (FPTR);
void * cmse_check_address_range (void *, size_t, int);
typeof(p) cmse_nsfptr_create (FPTR p);
intptr_t cmse_is_nsfptr (FPTR);
int cmse_nonsecure_caller (void);
```

7.13.8 AVR Built-in Functions

For each built-in function for AVR, there is an equally named, uppercase built-in macro defined. That way users can easily query if or if not a specific built-in is implemented or not. For example, if `__builtin_avr_nop` is available the macro `__BUILTIN_AVR_NOP` is defined to 1 and undefined otherwise.

<code>void __builtin_avr_nop (void)</code>	[Built-in Function]
<code>void __builtin_avr_sei (void)</code>	[Built-in Function]
<code>void __builtin_avr_cli (void)</code>	[Built-in Function]
<code>void __builtin_avr_sleep (void)</code>	[Built-in Function]
<code>void __builtin_avr_wdr (void)</code>	[Built-in Function]
<code>uint8_t __builtin_avr_swap (uint8_t)</code>	[Built-in Function]
<code>uint16_t __builtin_avr_fmul (uint8_t, uint8_t)</code>	[Built-in Function]
<code>int16_t __builtin_avr_fmuls (int8_t, int8_t)</code>	[Built-in Function]
<code>int16_t __builtin_avr_fmulsu (int8_t, uint8_t)</code>	[Built-in Function]

These built-in functions map to the respective machine instruction, i.e. `nop`, `sei`, `cli`, `sleep`, `wdr`, `swap`, `fmul`, `fmuls` resp. `fmulsu`. The three `fmul*` built-ins are implemented as library call if no hardware multiplier is available.

<code>void __builtin_avr_delay_cycles (uint32_t ticks)</code>	[Built-in Function]
---	---------------------

Delay execution for *ticks* cycles. Note that this built-in does not take into account the effect of interrupts that might increase delay time. *ticks* must be a compile-time integer constant; delays with a variable number of cycles are not supported.

<code>uint8_t __builtin_avr_insert_bits (uint32_t map, uint8_t bits, uint8_t val)</code>	[Built-in Function]
--	---------------------

Insert bits from *bits* into *val* and return the resulting value. The nibbles of *map* determine how the insertion is performed: Let *X* be the *n*-th nibble of *map*

1. If *X* is 0xf, then the *n*-th bit of *val* is returned unaltered.

2. If X is in the range $0 \dots 7$, then the n -th result bit is set to the X -th bit of *bits*
3. If X is in the range $8 \dots 0xe$, then the n -th result bit is undefined.

One typical use case for this built-in is adjusting input and output values to non-contiguous port layouts. Some examples:

```
// same as val, bits is unused
__builtin_avr_insert_bits (0xffffffff, bits, val);

// same as bits, val is unused
__builtin_avr_insert_bits (0x76543210, bits, val);

// same as rotating bits by 4
__builtin_avr_insert_bits (0x32107654, bits, 0);

// high nibble of result is the high nibble of val
// low nibble of result is the low nibble of bits
__builtin_avr_insert_bits (0xffff3210, bits, val);

// reverse the bit order of bits
__builtin_avr_insert_bits (0x01234567, bits, 0);
```

`uint8_t __builtin_avr_mask1 (uint8_t mask, uint8_t offs)` [Built-in Function]

Rotate the 8-bit constant value *mask* by an offset of *offs*, where *mask* is in $\{ 0x01, 0xfe, 0x7f, 0x80 \}$. This built-in can be used as an alternative to 8-bit expressions like $1 \ll \textit{offs}$ when their computation consumes too much time, and *offs* is known to be in the range $0 \dots 7$.

```
__builtin_avr_mask1 (1, offs)      // same like 1 << offs
__builtin_avr_mask1 (~1, offs)     // same like ~(1 << offs)
__builtin_avr_mask1 (0x80, offs)   // same like 0x80 >> offs
__builtin_avr_mask1 (~0x80, offs)  // same like ~(0x80 >> offs)
```

The open coded C versions take at least $5 + 4 * \textit{offs}$ cycles (and 5 instructions), whereas the built-in takes 7 cycles and instructions (8 cycles and instructions in the case of *mask* = 0x7f).

`void __builtin_avr_nops (uint16_t count)` [Built-in Function]

Insert *count* NOP instructions. The number of instructions must be a compile-time integer constant.

All of the following built-in functions are only available for GNU-C

`int8_t __builtin_avr_flash_segment (const __memx void*)` [Built-in Function]

This built-in takes a byte address to the 24-bit [AVR Named Address Spaces], page 589, `__memx` and returns the number of the flash segment (the 64 KiB chunk) where the address points to. Counting starts at 0. If the address does not point to flash memory, return -1.

```

size_t __builtin_avr_strlen_flash (const __flash      [Built-in Function]
    char*)
size_t __builtin_avr_strlen_flashx (const __flashx    [Built-in Function]
    char*)
size_t __builtin_avr_strlen_memx (const __memx        [Built-in Function]
    char*)

```

These built-ins return the length of a string located in named address-space `__flash`, `__flashx` or `__memx`, respectively. They are used to support functions like `strlen_F` from AVR-LibC (<https://avrdudes.github.io/avr-libc/avr-libc-user-manual/>)'s header `avr/flash.h`.

There are many more AVR-specific built-in functions that are used to implement the ISO/IEC TR 18037 “Embedded C” fixed-point functions of section 7.18a.6. You don’t need to use these built-ins directly. Instead, use the declarations as supplied by the `stdfix.h` header with GNU-C99:

```

#include <stdfix.h>

// Re-interpret the bit representation of unsigned 16-bit
// integer uval as Q-format 0.16 value.
unsigned fract get_bits (uint_ur_t uval)
{
    return urbits (uval);
}

```

7.13.9 Blackfin Built-in Functions

Currently, there are two Blackfin-specific built-in functions. These are used for generating CSYNC and SSYNC machine insns without using inline assembly; by using these built-in functions the compiler can automatically add workarounds for hardware errata involving these instructions. These functions are named as follows:

```

void __builtin_bfin_csync (void);
void __builtin_bfin_ssync (void);

```

7.13.10 BPF Built-in Functions

The following built-in functions are available for eBPF targets.

```

unsigned long long __builtin_bpf_load_byte (unsigned      [Built-in Function]
    long long offset)

```

Load a byte from the `struct sk_buff` packet data pointed to by the register `%r6`, and return it.

```

unsigned long long __builtin_bpf_load_half (unsigned      [Built-in Function]
    long long offset)

```

Load 16 bits from the `struct sk_buff` packet data pointed to by the register `%r6`, and return it.

```

unsigned long long __builtin_bpf_load_word (unsigned      [Built-in Function]
    long long offset)

```

Load 32 bits from the `struct sk_buff` packet data pointed to by the register `%r6`, and return it.

type `__builtin_preserve_access_index (type expr)` [Built-in Function]
 BPF Compile Once-Run Everywhere (CO-RE) support. Instruct GCC to generate CO-RE relocation records for any accesses to aggregate data structures (struct, union, array types) in *expr*. This builtin is otherwise transparent; *expr* may have any type and its value is returned. This builtin has no effect if `-mco-re` is not in effect (either specified or implied).

unsigned int `__builtin_preserve_field_info (expr, unsigned int kind)` [Built-in Function]

BPF Compile Once-Run Everywhere (CO-RE) support. This builtin is used to extract information to aid in struct/union relocations. *expr* is an access to a field of a struct or union. Depending on *kind*, different information is returned to the program. A CO-RE relocation for the access in *expr* with kind *kind* is recorded if `-mco-re` is in effect.

The following values are supported for *kind*:

`FIELD_BYTE_OFFSET = 0`

The returned value is the offset, in bytes, of the field from the beginning of the containing structure. For bit-fields, this is the byte offset of the containing word.

`FIELD_BYTE_SIZE = 1`

The returned value is the size, in bytes, of the field. For bit-fields, this is the size in bytes of the containing word.

`FIELD_EXISTENCE = 2`

The returned value is 1 if the field exists, 0 otherwise. Always 1 at compile time.

`FIELD_SIGNEDNESS = 3`

The returned value is 1 if the field is signed, 0 otherwise.

`FIELD_LSHIFT_U64 = 4`

`FIELD_RSHIFT_U64 = 5`

The returned value is the number of bits of left- or right-shifting (respectively) needed in order to recover the original value of the field, after it has been loaded by a read of `FIELD_BYTE_SIZE` bytes into an unsigned 64-bit value. Primarily useful for reading bit-field values from structures that may change between kernel versions.

Note that the return value is a constant which is known at compile time. If the field has a variable offset then `FIELD_BYTE_OFFSET`, `FIELD_LSHIFT_U64`, and `FIELD_RSHIFT_U64` are not supported. Similarly, if the field has a variable size then `FIELD_BYTE_SIZE`, `FIELD_LSHIFT_U64`, and `FIELD_RSHIFT_U64` are not supported.

For example, `__builtin_preserve_field_info` can be used to reliably extract bit-field values from a structure that may change between kernel versions:

```
struct S
{
    short a;
    int x:7;
    int y:5;
```

```

};

int
read_y (struct S *arg)
{
    unsigned long long val;
    unsigned int offset
        = __builtin_preserve_field_info (arg->y, FIELD_BYTE_OFFSET);
    unsigned int size
        = __builtin_preserve_field_info (arg->y, FIELD_BYTE_SIZE);

    /* Read size bytes from arg + offset into val.  */
    bpf_probe_read (&val, size, arg + offset);

    val <= __builtin_preserve_field_info (arg->y, FIELD_LSHIFT_U64);

    if (__builtin_preserve_field_info (arg->y, FIELD_SIGNEDNESS))
        val = ((long long) val
                >> __builtin_preserve_field_info (arg->y, FIELD_RSHIFT_U64));
    else
        val >= __builtin_preserve_field_info (arg->y, FIELD_RSHIFT_U64);

    return val;
}

```

**unsigned int __builtin_preserve_enum_value (type, [Built-in Function]
enum, unsigned int kind)**

BPF Compile Once-Run Everywhere (CO-RE) support. This builtin collects enum information and creates a CO-RE relocation relative to *enum* that should be of *type*. The *kind* specifies the action performed.

The following values are supported for *kind*:

ENUM_VALUE_EXISTS = 0

The return value is either 0 or 1 depending if the enum value exists in the target.

ENUM_VALUE = 1

The return value is the enum value in the target kernel.

**unsigned int __builtin_btf_type_id (type, unsigned [Built-in Function]
int kind)**

BPF Compile Once-Run Everywhere (CO-RE) support. This builtin is used to get the BTF type ID of a specified *type*. Depending on the *kind* argument, it either returns the ID of the local BTF information, or the BTF type ID in the target kernel.

The following values are supported for *kind*:

BTF_TYPE_ID_LOCAL = 0

Return the local BTF type ID. Always succeeds.

BTF_TYPE_ID_TARGET = 1

Return the target BTF type ID. If *type* does not exist in the target, returns 0.

`unsigned int __builtin_preserve_type_info (type, [Built-in Function]
 unsigned int kind)`

BPF Compile Once-Run Everywhere (CO-RE) support. This builtin performs named type (struct/union/enum/typedef) verifications. The type of verification depends on the *kind* argument provided. This builtin always returns 0 if *type* does not exist in the target kernel.

The following values are supported for *kind*:

`BTF_TYPE_EXISTS = 0`

Checks if *type* exists in the target.

`BTF_TYPE_MATCHES = 1`

Checks if *type* matches the local definition in the target kernel.

`BTF_TYPE_SIZE = 2`

Returns the size of the *type* within the target.

7.13.11 FR-V Built-in Functions

GCC provides many FR-V-specific built-in functions. In general, these functions are intended to be compatible with those described by *FR-V Family, Softune C/C++ Compiler Manual (V6)*, *Fujitsu Semiconductor*. The two exceptions are `__MDUNPACKH` and `__MBTOHE`, the GCC forms of which pass 128-bit values by pointer rather than by value.

Most of the functions are named after specific FR-V instructions. Such functions are said to be “directly mapped” and are summarized here in tabular form.

7.13.11.1 Argument Types

The arguments to the built-in functions can be divided into three groups: register numbers, compile-time constants and run-time values. In order to make this classification clear at a glance, the arguments and return values are given the following pseudo types:

Pseudo type	Real C type	Constant?	Description
<code>uh</code>	<code>unsigned short</code>	No	an unsigned halfword
<code>uw1</code>	<code>unsigned int</code>	No	an unsigned word
<code>sw1</code>	<code>int</code>	No	a signed word
<code>uw2</code>	<code>unsigned long long</code>	No	an unsigned doubleword
<code>sw2</code>	<code>long long</code>	No	a signed doubleword
<code>const</code>	<code>int</code>	Yes	an integer constant
<code>acc</code>	<code>int</code>	Yes	an ACC register number
<code>iacc</code>	<code>int</code>	Yes	an IACC register number

These pseudo types are not defined by GCC, they are simply a notational convenience used in this manual.

Arguments of type `uh`, `uw1`, `sw1`, `uw2` and `sw2` are evaluated at run time. They correspond to register operands in the underlying FR-V instructions.

`const` arguments represent immediate operands in the underlying FR-V instructions. They must be compile-time constants.

`acc` arguments are evaluated at compile time and specify the number of an accumulator register. For example, an `acc` argument of 2 selects the ACC2 register.

`iacc` arguments are similar to `acc` arguments but specify the number of an IACC register. See see Section 7.13.11.5 [Other Built-in Functions], page 858, for more details.

7.13.11.2 Directly-Mapped Integer Functions

The functions listed below map directly to FR-V I-type instructions.

Function prototype	Example usage	Assembly output
<code>sw1 __ADDSS (sw1, sw1)</code>	<code>c = __ADDSS (a, b)</code>	<code>ADDSS a,b,c</code>
<code>sw1 __SCAN (sw1, sw1)</code>	<code>c = __SCAN (a, b)</code>	<code>SCAN a,b,c</code>
<code>sw1 __SCUTSS (sw1)</code>	<code>b = __SCUTSS (a)</code>	<code>SCUTSS a,b</code>
<code>sw1 __SLASS (sw1, sw1)</code>	<code>c = __SLASS (a, b)</code>	<code>SLASS a,b,c</code>
<code>void __SMASS (sw1, sw1)</code>	<code>__SMASS (a, b)</code>	<code>SMASS a,b</code>
<code>void __SMSSS (sw1, sw1)</code>	<code>__SMSSS (a, b)</code>	<code>SMSSS a,b</code>
<code>void __SMU (sw1, sw1)</code>	<code>__SMU (a, b)</code>	<code>SMU a,b</code>
<code>sw2 __SMUL (sw1, sw1)</code>	<code>c = __SMUL (a, b)</code>	<code>SMUL a,b,c</code>
<code>sw1 __SUBSS (sw1, sw1)</code>	<code>c = __SUBSS (a, b)</code>	<code>SUBSS a,b,c</code>
<code>uw2 __UMUL (uw1, uw1)</code>	<code>c = __UMUL (a, b)</code>	<code>UMUL a,b,c</code>

7.13.11.3 Directly-Mapped Media Functions

The functions listed below map directly to FR-V M-type instructions.

Function prototype	Example usage	Assembly output
<code>uw1 __MABSHS (sw1)</code>	<code>b = __MABSHS (a)</code>	<code>MABSHS a,b</code>
<code>void __MADDACCS (acc, acc)</code>	<code>__MADDACCS (b, a)</code>	<code>MADDACCS a,b</code>
<code>sw1 __MADDHSS (sw1, sw1)</code>	<code>c = __MADDHSS (a, b)</code>	<code>MADDHSS a,b,c</code>
<code>uw1 __MADDHUS (uw1, uw1)</code>	<code>c = __MADDHUS (a, b)</code>	<code>MADDHUS a,b,c</code>
<code>uw1 __MAND (uw1, uw1)</code>	<code>c = __MAND (a, b)</code>	<code>MAND a,b,c</code>
<code>void __MASACCS (acc, acc)</code>	<code>__MASACCS (b, a)</code>	<code>MASACCS a,b</code>
<code>uw1 __MAVEH (uw1, uw1)</code>	<code>c = __MAVEH (a, b)</code>	<code>MAVEH a,b,c</code>
<code>uw2 __MBTOH (uw1)</code>	<code>b = __MBTOH (a)</code>	<code>MBTOH a,b</code>
<code>void __MBTOHE (uw1 *, uw1)</code>	<code>__MBTOHE (&b, a)</code>	<code>MBTOHE a,b</code>
<code>void __MCLRACC (acc)</code>	<code>__MCLRACC (a)</code>	<code>MCLRACC a</code>
<code>void __MCLRACCA (void)</code>	<code>__MCLRACCA ()</code>	<code>MCLRACCA</code>
<code>uw1 __Mcop1 (uw1, uw1)</code>	<code>c = __Mcop1 (a, b)</code>	<code>Mcop1 a,b,c</code>
<code>uw1 __Mcop2 (uw1, uw1)</code>	<code>c = __Mcop2 (a, b)</code>	<code>Mcop2 a,b,c</code>
<code>uw1 __MCPLHI (uw2, const)</code>	<code>c = __MCPLHI (a, b)</code>	<code>MCPLHI a,#b,c</code>
<code>uw1 __MCPLI (uw2, const)</code>	<code>c = __MCPLI (a, b)</code>	<code>MCPLI a,#b,c</code>
<code>void __MCPXIS (acc, sw1, sw1)</code>	<code>__MCPXIS (c, a, b)</code>	<code>MCPXIS a,b,c</code>
<code>void __MCPXIU (acc, uw1, uw1)</code>	<code>__MCPXIU (c, a, b)</code>	<code>MCPXIU a,b,c</code>
<code>void __MCPXRS (acc, sw1, sw1)</code>	<code>__MCPXRS (c, a, b)</code>	<code>MCPXRS a,b,c</code>
<code>void __MCPXRU (acc, uw1, uw1)</code>	<code>__MCPXRU (c, a, b)</code>	<code>MCPXRU a,b,c</code>
<code>uw1 __MCUT (acc, uw1)</code>	<code>c = __MCUT (a, b)</code>	<code>MCUT a,b,c</code>
<code>uw1 __MCUTSS (acc, sw1)</code>	<code>c = __MCUTSS (a, b)</code>	<code>MCUTSS a,b,c</code>
<code>void __MDADDACCS (acc, acc)</code>	<code>__MDADDACCS (b, a)</code>	<code>MDADDACCS a,b</code>
<code>void __MDASACCS (acc, acc)</code>	<code>__MDASACCS (b, a)</code>	<code>MDASACCS a,b</code>
<code>uw2 __MDCUTSSI (acc, const)</code>	<code>c = __MDCUTSSI (a, b)</code>	<code>MDCUTSSI a,#b,c</code>
<code>uw2 __MDPACKH (uw2, uw2)</code>	<code>c = __MDPACKH (a, b)</code>	<code>MDPACKH a,b,c</code>
<code>uw2 __MDROTLI (uw2, const)</code>	<code>c = __MDROTLI (a, b)</code>	<code>MDROTLI a,#b,c</code>

void __MDSUBACCS (acc, acc)	__MDSUBACCS (b, a)	MDSUBACCS a,b
void __MDUNPACKH (uw1 *, uw2)	__MDUNPACKH (&b, a)	MDUNPACKH a,b
uw2 __MEXPDHD (uw1, const)	c = __MEXPDHD (a, b)	MEXPDHD a,#b,c
uw1 __MEXPDHW (uw1, const)	c = __MEXPDHW (a, b)	MEXPDHW a,#b,c
uw1 __MHDSETH (uw1, const)	c = __MHDSETH (a, b)	MHDSETH a,#b,c
sw1 __MHDSETS (const)	b = __MHDSETS (a)	MHDSETS #a,b
uw1 __MHSETHIH (uw1, const)	b = __MHSETHIH (b, a)	MHSETHIH #a,b
sw1 __MHSETHIS (sw1, const)	b = __MHSETHIS (b, a)	MHSETHIS #a,b
uw1 __MHSETLOH (uw1, const)	b = __MHSETLOH (b, a)	MHSETLOH #a,b
sw1 __MHSETLOS (sw1, const)	b = __MHSETLOS (b, a)	MHSETLOS #a,b
uw1 __MHTOB (uw2)	b = __MHTOB (a)	MHTOB a,b
void __MMACHS (acc, sw1, sw1)	__MMACHS (c, a, b)	MMACHS a,b,c
void __MMACHU (acc, uw1, uw1)	__MMACHU (c, a, b)	MMACHU a,b,c
void __MMRDHS (acc, sw1, sw1)	__MMRDHS (c, a, b)	MMRDHS a,b,c
void __MMRDHU (acc, uw1, uw1)	__MMRDHU (c, a, b)	MMRDHU a,b,c
void __MMULHS (acc, sw1, sw1)	__MMULHS (c, a, b)	MMULHS a,b,c
void __MMULHU (acc, uw1, uw1)	__MMULHU (c, a, b)	MMULHU a,b,c
void __MMULXHS (acc, sw1, sw1)	__MMULXHS (c, a, b)	MMULXHS a,b,c
void __MMULXHU (acc, uw1, uw1)	__MMULXHU (c, a, b)	MMULXHU a,b,c
uw1 __MNOT (uw1)	b = __MNOT (a)	MNOT a,b
uw1 __MOR (uw1, uw1)	c = __MOR (a, b)	MOR a,b,c
uw1 __MPACKH (uh, uh)	c = __MPACKH (a, b)	MPACKH a,b,c
sw2 __MQADHDSS (sw2, sw2)	c = __MQADHDSS (a, b)	MQADHDSS a,b,c
uw2 __MQADHDUS (uw2, uw2)	c = __MQADHDUS (a, b)	MQADHDUS a,b,c
void __MQCPXIS (acc, sw2, sw2)	__MQCPXIS (c, a, b)	MQCPXIS a,b,c
void __MQCPXIU (acc, uw2, uw2)	__MQCPXIU (c, a, b)	MQCPXIU a,b,c
void __MQCPXRS (acc, sw2, sw2)	__MQCPXRS (c, a, b)	MQCPXRS a,b,c
void __MQCPXRU (acc, uw2, uw2)	__MQCPXRU (c, a, b)	MQCPXRU a,b,c
sw2 __MQLCLRHS (sw2, sw2)	c = __MQLCLRHS (a, b)	MQLCLRHS a,b,c
sw2 __MQLMTHS (sw2, sw2)	c = __MQLMTHS (a, b)	MQLMTHS a,b,c
void __MQMACHS (acc, sw2, sw2)	__MQMACHS (c, a, b)	MQMACHS a,b,c
void __MQMACHU (acc, uw2, uw2)	__MQMACHU (c, a, b)	MQMACHU a,b,c
void __MQMACXHS (acc, sw2, sw2)	__MQMACXHS (c, a, b)	MQMACXHS a,b,c
void __MQMULHS (acc, sw2, sw2)	__MQMULHS (c, a, b)	MQMULHS a,b,c
void __MQMULHU (acc, uw2, uw2)	__MQMULHU (c, a, b)	MQMULHU a,b,c
void __MQMULXHS (acc, sw2, sw2)	__MQMULXHS (c, a, b)	MQMULXHS a,b,c
void __MQMULXHU (acc, uw2, uw2)	__MQMULXHU (c, a, b)	MQMULXHU a,b,c
sw2 __MQSATHS (sw2, sw2)	c = __MQSATHS (a, b)	MQSATHS a,b,c
uw2 __MQSLLHI (uw2, int)	c = __MQSLLHI (a, b)	MQSLLHI a,b,c
sw2 __MQSRAHI (sw2, int)	c = __MQSRAHI (a, b)	MQSRAHI a,b,c
sw2 __MQSUBHSS (sw2, sw2)	c = __MQSUBHSS (a, b)	MQSUBHSS a,b,c
uw2 __MQSUBHUS (uw2, uw2)	c = __MQSUBHUS (a, b)	MQSUBHUS a,b,c
void __MQXMACHS (acc, sw2, sw2)	__MQXMACHS (c, a, b)	MQXMACHS a,b,c
void __MQXMACXHS (acc, sw2, sw2)	__MQXMACXHS (c, a, b)	MQXMACXHS a,b,c
uw1 __MRDACC (acc)	b = __MRDACC (a)	MRDACC a,b
uw1 __MRDACCG (acc)	b = __MRDACCG (a)	MRDACCG a,b
uw1 __MROTLI (uw1, const)	c = __MROTLI (a, b)	MROTLI a,#b,c

<code>uw1 __MROTRI (uw1, const)</code>	<code>c = __MROTRI (a, b)</code>	<code>MROTRI a,#b,c</code>
<code>sw1 __MSATHS (sw1, sw1)</code>	<code>c = __MSATHS (a, b)</code>	<code>MSATHS a,b,c</code>
<code>uw1 __MSATHU (uw1, uw1)</code>	<code>c = __MSATHU (a, b)</code>	<code>MSATHU a,b,c</code>
<code>uw1 __MSLLHI (uw1, const)</code>	<code>c = __MSLLHI (a, b)</code>	<code>MSLLHI a,#b,c</code>
<code>sw1 __MSRAHI (sw1, const)</code>	<code>c = __MSRAHI (a, b)</code>	<code>MSRAHI a,#b,c</code>
<code>uw1 __MSRLHI (uw1, const)</code>	<code>c = __MSRLHI (a, b)</code>	<code>MSRLHI a,#b,c</code>
<code>void __MSUBACCS (acc, acc)</code>	<code>__MSUBACCS (b, a)</code>	<code>MSUBACCS a,b</code>
<code>sw1 __MSUBHSS (sw1, sw1)</code>	<code>c = __MSUBHSS (a, b)</code>	<code>MSUBHSS a,b,c</code>
<code>uw1 __MSUBHUS (uw1, uw1)</code>	<code>c = __MSUBHUS (a, b)</code>	<code>MSUBHUS a,b,c</code>
<code>void __MTRAP (void)</code>	<code>__MTRAP ()</code>	<code>MTRAP</code>
<code>uw2 __MUNPACKH (uw1)</code>	<code>b = __MUNPACKH (a)</code>	<code>MUNPACKH a,b</code>
<code>uw1 __MWCUT (uw2, uw1)</code>	<code>c = __MWCUT (a, b)</code>	<code>MWCUT a,b,c</code>
<code>void __MWTACC (acc, uw1)</code>	<code>__MWTACC (b, a)</code>	<code>MWTACC a,b</code>
<code>void __MWTACCG (acc, uw1)</code>	<code>__MWTACCG (b, a)</code>	<code>MWTACCG a,b</code>
<code>uw1 __MXOR (uw1, uw1)</code>	<code>c = __MXOR (a, b)</code>	<code>MXOR a,b,c</code>

7.13.11.4 Raw Read/Write Functions

This sections describes built-in functions related to read and write instructions to access memory. These functions generate `membar` instructions to flush the I/O load and stores where appropriate, as described in Fujitsu's manual described above.

```

unsigned char __builtin_read8 (void *data)
unsigned short __builtin_read16 (void *data)
unsigned long __builtin_read32 (void *data)
unsigned long long __builtin_read64 (void *data)
void __builtin_write8 (void *data, unsigned char datum)
void __builtin_write16 (void *data, unsigned short datum)
void __builtin_write32 (void *data, unsigned long datum)
void __builtin_write64 (void *data, unsigned long long datum)

```

7.13.11.5 Other Built-in Functions

This section describes built-in functions that are not named after a specific FR-V instruction.

```

sw2 __IACCreadl1 (iacc reg)
    Return the full 64-bit value of IACC0. The reg argument is reserved for future
    expansion and must be 0.

sw1 __IACCreadl (iacc reg)
    Return the value of IACC0H if reg is 0 and IACC0L if reg is 1. Other values
    of reg are rejected as invalid.

void __IACCsetl1 (iacc reg, sw2 x)
    Set the full 64-bit value of IACC0 to x. The reg argument is reserved for future
    expansion and must be 0.

void __IACCsetl (iacc reg, sw1 x)
    Set IACC0H to x if reg is 0 and IACC0L to x if reg is 1. Other values of reg
    are rejected as invalid.

```

```
void __data_prefetch0 (const void *x)
```

Use the `dcpl` instruction to load the contents of address `x` into the data cache.

```
void __data_prefetch (const void *x)
```

Use the `nldub` instruction to load the contents of address `x` into the data cache.

The instruction is issued in slot I1.

7.13.12 LoongArch Base Built-in Functions

These built-in functions are available for LoongArch.

7.13.12.1 Data Types

- `imm0_31`, a compile-time constant in range 0 to 31;
- `imm0_16383`, a compile-time constant in range 0 to 16383;
- `imm0_32767`, a compile-time constant in range 0 to 32767;
- `imm_n2048_2047`, a compile-time constant in range -2048 to 2047;

7.13.12.2 Directly-mapped Builtin Functions

The intrinsics provided are listed below:

```
unsigned int __builtin_loongarch_movfcsr2gr (imm0_31)
void __builtin_loongarch_movgr2fcsr (imm0_31, unsigned int)
void __builtin_loongarch_cacop_d (imm0_31, unsigned long int, imm_n2048_2047)
unsigned int __builtin_loongarch_cpucfg (unsigned int)
void __builtin_loongarch_asrtle_d (long int, long int)
void __builtin_loongarch_asrtgt_d (long int, long int)
long int __builtin_loongarch_lddir_d (long int, imm0_31)
void __builtin_loongarch_ldpte_d (long int, imm0_31)

int __builtin_loongarch_crc_w_b_w (char, int)
int __builtin_loongarch_crc_w_h_w (short, int)
int __builtin_loongarch_crc_w_w_w (int, int)
int __builtin_loongarch_crc_w_d_w (long int, int)
int __builtin_loongarch_crcc_w_b_w (char, int)
int __builtin_loongarch_crcc_w_h_w (short, int)
int __builtin_loongarch_crcc_w_w_w (int, int)
int __builtin_loongarch_crcc_w_d_w (long int, int)

unsigned int __builtin_loongarch_csrrd_w (imm0_16383)
unsigned int __builtin_loongarch_csrwr_w (unsigned int, imm0_16383)
unsigned int __builtin_loongarch_csrchg_w (unsigned int, unsigned int, imm0_16383)
unsigned long int __builtin_loongarch_csrrd_d (imm0_16383)
unsigned long int __builtin_loongarch_csrwr_d (unsigned long int, imm0_16383)
unsigned long int __builtin_loongarch_csrchg_d (unsigned long int, unsigned long int, imm0_16383)

unsigned char __builtin_loongarch_iocsrrd_b (unsigned int)
unsigned short __builtin_loongarch_iocsrrd_h (unsigned int)
unsigned int __builtin_loongarch_iocsrrd_w (unsigned int)
unsigned long int __builtin_loongarch_iocsrrd_d (unsigned int)
void __builtin_loongarch_iocsrwr_b (unsigned char, unsigned int)
void __builtin_loongarch_iocsrwr_h (unsigned short, unsigned int)
void __builtin_loongarch_iocsrwr_w (unsigned int, unsigned int)
void __builtin_loongarch_iocsrwr_d (unsigned long int, unsigned int)

void __builtin_loongarch_dbar (imm0_32767)
```

```
void __builtin_loongarch_ibar (imm0_32767)

void __builtin_loongarch_syscall (imm0_32767)
void __builtin_loongarch_break (imm0_32767)
```

These intrinsic functions are available by using `-mfrecipe`.

```
float __builtin_loongarch_frecipe_s (float);
double __builtin_loongarch_frecipe_d (double);
float __builtin_loongarch_frqrte_s (float);
double __builtin_loongarch_frqrte_d (double);
```

Note: Since the control register is divided into 32-bit and 64-bit, but the access instruction is not distinguished. So GCC renames the control instructions when implementing intrinsics.

Take the `csrrd` instruction as an example, built-in functions are implemented as follows:

```
__builtin_loongarch_csrrd_w // When reading the 32-bit control register use.
__builtin_loongarch_csrrd_d // When reading the 64-bit control register use.
```

For the convenience of use, the built-in functions are encapsulated, the encapsulated functions and `__drdtime_t`, `__rdtime_t` are defined in the `larchintrin.h`. So if you call the following function you need to include `larchintrin.h`.

```
typedef struct drdtime{
    unsigned long dvalue;
    unsigned long dtimeid;
} __drdtime_t;

typedef struct rdtime{
    unsigned int value;
    unsigned int timeid;
} __rdtime_t;

__drdtime_t __rdtime_d (void)
__rdtime_t __rdtime_w (void)
__rdtime_t __rdtimeh_w (void)
unsigned int __movfcsr2gr (imm0_31)
void __movgr2fcsr (imm0_31, unsigned int)
void __cacop_d (imm0_31, unsigned long, imm_n2048_2047)
unsigned int __cpucfg (unsigned int)
void __asrtle_d (long int, long int)
void __asrtgt_d (long int, long int)
long int __lddir_d (long int, imm0_31)
void __ldpte_d (long int, imm0_31)

int __crc_w_b_w (char, int)
int __crc_w_h_w (short, int)
int __crc_w_w_w (int, int)
int __crc_w_d_w (long int, int)
int __crcc_w_b_w (char, int)
int __crcc_w_h_w (short, int)
int __crcc_w_w_w (int, int)
int __crcc_w_d_w (long int, int)

unsigned int __csrrd_w (imm0_16383)
unsigned int __csrwr_w (unsigned int, imm0_16383)
unsigned int __csrxchg_w (unsigned int, unsigned int, imm0_16383)
unsigned long __csrrd_d (imm0_16383)
unsigned long __csrwr_d (unsigned long, imm0_16383)
unsigned long __csrxchg_d (unsigned long, unsigned long, imm0_16383)

unsigned char __iocsrrd_b (unsigned int)
```

```

unsigned short __iocsrrd_h (unsigned int)
unsigned int __iocsrrd_w (unsigned int)
unsigned long __iocsrrd_d (unsigned int)
void __iocsrwr_b (unsigned char, unsigned int)
void __iocsrwr_h (unsigned short, unsigned int)
void __iocsrwr_w (unsigned int, unsigned int)
void __iocsrwr_d (unsigned long, unsigned int)

void __dbar (imm0_32767)
void __ibar (imm0_32767)

void __syscall (imm0_32767)
void __break (imm0_32767)

```

7.13.12.3 Directly-mapped Division Builtin Functions

These intrinsic functions are available by including `larchintrin.h` and using `-mfrecipe`.

```

float __frecipe_s (float);
double __frecipe_d (double);
float __frsqtrte_s (float);
double __frsqtrte_d (double);

```

7.13.12.4 Other Builtin Functions

Additional built-in functions are available for LoongArch family processors to efficiently use 128-bit floating-point (`__float128`) values.

The following are the basic built-in functions supported.

```

__float128 __builtin_fabsq (__float128);
__float128 __builtin_copysignq (__float128, __float128);
__float128 __builtin_infq (void);
__float128 __builtin_huge_valq (void);
__float128 __builtin_nanq (void);
__float128 __builtin_nansq (void);

```

Returns the value that is currently set in the ‘`tp`’ register.

```

void * __builtin_thread_pointer (void)

```

7.13.13 LoongArch SX Vector Intrinsics

GCC provides intrinsics to access the LSX (Loongson SIMD Extension) instructions. The interface is made available by including `<lsxintrin.h>` and using `-mlsx`.

7.13.13.1 SX Data Types

The following vectors typedefs are included in `lsxintrin.h`:

- `__m128i`, a 128-bit vector of fixed point;
- `__m128`, a 128-bit vector of single precision floating point;
- `__m128d`, a 128-bit vector of double precision floating point.

Instructions and corresponding built-ins may have additional restrictions and/or input/output values manipulated:

- `imm0_1`, an integer literal in range 0 to 1;
- `imm0_3`, an integer literal in range 0 to 3;
- `imm0_7`, an integer literal in range 0 to 7;

- `imm0_15`, an integer literal in range 0 to 15;
- `imm0_31`, an integer literal in range 0 to 31;
- `imm0_63`, an integer literal in range 0 to 63;
- `imm0_127`, an integer literal in range 0 to 127;
- `imm0_255`, an integer literal in range 0 to 255;
- `imm_n16_15`, an integer literal in range -16 to 15;
- `imm_n128_127`, an integer literal in range -128 to 127;
- `imm_n256_255`, an integer literal in range -256 to 255;
- `imm_n512_511`, an integer literal in range -512 to 511;
- `imm_n1024_1023`, an integer literal in range -1024 to 1023;
- `imm_n2048_2047`, an integer literal in range -2048 to 2047.

7.13.13.2 Directly-mapped SX Builtin Functions

For convenience, GCC defines functions `__lsx_vrepli_{b/h/w/d}` and `__lsx_b[n]z_{v/b/h/w/d}`, which are implemented as follows:

- `__lsx_vrepli_{b/h/w/d}`: Implemented the case where the highest bit of `vldi` instruction `i13` is 1.

```
i13[12] == 1'b0
case i13[11:10] of :
    2'b00: __lsx_vrepli_b (imm_n512_511)
    2'b01: __lsx_vrepli_h (imm_n512_511)
    2'b10: __lsx_vrepli_w (imm_n512_511)
    2'b11: __lsx_vrepli_d (imm_n512_511)
```

- `__lsx_b[n]z_{v/b/h/w/d}`: Since the `vseteqz` class directive cannot be used on its own, this function is defined.

```
_lsx_bz_v => vseteqz.v + bcnez
_lsx_bnz_v => vsetnez.v + bcnez
_lsx_bz_b => vsetanyeqz.b + bcnez
_lsx_bz_h => vsetanyeqz.h + bcnez
_lsx_bz_w => vsetanyeqz.w + bcnez
_lsx_bz_d => vsetanyeqz.d + bcnez
_lsx_bnz_b => vsetallnez.b + bcnez
_lsx_bnz_h => vsetallnez.h + bcnez
_lsx_bnz_w => vsetallnez.w + bcnez
_lsx_bnz_d => vsetallnez.d + bcnez
```

eg:

```
#include <lsxintrin.h>
```

```
extern __m128i a;
```

```
void
test (void)
{
    if (__lsx_bz_v (a))
        printf ("1\n");
    else
        printf ("2\n");
}
```

Note: For directives where the intent operand is also the source operand (modifying only part of the bitfield of the intent register), the first parameter in the builtin call function is used as the intent operand.

```
eg:
#include <lsxintrin.h>

extern __m128i dst;
extern int src;

void
test (void)
{
    dst = __lsx_vinsgr2vr_b (dst, src, 3);
}
```

The intrinsics provided are listed below:

```
int __lsx_bnz_b (__m128i);
int __lsx_bnz_d (__m128i);
int __lsx_bnz_h (__m128i);
int __lsx_bnz_v (__m128i);
int __lsx_bnz_w (__m128i);
int __lsx_bz_b (__m128i);
int __lsx_bz_d (__m128i);
int __lsx_bz_h (__m128i);
int __lsx_bz_v (__m128i);
int __lsx_bz_w (__m128i);
__m128i __lsx_vabsd_b (__m128i, __m128i);
__m128i __lsx_vabsd_bu (__m128i, __m128i);
__m128i __lsx_vabsd_d (__m128i, __m128i);
__m128i __lsx_vabsd_du (__m128i, __m128i);
__m128i __lsx_vabsd_h (__m128i, __m128i);
__m128i __lsx_vabsd_hu (__m128i, __m128i);
__m128i __lsx_vabsd_w (__m128i, __m128i);
__m128i __lsx_vabsd_wu (__m128i, __m128i);
__m128i __lsx_vadda_b (__m128i, __m128i);
__m128i __lsx_vadda_d (__m128i, __m128i);
__m128i __lsx_vadda_h (__m128i, __m128i);
__m128i __lsx_vadda_w (__m128i, __m128i);
__m128i __lsx_vadd_b (__m128i, __m128i);
__m128i __lsx_vadd_d (__m128i, __m128i);
__m128i __lsx_vadd_h (__m128i, __m128i);
__m128i __lsx_vaddi_bu (__m128i, imm0_31);
__m128i __lsx_vaddi_du (__m128i, imm0_31);
__m128i __lsx_vaddi_hu (__m128i, imm0_31);
__m128i __lsx_vaddi_wu (__m128i, imm0_31);
__m128i __lsx_vadd_q (__m128i, __m128i);
__m128i __lsx_vadd_w (__m128i, __m128i);
__m128i __lsx_vaddwev_d_w (__m128i, __m128i);
__m128i __lsx_vaddwev_d_wu (__m128i, __m128i);
__m128i __lsx_vaddwev_d_wu_w (__m128i, __m128i);
__m128i __lsx_vaddwev_h_b (__m128i, __m128i);
__m128i __lsx_vaddwev_h_bu (__m128i, __m128i);
__m128i __lsx_vaddwev_h_bu_b (__m128i, __m128i);
__m128i __lsx_vaddwev_q_d (__m128i, __m128i);
__m128i __lsx_vaddwev_q_du (__m128i, __m128i);
__m128i __lsx_vaddwev_q_du_d (__m128i, __m128i);
__m128i __lsx_vaddwev_w_h (__m128i, __m128i);
__m128i __lsx_vaddwev_w_hu (__m128i, __m128i);
```

```

__m128i __lsx_vaddwev_w_hu_h (__m128i, __m128i);
__m128i __lsx_vaddwod_d_w (__m128i, __m128i);
__m128i __lsx_vaddwod_d_wu (__m128i, __m128i);
__m128i __lsx_vaddwod_d_wu_w (__m128i, __m128i);
__m128i __lsx_vaddwod_h_b (__m128i, __m128i);
__m128i __lsx_vaddwod_h_bu (__m128i, __m128i);
__m128i __lsx_vaddwod_h_bu_b (__m128i, __m128i);
__m128i __lsx_vaddwod_q_d (__m128i, __m128i);
__m128i __lsx_vaddwod_q_du (__m128i, __m128i);
__m128i __lsx_vaddwod_q_du_d (__m128i, __m128i);
__m128i __lsx_vaddwod_w_h (__m128i, __m128i);
__m128i __lsx_vaddwod_w_hu (__m128i, __m128i);
__m128i __lsx_vaddwod_w_hu_h (__m128i, __m128i);
__m128i __lsx_vandi_b (__m128i, imm0_255);
__m128i __lsx_vandn_v (__m128i, __m128i);
__m128i __lsx_vand_v (__m128i, __m128i);
__m128i __lsx_vavg_b (__m128i, __m128i);
__m128i __lsx_vavg_bu (__m128i, __m128i);
__m128i __lsx_vavg_d (__m128i, __m128i);
__m128i __lsx_vavg_du (__m128i, __m128i);
__m128i __lsx_vavg_h (__m128i, __m128i);
__m128i __lsx_vavg_hu (__m128i, __m128i);
__m128i __lsx_vavgr_b (__m128i, __m128i);
__m128i __lsx_vavgr_bu (__m128i, __m128i);
__m128i __lsx_vavgr_d (__m128i, __m128i);
__m128i __lsx_vavgr_du (__m128i, __m128i);
__m128i __lsx_vavgr_h (__m128i, __m128i);
__m128i __lsx_vavgr_hu (__m128i, __m128i);
__m128i __lsx_vavgr_w (__m128i, __m128i);
__m128i __lsx_vavgr_wu (__m128i, __m128i);
__m128i __lsx_vavg_w (__m128i, __m128i);
__m128i __lsx_vavg_wu (__m128i, __m128i);
__m128i __lsx_vbitclr_b (__m128i, __m128i);
__m128i __lsx_vbitclr_d (__m128i, __m128i);
__m128i __lsx_vbitclr_h (__m128i, __m128i);
__m128i __lsx_vbitclri_b (__m128i, imm0_7);
__m128i __lsx_vbitclri_d (__m128i, imm0_63);
__m128i __lsx_vbitclri_h (__m128i, imm0_15);
__m128i __lsx_vbitclri_w (__m128i, imm0_31);
__m128i __lsx_vbitclr_w (__m128i, __m128i);
__m128i __lsx_vbitrev_b (__m128i, __m128i);
__m128i __lsx_vbitrev_d (__m128i, __m128i);
__m128i __lsx_vbitrev_h (__m128i, __m128i);
__m128i __lsx_vbitrevi_b (__m128i, imm0_7);
__m128i __lsx_vbitrevi_d (__m128i, imm0_63);
__m128i __lsx_vbitrevi_h (__m128i, imm0_15);
__m128i __lsx_vbitrevi_w (__m128i, imm0_31);
__m128i __lsx_vbitrev_w (__m128i, __m128i);
__m128i __lsx_vbitseli_b (__m128i, __m128i, imm0_255);
__m128i __lsx_vbitsel_v (__m128i, __m128i, __m128i);
__m128i __lsx_vbitset_b (__m128i, __m128i);
__m128i __lsx_vbitset_d (__m128i, __m128i);
__m128i __lsx_vbitset_h (__m128i, __m128i);
__m128i __lsx_vbitseti_b (__m128i, imm0_7);
__m128i __lsx_vbitseti_d (__m128i, imm0_63);
__m128i __lsx_vbitseti_h (__m128i, imm0_15);
__m128i __lsx_vbitseti_w (__m128i, imm0_31);
__m128i __lsx_vbitset_w (__m128i, __m128i);

```

```

__m128i __lsx_vbsll_v (__m128i, imm0_31);
__m128i __lsx_vbsrl_v (__m128i, imm0_31);
__m128i __lsx_vclo_b (__m128i);
__m128i __lsx_vclo_d (__m128i);
__m128i __lsx_vclo_h (__m128i);
__m128i __lsx_vclo_w (__m128i);
__m128i __lsx_vclz_b (__m128i);
__m128i __lsx_vclz_d (__m128i);
__m128i __lsx_vclz_h (__m128i);
__m128i __lsx_vclz_w (__m128i);
__m128i __lsx_vdiv_b (__m128i, __m128i);
__m128i __lsx_vdiv_bu (__m128i, __m128i);
__m128i __lsx_vdiv_d (__m128i, __m128i);
__m128i __lsx_vdiv_du (__m128i, __m128i);
__m128i __lsx_vdiv_h (__m128i, __m128i);
__m128i __lsx_vdiv_hu (__m128i, __m128i);
__m128i __lsx_vdiv_w (__m128i, __m128i);
__m128i __lsx_vdiv_wu (__m128i, __m128i);
__m128i __lsx_vexth_du_wu (__m128i);
__m128i __lsx_vexth_d_w (__m128i);
__m128i __lsx_vexth_h_b (__m128i);
__m128i __lsx_vexth_hu_bu (__m128i);
__m128i __lsx_vexth_q_d (__m128i);
__m128i __lsx_vexth_qu_du (__m128i);
__m128i __lsx_vexth_w_h (__m128i);
__m128i __lsx_vexth_wu_hu (__m128i);
__m128i __lsx_vextl_q_d (__m128i);
__m128i __lsx_vextl_qu_du (__m128i);
__m128i __lsx_vextrins_b (__m128i, __m128i, imm0_255);
__m128i __lsx_vextrins_d (__m128i, __m128i, imm0_255);
__m128i __lsx_vextrins_h (__m128i, __m128i, imm0_255);
__m128i __lsx_vextrins_w (__m128i, __m128i, imm0_255);
__m128d __lsx_vfadd_d (__m128d, __m128d);
__m128 __lsx_vfadd_s (__m128, __m128);
__m128i __lsx_vfclass_d (__m128d);
__m128i __lsx_vfclass_s (__m128);
__m128i __lsx_vfcmp_caf_d (__m128d, __m128d);
__m128i __lsx_vfcmp_caf_s (__m128, __m128);
__m128i __lsx_vfcmp_ceq_d (__m128d, __m128d);
__m128i __lsx_vfcmp_ceq_s (__m128, __m128);
__m128i __lsx_vfcmp_cle_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cle_s (__m128, __m128);
__m128i __lsx_vfcmp_clt_d (__m128d, __m128d);
__m128i __lsx_vfcmp_clt_s (__m128, __m128);
__m128i __lsx_vfcmp_cne_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cne_s (__m128, __m128);
__m128i __lsx_vfcmp_cor_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cor_s (__m128, __m128);
__m128i __lsx_vfcmp_cueq_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cueq_s (__m128, __m128);
__m128i __lsx_vfcmp_cule_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cule_s (__m128, __m128);
__m128i __lsx_vfcmp_cult_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cult_s (__m128, __m128);
__m128i __lsx_vfcmp_cun_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cune_d (__m128d, __m128d);
__m128i __lsx_vfcmp_cune_s (__m128, __m128);
__m128i __lsx_vfcmp_cun_s (__m128, __m128);

```

```

__m128i __lsx_vfcmp_saf_d (__m128d, __m128d);
__m128i __lsx_vfcmp_saf_s (__m128, __m128);
__m128i __lsx_vfcmp_seq_d (__m128d, __m128d);
__m128i __lsx_vfcmp_seq_s (__m128, __m128);
__m128i __lsx_vfcmp_sle_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sle_s (__m128, __m128);
__m128i __lsx_vfcmp_slt_d (__m128d, __m128d);
__m128i __lsx_vfcmp_slt_s (__m128, __m128);
__m128i __lsx_vfcmp_sne_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sne_s (__m128, __m128);
__m128i __lsx_vfcmp_sor_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sor_s (__m128, __m128);
__m128i __lsx_vfcmp_sueq_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sueq_s (__m128, __m128);
__m128i __lsx_vfcmp_sule_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sule_s (__m128, __m128);
__m128i __lsx_vfcmp_sult_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sult_s (__m128, __m128);
__m128i __lsx_vfcmp_sun_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sune_d (__m128d, __m128d);
__m128i __lsx_vfcmp_sune_s (__m128, __m128);
__m128i __lsx_vfcmp_sun_s (__m128, __m128);
__m128d __lsx_vfcvth_d_s (__m128);
__m128i __lsx_vfcvt_h_s (__m128, __m128);
__m128 __lsx_vfcvth_s_h (__m128i);
__m128d __lsx_vfcvtl_d_s (__m128);
__m128 __lsx_vfcvtl_s_h (__m128i);
__m128 __lsx_vfcvt_s_d (__m128d, __m128d);
__m128d __lsx_vfdiv_d (__m128d, __m128d);
__m128 __lsx_vfdiv_s (__m128, __m128);
__m128d __lsx_vffint_d_l (__m128i);
__m128d __lsx_vffint_d_lu (__m128i);
__m128d __lsx_vffinth_d_w (__m128i);
__m128d __lsx_vffintl_d_w (__m128i);
__m128 __lsx_vffint_s_l (__m128i, __m128i);
__m128 __lsx_vffint_s_w (__m128i);
__m128 __lsx_vffint_s_wu (__m128i);
__m128d __lsx_vflogb_d (__m128d);
__m128 __lsx_vflogb_s (__m128);
__m128d __lsx_vfmadd_d (__m128d, __m128d, __m128d);
__m128 __lsx_vfmadd_s (__m128, __m128, __m128);
__m128d __lsx_vfmaxa_d (__m128d, __m128d);
__m128 __lsx_vfmaxa_s (__m128, __m128);
__m128d __lsx_vfmax_d (__m128d, __m128d);
__m128 __lsx_vfmax_s (__m128, __m128);
__m128d __lsx_vfmina_d (__m128d, __m128d);
__m128 __lsx_vfmina_s (__m128, __m128);
__m128d __lsx_vfmin_d (__m128d, __m128d);
__m128 __lsx_vfmin_s (__m128, __m128);
__m128d __lsx_vfmsub_d (__m128d, __m128d, __m128d);
__m128 __lsx_vfmsub_s (__m128, __m128, __m128);
__m128d __lsx_vfmul_d (__m128d, __m128d);
__m128 __lsx_vfmul_s (__m128, __m128);
__m128d __lsx_vfnmadd_d (__m128d, __m128d, __m128d);
__m128 __lsx_vfnmadd_s (__m128, __m128, __m128);
__m128d __lsx_vfnmsub_d (__m128d, __m128d, __m128d);
__m128 __lsx_vfnmsub_s (__m128, __m128, __m128);
__m128d __lsx_vfrecip_d (__m128d);

```

```

__m128 __lsx_vfrecip_s (__m128);
__m128d __lsx_vfrint_d (__m128d);
__m128d __lsx_vfrintrm_d (__m128d);
__m128 __lsx_vfrintrm_s (__m128);
__m128d __lsx_vfrintrne_d (__m128d);
__m128 __lsx_vfrintrne_s (__m128);
__m128d __lsx_vfrintrp_d (__m128d);
__m128 __lsx_vfrintrp_s (__m128);
__m128d __lsx_vfrintrz_d (__m128d);
__m128 __lsx_vfrintrz_s (__m128);
__m128 __lsx_vfrint_s (__m128);
__m128d __lsx_vfrsqrt_d (__m128d);
__m128 __lsx_vfrsqrt_s (__m128);
__m128i __lsx_vfrstp_b (__m128i, __m128i, __m128i);
__m128i __lsx_vfrstp_h (__m128i, __m128i, __m128i);
__m128i __lsx_vfrstpi_b (__m128i, __m128i, imm0_31);
__m128i __lsx_vfrstpi_h (__m128i, __m128i, imm0_31);
__m128d __lsx_vfsqrt_d (__m128d);
__m128 __lsx_vfsqrt_s (__m128);
__m128d __lsx_vfsub_d (__m128d, __m128d);
__m128 __lsx_vfsub_s (__m128, __m128);
__m128i __lsx_vftinth_l_s (__m128);
__m128i __lsx_vftint_l_d (__m128d);
__m128i __lsx_vftintl_l_s (__m128);
__m128i __lsx_vftint_lu_d (__m128d);
__m128i __lsx_vftintrmh_l_s (__m128);
__m128i __lsx_vftintrm_l_d (__m128d);
__m128i __lsx_vftintrml_l_s (__m128);
__m128i __lsx_vftintrm_w_d (__m128d, __m128d);
__m128i __lsx_vftintrm_w_s (__m128);
__m128i __lsx_vftintrneh_l_s (__m128);
__m128i __lsx_vftintrne_l_d (__m128d);
__m128i __lsx_vftintrnel_l_s (__m128);
__m128i __lsx_vftintrne_w_d (__m128d, __m128d);
__m128i __lsx_vftintrne_w_s (__m128);
__m128i __lsx_vftintrph_l_s (__m128);
__m128i __lsx_vftintrp_l_d (__m128d);
__m128i __lsx_vftintrpl_l_s (__m128);
__m128i __lsx_vftintrp_w_d (__m128d, __m128d);
__m128i __lsx_vftintrp_w_s (__m128);
__m128i __lsx_vftintrzh_l_s (__m128);
__m128i __lsx_vftintrz_l_d (__m128d);
__m128i __lsx_vftintrzl_l_s (__m128);
__m128i __lsx_vftintrz_lu_d (__m128d);
__m128i __lsx_vftintrz_w_d (__m128d, __m128d);
__m128i __lsx_vftintrz_w_s (__m128);
__m128i __lsx_vftintrz_wu_s (__m128);
__m128i __lsx_vftint_w_d (__m128d, __m128d);
__m128i __lsx_vftint_w_s (__m128);
__m128i __lsx_vftint_wu_s (__m128);
__m128i __lsx_vhaddw_du_wu (__m128i, __m128i);
__m128i __lsx_vhaddw_d_w (__m128i, __m128i);
__m128i __lsx_vhaddw_h_b (__m128i, __m128i);
__m128i __lsx_vhaddw_hu_bu (__m128i, __m128i);
__m128i __lsx_vhaddw_q_d (__m128i, __m128i);
__m128i __lsx_vhaddw_qu_du (__m128i, __m128i);
__m128i __lsx_vhaddw_w_h (__m128i, __m128i);
__m128i __lsx_vhaddw_wu_hu (__m128i, __m128i);

```

```

__m128i __lsx_vhsubw_du_wu (__m128i, __m128i);
__m128i __lsx_vhsubw_d_w (__m128i, __m128i);
__m128i __lsx_vhsubw_h_b (__m128i, __m128i);
__m128i __lsx_vhsubw_hu_bu (__m128i, __m128i);
__m128i __lsx_vhsubw_q_d (__m128i, __m128i);
__m128i __lsx_vhsubw_qu_du (__m128i, __m128i);
__m128i __lsx_vhsubw_w_h (__m128i, __m128i);
__m128i __lsx_vhsubw_wu_hu (__m128i, __m128i);
__m128i __lsx_vilvh_b (__m128i, __m128i);
__m128i __lsx_vilvh_d (__m128i, __m128i);
__m128i __lsx_vilvh_h (__m128i, __m128i);
__m128i __lsx_vilvh_w (__m128i, __m128i);
__m128i __lsx_vilvl_b (__m128i, __m128i);
__m128i __lsx_vilvl_d (__m128i, __m128i);
__m128i __lsx_vilvl_h (__m128i, __m128i);
__m128i __lsx_vilvl_w (__m128i, __m128i);
__m128i __lsx_vinsgr2vr_b (__m128i, int, imm0_15);
__m128i __lsx_vinsgr2vr_d (__m128i, long int, imm0_1);
__m128i __lsx_vinsgr2vr_h (__m128i, int, imm0_7);
__m128i __lsx_vinsgr2vr_w (__m128i, int, imm0_3);
__m128i __lsx_vld (void *, imm_n2048_2047);
__m128i __lsx_vldi (imm_n1024_1023);
__m128i __lsx_vldrepl_b (void *, imm_n2048_2047);
__m128i __lsx_vldrepl_d (void *, imm_n256_255);
__m128i __lsx_vldrepl_h (void *, imm_n1024_1023);
__m128i __lsx_vldrepl_w (void *, imm_n512_511);
__m128i __lsx_vldx (void *, long int);
__m128i __lsx_vmadd_b (__m128i, __m128i, __m128i);
__m128i __lsx_vmadd_d (__m128i, __m128i, __m128i);
__m128i __lsx_vmadd_h (__m128i, __m128i, __m128i);
__m128i __lsx_vmadd_w (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_d_w (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_d_wu (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_d_wu_w (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_h_b (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_h_bu (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_h_bu_b (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_q_d (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_q_du (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_q_du_d (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_w_h (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_w_hu (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwev_w_hu_h (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_d_w (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_d_wu (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_d_wu_w (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_h_b (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_h_bu (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_h_bu_b (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_q_d (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_q_du (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_q_du_d (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_w_h (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_w_hu (__m128i, __m128i, __m128i);
__m128i __lsx_vmaddwod_w_hu_h (__m128i, __m128i, __m128i);
__m128i __lsx_vmax_b (__m128i, __m128i);
__m128i __lsx_vmax_bu (__m128i, __m128i);
__m128i __lsx_vmax_d (__m128i, __m128i);

```

```

__m128i __lsx_vmax_du (__m128i, __m128i);
__m128i __lsx_vmax_h (__m128i, __m128i);
__m128i __lsx_vmax_hu (__m128i, __m128i);
__m128i __lsx_vmaxi_b (__m128i, imm_n16_15);
__m128i __lsx_vmaxi_bu (__m128i, imm0_31);
__m128i __lsx_vmaxi_d (__m128i, imm_n16_15);
__m128i __lsx_vmaxi_du (__m128i, imm0_31);
__m128i __lsx_vmaxi_h (__m128i, imm_n16_15);
__m128i __lsx_vmaxi_hu (__m128i, imm0_31);
__m128i __lsx_vmaxi_w (__m128i, imm_n16_15);
__m128i __lsx_vmaxi_wu (__m128i, imm0_31);
__m128i __lsx_vmax_w (__m128i, __m128i);
__m128i __lsx_vmax_wu (__m128i, __m128i);
__m128i __lsx_vmin_b (__m128i, __m128i);
__m128i __lsx_vmin_bu (__m128i, __m128i);
__m128i __lsx_vmin_d (__m128i, __m128i);
__m128i __lsx_vmin_du (__m128i, __m128i);
__m128i __lsx_vmin_h (__m128i, __m128i);
__m128i __lsx_vmin_hu (__m128i, __m128i);
__m128i __lsx_vmini_b (__m128i, imm_n16_15);
__m128i __lsx_vmini_bu (__m128i, imm0_31);
__m128i __lsx_vmini_d (__m128i, imm_n16_15);
__m128i __lsx_vmini_du (__m128i, imm0_31);
__m128i __lsx_vmini_h (__m128i, imm_n16_15);
__m128i __lsx_vmini_hu (__m128i, imm0_31);
__m128i __lsx_vmini_w (__m128i, imm_n16_15);
__m128i __lsx_vmini_wu (__m128i, imm0_31);
__m128i __lsx_vmin_w (__m128i, __m128i);
__m128i __lsx_vmin_wu (__m128i, __m128i);
__m128i __lsx_vmod_b (__m128i, __m128i);
__m128i __lsx_vmod_bu (__m128i, __m128i);
__m128i __lsx_vmod_d (__m128i, __m128i);
__m128i __lsx_vmod_du (__m128i, __m128i);
__m128i __lsx_vmod_h (__m128i, __m128i);
__m128i __lsx_vmod_hu (__m128i, __m128i);
__m128i __lsx_vmod_w (__m128i, __m128i);
__m128i __lsx_vmod_wu (__m128i, __m128i);
__m128i __lsx_vmskgez_b (__m128i);
__m128i __lsx_vmskltz_b (__m128i);
__m128i __lsx_vmskltz_d (__m128i);
__m128i __lsx_vmskltz_h (__m128i);
__m128i __lsx_vmskltz_w (__m128i);
__m128i __lsx_vmsknz_b (__m128i);
__m128i __lsx_vmsub_b (__m128i, __m128i, __m128i);
__m128i __lsx_vmsub_d (__m128i, __m128i, __m128i);
__m128i __lsx_vmsub_h (__m128i, __m128i, __m128i);
__m128i __lsx_vmsub_w (__m128i, __m128i, __m128i);
__m128i __lsx_vmuh_b (__m128i, __m128i);
__m128i __lsx_vmuh_bu (__m128i, __m128i);
__m128i __lsx_vmuh_d (__m128i, __m128i);
__m128i __lsx_vmuh_du (__m128i, __m128i);
__m128i __lsx_vmuh_h (__m128i, __m128i);
__m128i __lsx_vmuh_hu (__m128i, __m128i);
__m128i __lsx_vmuh_w (__m128i, __m128i);
__m128i __lsx_vmuh_wu (__m128i, __m128i);
__m128i __lsx_vmul_b (__m128i, __m128i);
__m128i __lsx_vmul_d (__m128i, __m128i);
__m128i __lsx_vmul_h (__m128i, __m128i);

```

```

__m128i __lsx_vmul_w (__m128i, __m128i);
__m128i __lsx_vmulwev_d_w (__m128i, __m128i);
__m128i __lsx_vmulwev_d_wu (__m128i, __m128i);
__m128i __lsx_vmulwev_d_wu_w (__m128i, __m128i);
__m128i __lsx_vmulwev_h_b (__m128i, __m128i);
__m128i __lsx_vmulwev_h_bu (__m128i, __m128i);
__m128i __lsx_vmulwev_h_bu_b (__m128i, __m128i);
__m128i __lsx_vmulwev_q_d (__m128i, __m128i);
__m128i __lsx_vmulwev_q_du (__m128i, __m128i);
__m128i __lsx_vmulwev_q_du_d (__m128i, __m128i);
__m128i __lsx_vmulwev_w_h (__m128i, __m128i);
__m128i __lsx_vmulwev_w_hu (__m128i, __m128i);
__m128i __lsx_vmulwev_w_hu_h (__m128i, __m128i);
__m128i __lsx_vmulwod_d_w (__m128i, __m128i);
__m128i __lsx_vmulwod_d_wu (__m128i, __m128i);
__m128i __lsx_vmulwod_d_wu_w (__m128i, __m128i);
__m128i __lsx_vmulwod_h_b (__m128i, __m128i);
__m128i __lsx_vmulwod_h_bu (__m128i, __m128i);
__m128i __lsx_vmulwod_h_bu_b (__m128i, __m128i);
__m128i __lsx_vmulwod_q_d (__m128i, __m128i);
__m128i __lsx_vmulwod_q_du (__m128i, __m128i);
__m128i __lsx_vmulwod_q_du_d (__m128i, __m128i);
__m128i __lsx_vmulwod_w_h (__m128i, __m128i);
__m128i __lsx_vmulwod_w_hu (__m128i, __m128i);
__m128i __lsx_vmulwod_w_hu_h (__m128i, __m128i);
__m128i __lsx_vneg_b (__m128i);
__m128i __lsx_vneg_d (__m128i);
__m128i __lsx_vneg_h (__m128i);
__m128i __lsx_vneg_w (__m128i);
__m128i __lsx_vnori_b (__m128i, imm0_255);
__m128i __lsx_vnor_v (__m128i, __m128i);
__m128i __lsx_vori_b (__m128i, imm0_255);
__m128i __lsx_vorn_v (__m128i, __m128i);
__m128i __lsx_vor_v (__m128i, __m128i);
__m128i __lsx_vpackev_b (__m128i, __m128i);
__m128i __lsx_vpackev_d (__m128i, __m128i);
__m128i __lsx_vpackev_h (__m128i, __m128i);
__m128i __lsx_vpackev_w (__m128i, __m128i);
__m128i __lsx_vpackod_b (__m128i, __m128i);
__m128i __lsx_vpackod_d (__m128i, __m128i);
__m128i __lsx_vpackod_h (__m128i, __m128i);
__m128i __lsx_vpackod_w (__m128i, __m128i);
__m128i __lsx_vpcnt_b (__m128i);
__m128i __lsx_vpcnt_d (__m128i);
__m128i __lsx_vpcnt_h (__m128i);
__m128i __lsx_vpcnt_w (__m128i);
__m128i __lsx_vpermi_w (__m128i, __m128i, imm0_255);
__m128i __lsx_vpicev_b (__m128i, __m128i);
__m128i __lsx_vpicev_d (__m128i, __m128i);
__m128i __lsx_vpicev_h (__m128i, __m128i);
__m128i __lsx_vpicev_w (__m128i, __m128i);
__m128i __lsx_vpickod_b (__m128i, __m128i);
__m128i __lsx_vpickod_d (__m128i, __m128i);
__m128i __lsx_vpickod_h (__m128i, __m128i);
__m128i __lsx_vpickod_w (__m128i, __m128i);
int __lsx_vpickve2gr_b (__m128i, imm0_15);
unsigned int __lsx_vpickve2gr_bu (__m128i, imm0_15);
long int __lsx_vpickve2gr_d (__m128i, imm0_1);

```

```

unsigned long int __lsx_vpickve2gr_du (__m128i, imm0_1);
int __lsx_vpickve2gr_h (__m128i, imm0_7);
unsigned int __lsx_vpickve2gr_hu (__m128i, imm0_7);
int __lsx_vpickve2gr_w (__m128i, imm0_3);
unsigned int __lsx_vpickve2gr_wu (__m128i, imm0_3);
__m128i __lsx_vreplgr2vr_b (int);
__m128i __lsx_vreplgr2vr_d (long int);
__m128i __lsx_vreplgr2vr_h (int);
__m128i __lsx_vreplgr2vr_w (int);
__m128i __lsx_vrepli_b (imm_n512_511);
__m128i __lsx_vrepli_d (imm_n512_511);
__m128i __lsx_vrepli_h (imm_n512_511);
__m128i __lsx_vrepli_w (imm_n512_511);
__m128i __lsx_vreplve_b (__m128i, int);
__m128i __lsx_vreplve_d (__m128i, int);
__m128i __lsx_vreplve_h (__m128i, int);
__m128i __lsx_vreplvei_b (__m128i, imm0_15);
__m128i __lsx_vreplvei_d (__m128i, imm0_1);
__m128i __lsx_vreplvei_h (__m128i, imm0_7);
__m128i __lsx_vreplvei_w (__m128i, imm0_3);
__m128i __lsx_vreplve_w (__m128i, int);
__m128i __lsx_vrotr_b (__m128i, __m128i);
__m128i __lsx_vrotr_d (__m128i, __m128i);
__m128i __lsx_vrotr_h (__m128i, __m128i);
__m128i __lsx_vrotri_b (__m128i, imm0_7);
__m128i __lsx_vrotri_d (__m128i, imm0_63);
__m128i __lsx_vrotri_h (__m128i, imm0_15);
__m128i __lsx_vrotri_w (__m128i, imm0_31);
__m128i __lsx_vrotr_w (__m128i, __m128i);
__m128i __lsx_vsadd_b (__m128i, __m128i);
__m128i __lsx_vsadd_bu (__m128i, __m128i);
__m128i __lsx_vsadd_d (__m128i, __m128i);
__m128i __lsx_vsadd_du (__m128i, __m128i);
__m128i __lsx_vsadd_h (__m128i, __m128i);
__m128i __lsx_vsadd_hu (__m128i, __m128i);
__m128i __lsx_vsadd_w (__m128i, __m128i);
__m128i __lsx_vsadd_wu (__m128i, __m128i);
__m128i __lsx_vsat_b (__m128i, imm0_7);
__m128i __lsx_vsat_bu (__m128i, imm0_7);
__m128i __lsx_vsat_d (__m128i, imm0_63);
__m128i __lsx_vsat_du (__m128i, imm0_63);
__m128i __lsx_vsat_h (__m128i, imm0_15);
__m128i __lsx_vsat_hu (__m128i, imm0_15);
__m128i __lsx_vsat_w (__m128i, imm0_31);
__m128i __lsx_vsat_wu (__m128i, imm0_31);
__m128i __lsx_vseq_b (__m128i, __m128i);
__m128i __lsx_vseq_d (__m128i, __m128i);
__m128i __lsx_vseq_h (__m128i, __m128i);
__m128i __lsx_vseqi_b (__m128i, imm_n16_15);
__m128i __lsx_vseqi_d (__m128i, imm_n16_15);
__m128i __lsx_vseqi_h (__m128i, imm_n16_15);
__m128i __lsx_vseqi_w (__m128i, imm_n16_15);
__m128i __lsx_vseq_w (__m128i, __m128i);
__m128i __lsx_vshuf4i_b (__m128i, imm0_255);
__m128i __lsx_vshuf4i_d (__m128i, __m128i, imm0_255);
__m128i __lsx_vshuf4i_h (__m128i, imm0_255);
__m128i __lsx_vshuf4i_w (__m128i, imm0_255);
__m128i __lsx_vshuf_b (__m128i, __m128i, __m128i);

```



```

__m128i __lsx_vsrai_h (__m128i, imm0_15);
__m128i __lsx_vsrai_w (__m128i, imm0_31);
__m128i __lsx_vsrar_b_h (__m128i, __m128i);
__m128i __lsx_vsrar_h_w (__m128i, __m128i);
__m128i __lsx_vsrani_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vsrani_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vsrani_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vsrani_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vsrar_w_d (__m128i, __m128i);
__m128i __lsx_vsrar_b (__m128i, __m128i);
__m128i __lsx_vsrar_d (__m128i, __m128i);
__m128i __lsx_vsrar_h (__m128i, __m128i);
__m128i __lsx_vsrari_b (__m128i, imm0_7);
__m128i __lsx_vsrari_d (__m128i, imm0_63);
__m128i __lsx_vsrari_h (__m128i, imm0_15);
__m128i __lsx_vsrari_w (__m128i, imm0_31);
__m128i __lsx_vsrarn_b_h (__m128i, __m128i);
__m128i __lsx_vsrarn_h_w (__m128i, __m128i);
__m128i __lsx_vsrarni_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vsrarni_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vsrarni_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vsrarni_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vsrarn_w_d (__m128i, __m128i);
__m128i __lsx_vsrar_w (__m128i, __m128i);
__m128i __lsx_vsra_w (__m128i, __m128i);
__m128i __lsx_vsrl_b (__m128i, __m128i);
__m128i __lsx_vsrl_d (__m128i, __m128i);
__m128i __lsx_vsrl_h (__m128i, __m128i);
__m128i __lsx_vsrli_b (__m128i, imm0_7);
__m128i __lsx_vsrli_d (__m128i, imm0_63);
__m128i __lsx_vsrli_h (__m128i, imm0_15);
__m128i __lsx_vsrli_w (__m128i, imm0_31);
__m128i __lsx_vsrln_b_h (__m128i, __m128i);
__m128i __lsx_vsrln_h_w (__m128i, __m128i);
__m128i __lsx_vsrlni_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vsrlni_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vsrlni_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vsrlni_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vsrln_w_d (__m128i, __m128i);
__m128i __lsx_vsrlr_b (__m128i, __m128i);
__m128i __lsx_vsrlr_d (__m128i, __m128i);
__m128i __lsx_vsrlr_h (__m128i, __m128i);
__m128i __lsx_vsrlri_b (__m128i, imm0_7);
__m128i __lsx_vsrlri_d (__m128i, imm0_63);
__m128i __lsx_vsrlri_h (__m128i, imm0_15);
__m128i __lsx_vsrlri_w (__m128i, imm0_31);
__m128i __lsx_vsrlrn_b_h (__m128i, __m128i);
__m128i __lsx_vsrlrn_h_w (__m128i, __m128i);
__m128i __lsx_vsrlrni_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vsrlrni_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vsrlrni_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vsrlrni_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vsrlrn_w_d (__m128i, __m128i);
__m128i __lsx_vsrlr_w (__m128i, __m128i);
__m128i __lsx_vsrl_w (__m128i, __m128i);
__m128i __lsx_vssran_b_h (__m128i, __m128i);
__m128i __lsx_vssran_bu_h (__m128i, __m128i);
__m128i __lsx_vssran_hu_w (__m128i, __m128i);

```

```

__m128i __lsx_vssran_h_w (__m128i, __m128i);
__m128i __lsx_vssrani_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrani_bu_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrani_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrani_du_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrani_hu_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrani_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrani_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssrani_wu_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssran_w_d (__m128i, __m128i);
__m128i __lsx_vssran_wu_d (__m128i, __m128i);
__m128i __lsx_vssrarn_b_h (__m128i, __m128i);
__m128i __lsx_vssrarn_bu_h (__m128i, __m128i);
__m128i __lsx_vssrarn_hu_w (__m128i, __m128i);
__m128i __lsx_vssrarn_h_w (__m128i, __m128i);
__m128i __lsx_vssrarni_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrarni_bu_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrarni_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrarni_du_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrarni_hu_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrarni_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrarni_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssrarni_wu_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssrarn_w_d (__m128i, __m128i);
__m128i __lsx_vssrarn_wu_d (__m128i, __m128i);
__m128i __lsx_vssrln_b_h (__m128i, __m128i);
__m128i __lsx_vssrln_bu_h (__m128i, __m128i);
__m128i __lsx_vssrln_hu_w (__m128i, __m128i);
__m128i __lsx_vssrln_h_w (__m128i, __m128i);
__m128i __lsx_vssrlni_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrlni_bu_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrlni_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrlni_du_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrlni_hu_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrlni_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrlni_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssrlni_wu_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssrln_w_d (__m128i, __m128i);
__m128i __lsx_vssrln_wu_d (__m128i, __m128i);
__m128i __lsx_vssrlrn_b_h (__m128i, __m128i);
__m128i __lsx_vssrlrn_bu_h (__m128i, __m128i);
__m128i __lsx_vssrlrn_hu_w (__m128i, __m128i);
__m128i __lsx_vssrlrn_h_w (__m128i, __m128i);
__m128i __lsx_vssrlrni_b_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrlrni_bu_h (__m128i, __m128i, imm0_15);
__m128i __lsx_vssrlrni_d_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrlrni_du_q (__m128i, __m128i, imm0_127);
__m128i __lsx_vssrlrni_hu_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrlrni_h_w (__m128i, __m128i, imm0_31);
__m128i __lsx_vssrlrni_w_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssrlrni_wu_d (__m128i, __m128i, imm0_63);
__m128i __lsx_vssrlrn_w_d (__m128i, __m128i);
__m128i __lsx_vssrlrn_wu_d (__m128i, __m128i);
__m128i __lsx_vssub_b (__m128i, __m128i);
__m128i __lsx_vssub_bu (__m128i, __m128i);
__m128i __lsx_vssub_d (__m128i, __m128i);
__m128i __lsx_vssub_du (__m128i, __m128i);
__m128i __lsx_vssub_h (__m128i, __m128i);

```

```

__m128i __lsx_vssub_hu (__m128i, __m128i);
__m128i __lsx_vssub_w (__m128i, __m128i);
__m128i __lsx_vssub_wu (__m128i, __m128i);
void __lsx_vst (__m128i, void *, imm_n2048_2047);
void __lsx_vstelm_b (__m128i, void *, imm_n128_127, imm0_15);
void __lsx_vstelm_d (__m128i, void *, imm_n128_127, imm0_1);
void __lsx_vstelm_h (__m128i, void *, imm_n128_127, imm0_7);
void __lsx_vstelm_w (__m128i, void *, imm_n128_127, imm0_3);
void __lsx_vstx (__m128i, void *, long int);
__m128i __lsx_vsub_b (__m128i, __m128i);
__m128i __lsx_vsub_d (__m128i, __m128i);
__m128i __lsx_vsub_h (__m128i, __m128i);
__m128i __lsx_vsubi_bu (__m128i, imm0_31);
__m128i __lsx_vsubi_du (__m128i, imm0_31);
__m128i __lsx_vsubi_hu (__m128i, imm0_31);
__m128i __lsx_vsubi_wu (__m128i, imm0_31);
__m128i __lsx_vsub_q (__m128i, __m128i);
__m128i __lsx_vsub_w (__m128i, __m128i);
__m128i __lsx_vsubwev_d_w (__m128i, __m128i);
__m128i __lsx_vsubwev_d_wu (__m128i, __m128i);
__m128i __lsx_vsubwev_h_b (__m128i, __m128i);
__m128i __lsx_vsubwev_h_bu (__m128i, __m128i);
__m128i __lsx_vsubwev_q_d (__m128i, __m128i);
__m128i __lsx_vsubwev_q_du (__m128i, __m128i);
__m128i __lsx_vsubwev_w_h (__m128i, __m128i);
__m128i __lsx_vsubwev_w_hu (__m128i, __m128i);
__m128i __lsx_vsubwod_d_w (__m128i, __m128i);
__m128i __lsx_vsubwod_d_wu (__m128i, __m128i);
__m128i __lsx_vsubwod_h_b (__m128i, __m128i);
__m128i __lsx_vsubwod_h_bu (__m128i, __m128i);
__m128i __lsx_vsubwod_q_d (__m128i, __m128i);
__m128i __lsx_vsubwod_q_du (__m128i, __m128i);
__m128i __lsx_vsubwod_w_h (__m128i, __m128i);
__m128i __lsx_vsubwod_w_hu (__m128i, __m128i);
__m128i __lsx_vxori_b (__m128i, imm0_255);
__m128i __lsx_vxor_v (__m128i, __m128i);

```

7.13.13.3 Directly-mapped SX Division Builtin Functions

These intrinsic functions are available by including `lsxintrin.h` and using `-mfrecipe` and `-mlsx`.

```

__m128d __lsx_vfrecipe_d (__m128d);
__m128 __lsx_vfrecipe_s (__m128);
__m128d __lsx_vfrsqrt_d (__m128d);
__m128 __lsx_vfrsqrt_s (__m128);

```

7.13.14 LoongArch ASX Vector Intrinsics

GCC provides intrinsics to access the LASX (Loongson Advanced SIMD Extension) instructions. The interface is made available by including `<lasxintrin.h>` and using `-mlasx`.

7.13.14.1 ASX Data Types

The following vectors typedefs are included in `lasxintrin.h`:

- `__m256i`, a 256-bit vector of fixed point;
- `__m256`, a 256-bit vector of single precision floating point;
- `__m256d`, a 256-bit vector of double precision floating point.

Instructions and corresponding built-ins may have additional restrictions and/or input/output values manipulated:

- `imm0_1`, an integer literal in range 0 to 1.
- `imm0_3`, an integer literal in range 0 to 3.
- `imm0_7`, an integer literal in range 0 to 7.
- `imm0_15`, an integer literal in range 0 to 15.
- `imm0_31`, an integer literal in range 0 to 31.
- `imm0_63`, an integer literal in range 0 to 63.
- `imm0_127`, an integer literal in range 0 to 127.
- `imm0_255`, an integer literal in range 0 to 255.
- `imm_n16_15`, an integer literal in range -16 to 15.
- `imm_n128_127`, an integer literal in range -128 to 127.
- `imm_n256_255`, an integer literal in range -256 to 255.
- `imm_n512_511`, an integer literal in range -512 to 511.
- `imm_n1024_1023`, an integer literal in range -1024 to 1023.
- `imm_n2048_2047`, an integer literal in range -2048 to 2047.

7.13.14.2 Directly-mapped ASX Builtin Functions

For convenience, GCC defines functions `__lasx_xvrepli_{b/h/w/d}` and `__lasx_b[n]z_{v/b/h/w/d}`, which are implemented as follows:

- a. `__lasx_xvrepli_{b/h/w/d}`: Implemented the case where the highest bit of `xvldi` instruction `i13` is 1.

```
i13[12] == 1'b0
case i13[11:10] of :
    2'b00: __lasx_xvrepli_b (imm_n512_511)
    2'b01: __lasx_xvrepli_h (imm_n512_511)
    2'b10: __lasx_xvrepli_w (imm_n512_511)
    2'b11: __lasx_xvrepli_d (imm_n512_511)
```

- b. `__lasx_b[n]z_{v/b/h/w/d}`: Since the `xvseteqz` class directive cannot be used on its own, this function is defined.

```
__lasx_xbz_v => xvseteqz.v + bcnez
__lasx_xbnz_v => xvsetnez.v + bcnez
__lasx_xbz_b => xvsetanyeqz.b + bcnez
__lasx_xbz_h => xvsetanyeqz.h + bcnez
__lasx_xbz_w => xvsetanyeqz.w + bcnez
__lasx_xbz_d => xvsetanyeqz.d + bcnez
__lasx_xbnz_b => xvsetallnez.b + bcnez
__lasx_xbnz_h => xvsetallnez.h + bcnez
__lasx_xbnz_w => xvsetallnez.w + bcnez
__lasx_xbnz_d => xvsetallnez.d + bcnez
```

eg:

```
#include <lasxintrin.h>
```

```
extern __m256i a;
```

```
void
```

```

test (void)
{
    if (__lasx_xbz_v (a))
        printf ("1\n");
    else
        printf ("2\n");
}

```

Note: For directives where the intent operand is also the source operand (modifying only part of the bitfield of the intent register), the first parameter in the builtin call function is used as the intent operand.

```

eg:
#include <lasxintrin.h>
extern __m256i dst;
int src;

void
test (void)
{
    dst = __lasx_xvinsgr2vr_w (dst, src, 3);
}

```

The intrinsics provided are listed below:

```

__m256i __lasx_vext2xv_d_b (__m256i);
__m256i __lasx_vext2xv_d_h (__m256i);
__m256i __lasx_vext2xv_du_bu (__m256i);
__m256i __lasx_vext2xv_du_hu (__m256i);
__m256i __lasx_vext2xv_du_wu (__m256i);
__m256i __lasx_vext2xv_d_w (__m256i);
__m256i __lasx_vext2xv_h_b (__m256i);
__m256i __lasx_vext2xv_hu_bu (__m256i);
__m256i __lasx_vext2xv_w_b (__m256i);
__m256i __lasx_vext2xv_w_h (__m256i);
__m256i __lasx_vext2xv_wu_bu (__m256i);
__m256i __lasx_vext2xv_wu_hu (__m256i);
int __lasx_xbnz_b (__m256i);
int __lasx_xbnz_d (__m256i);
int __lasx_xbnz_h (__m256i);
int __lasx_xbnz_v (__m256i);
int __lasx_xbnz_w (__m256i);
int __lasx_xbz_b (__m256i);
int __lasx_xbz_d (__m256i);
int __lasx_xbz_h (__m256i);
int __lasx_xbz_v (__m256i);
int __lasx_xbz_w (__m256i);
__m256i __lasx_xvabsd_b (__m256i, __m256i);
__m256i __lasx_xvabsd_bu (__m256i, __m256i);
__m256i __lasx_xvabsd_d (__m256i, __m256i);
__m256i __lasx_xvabsd_du (__m256i, __m256i);
__m256i __lasx_xvabsd_h (__m256i, __m256i);
__m256i __lasx_xvabsd_hu (__m256i, __m256i);
__m256i __lasx_xvabsd_w (__m256i, __m256i);
__m256i __lasx_xvabsd_wu (__m256i, __m256i);
__m256i __lasx_xvadda_b (__m256i, __m256i);
__m256i __lasx_xvadda_d (__m256i, __m256i);
__m256i __lasx_xvadda_h (__m256i, __m256i);
__m256i __lasx_xvadda_w (__m256i, __m256i);
__m256i __lasx_xvadd_b (__m256i, __m256i);

```

```

__m256i __lasx_xvadd_d (__m256i, __m256i);
__m256i __lasx_xvadd_h (__m256i, __m256i);
__m256i __lasx_xvaddi_bu (__m256i, imm0_31);
__m256i __lasx_xvaddi_du (__m256i, imm0_31);
__m256i __lasx_xvaddi_hu (__m256i, imm0_31);
__m256i __lasx_xvaddi_wu (__m256i, imm0_31);
__m256i __lasx_xvadd_q (__m256i, __m256i);
__m256i __lasx_xvadd_w (__m256i, __m256i);
__m256i __lasx_xvaddwev_d_w (__m256i, __m256i);
__m256i __lasx_xvaddwev_d_wu (__m256i, __m256i);
__m256i __lasx_xvaddwev_d_wu_w (__m256i, __m256i);
__m256i __lasx_xvaddwev_h_b (__m256i, __m256i);
__m256i __lasx_xvaddwev_h_bu (__m256i, __m256i);
__m256i __lasx_xvaddwev_h_bu_b (__m256i, __m256i);
__m256i __lasx_xvaddwev_q_d (__m256i, __m256i);
__m256i __lasx_xvaddwev_q_du (__m256i, __m256i);
__m256i __lasx_xvaddwev_q_du_d (__m256i, __m256i);
__m256i __lasx_xvaddwev_w_h (__m256i, __m256i);
__m256i __lasx_xvaddwev_w_hu (__m256i, __m256i);
__m256i __lasx_xvaddwev_w_hu_h (__m256i, __m256i);
__m256i __lasx_xvaddwod_d_w (__m256i, __m256i);
__m256i __lasx_xvaddwod_d_wu (__m256i, __m256i);
__m256i __lasx_xvaddwod_d_wu_w (__m256i, __m256i);
__m256i __lasx_xvaddwod_h_b (__m256i, __m256i);
__m256i __lasx_xvaddwod_h_bu (__m256i, __m256i);
__m256i __lasx_xvaddwod_h_bu_b (__m256i, __m256i);
__m256i __lasx_xvaddwod_q_d (__m256i, __m256i);
__m256i __lasx_xvaddwod_q_du (__m256i, __m256i);
__m256i __lasx_xvaddwod_q_du_d (__m256i, __m256i);
__m256i __lasx_xvaddwod_w_h (__m256i, __m256i);
__m256i __lasx_xvaddwod_w_hu (__m256i, __m256i);
__m256i __lasx_xvaddwod_w_hu_h (__m256i, __m256i);
__m256i __lasx_xvandi_b (__m256i, imm0_255);
__m256i __lasx_xvandn_v (__m256i, __m256i);
__m256i __lasx_xvand_v (__m256i, __m256i);
__m256i __lasx_xvavg_b (__m256i, __m256i);
__m256i __lasx_xvavg_bu (__m256i, __m256i);
__m256i __lasx_xvavg_d (__m256i, __m256i);
__m256i __lasx_xvavg_du (__m256i, __m256i);
__m256i __lasx_xvavg_h (__m256i, __m256i);
__m256i __lasx_xvavg_hu (__m256i, __m256i);
__m256i __lasx_xvavgr_b (__m256i, __m256i);
__m256i __lasx_xvavgr_bu (__m256i, __m256i);
__m256i __lasx_xvavgr_d (__m256i, __m256i);
__m256i __lasx_xvavgr_du (__m256i, __m256i);
__m256i __lasx_xvavgr_h (__m256i, __m256i);
__m256i __lasx_xvavgr_hu (__m256i, __m256i);
__m256i __lasx_xvavgr_w (__m256i, __m256i);
__m256i __lasx_xvavgr_wu (__m256i, __m256i);
__m256i __lasx_xvavg_w (__m256i, __m256i);
__m256i __lasx_xvavg_wu (__m256i, __m256i);
__m256i __lasx_xvbitclr_b (__m256i, __m256i);
__m256i __lasx_xvbitclr_d (__m256i, __m256i);
__m256i __lasx_xvbitclr_h (__m256i, __m256i);
__m256i __lasx_xvbitclri_b (__m256i, imm0_7);
__m256i __lasx_xvbitclri_d (__m256i, imm0_63);
__m256i __lasx_xvbitclri_h (__m256i, imm0_15);
__m256i __lasx_xvbitclri_w (__m256i, imm0_31);

```

```

__m256i __lasx_xvbitclr_w (__m256i, __m256i);
__m256i __lasx_xvbitrev_b (__m256i, __m256i);
__m256i __lasx_xvbitrev_d (__m256i, __m256i);
__m256i __lasx_xvbitrev_h (__m256i, __m256i);
__m256i __lasx_xvbitrevi_b (__m256i, imm0_7);
__m256i __lasx_xvbitrevi_d (__m256i, imm0_63);
__m256i __lasx_xvbitrevi_h (__m256i, imm0_15);
__m256i __lasx_xvbitrevi_w (__m256i, imm0_31);
__m256i __lasx_xvbitrev_w (__m256i, __m256i);
__m256i __lasx_xvbitseli_b (__m256i, __m256i, imm0_255);
__m256i __lasx_xvbitsel_v (__m256i, __m256i, __m256i);
__m256i __lasx_xvbitset_b (__m256i, __m256i);
__m256i __lasx_xvbitset_d (__m256i, __m256i);
__m256i __lasx_xvbitset_h (__m256i, __m256i);
__m256i __lasx_xvbitseti_b (__m256i, imm0_7);
__m256i __lasx_xvbitseti_d (__m256i, imm0_63);
__m256i __lasx_xvbitseti_h (__m256i, imm0_15);
__m256i __lasx_xvbitseti_w (__m256i, imm0_31);
__m256i __lasx_xvbitset_w (__m256i, __m256i);
__m256i __lasx_xvbsll_v (__m256i, imm0_31);
__m256i __lasx_xvbsrl_v (__m256i, imm0_31);
__m256i __lasx_xvclo_b (__m256i);
__m256i __lasx_xvclo_d (__m256i);
__m256i __lasx_xvclo_h (__m256i);
__m256i __lasx_xvclo_w (__m256i);
__m256i __lasx_xvclz_b (__m256i);
__m256i __lasx_xvclz_d (__m256i);
__m256i __lasx_xvclz_h (__m256i);
__m256i __lasx_xvclz_w (__m256i);
__m256i __lasx_xvdiv_b (__m256i, __m256i);
__m256i __lasx_xvdiv_bu (__m256i, __m256i);
__m256i __lasx_xvdiv_d (__m256i, __m256i);
__m256i __lasx_xvdiv_du (__m256i, __m256i);
__m256i __lasx_xvdiv_h (__m256i, __m256i);
__m256i __lasx_xvdiv_hu (__m256i, __m256i);
__m256i __lasx_xvdiv_w (__m256i, __m256i);
__m256i __lasx_xvdiv_wu (__m256i, __m256i);
__m256i __lasx_xvexth_du_wu (__m256i);
__m256i __lasx_xvexth_d_w (__m256i);
__m256i __lasx_xvexth_h_b (__m256i);
__m256i __lasx_xvexth_hu_bu (__m256i);
__m256i __lasx_xvexth_q_d (__m256i);
__m256i __lasx_xvexth_qu_du (__m256i);
__m256i __lasx_xvexth_w_h (__m256i);
__m256i __lasx_xvexth_wu_hu (__m256i);
__m256i __lasx_xvextl_q_d (__m256i);
__m256i __lasx_xvextl_qu_du (__m256i);
__m256i __lasx_xvextrins_b (__m256i, __m256i, imm0_255);
__m256i __lasx_xvextrins_d (__m256i, __m256i, imm0_255);
__m256i __lasx_xvextrins_h (__m256i, __m256i, imm0_255);
__m256i __lasx_xvextrins_w (__m256i, __m256i, imm0_255);
__m256d __lasx_xvfadd_d (__m256d, __m256d);
__m256 __lasx_xvfadd_s (__m256, __m256);
__m256i __lasx_xvfclass_d (__m256d);
__m256i __lasx_xvfclass_s (__m256);
__m256i __lasx_xvfcmp_caf_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_caf_s (__m256, __m256);
__m256i __lasx_xvfcmp_ceq_d (__m256d, __m256d);

```

```

__m256i __lasx_xvfcmp_ceq_s (__m256, __m256);
__m256i __lasx_xvfcmp_cle_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cle_s (__m256, __m256);
__m256i __lasx_xvfcmp_clt_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_clt_s (__m256, __m256);
__m256i __lasx_xvfcmp_cne_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cne_s (__m256, __m256);
__m256i __lasx_xvfcmp_cor_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cor_s (__m256, __m256);
__m256i __lasx_xvfcmp_cueq_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cueq_s (__m256, __m256);
__m256i __lasx_xvfcmp_cule_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cule_s (__m256, __m256);
__m256i __lasx_xvfcmp_cult_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cult_s (__m256, __m256);
__m256i __lasx_xvfcmp_cun_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cune_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_cune_s (__m256, __m256);
__m256i __lasx_xvfcmp_cun_s (__m256, __m256);
__m256i __lasx_xvfcmp_saf_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_saf_s (__m256, __m256);
__m256i __lasx_xvfcmp_seq_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_seq_s (__m256, __m256);
__m256i __lasx_xvfcmp_sle_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sle_s (__m256, __m256);
__m256i __lasx_xvfcmp_slt_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_slt_s (__m256, __m256);
__m256i __lasx_xvfcmp_sne_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sne_s (__m256, __m256);
__m256i __lasx_xvfcmp_sor_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sor_s (__m256, __m256);
__m256i __lasx_xvfcmp_sueq_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sueq_s (__m256, __m256);
__m256i __lasx_xvfcmp_sule_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sule_s (__m256, __m256);
__m256i __lasx_xvfcmp_sult_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sult_s (__m256, __m256);
__m256i __lasx_xvfcmp_sun_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sune_d (__m256d, __m256d);
__m256i __lasx_xvfcmp_sune_s (__m256, __m256);
__m256i __lasx_xvfcmp_sun_s (__m256, __m256);
__m256d __lasx_xvfcvth_d_s (__m256);
__m256i __lasx_xvfcvt_h_s (__m256, __m256);
__m256 __lasx_xvfcvth_s_h (__m256i);
__m256d __lasx_xvfcvth_d_s (__m256);
__m256 __lasx_xvfcvth_s_h (__m256i);
__m256 __lasx_xvfcvt_s_d (__m256d, __m256d);
__m256d __lasx_xvfdiv_d (__m256d, __m256d);
__m256 __lasx_xvfdiv_s (__m256, __m256);
__m256d __lasx_xvffint_d_l (__m256i);
__m256d __lasx_xvffint_d_lu (__m256i);
__m256d __lasx_xvffint_d_w (__m256i);
__m256d __lasx_xvffintl_d_w (__m256i);
__m256 __lasx_xvffint_s_l (__m256i, __m256i);
__m256 __lasx_xvffint_s_w (__m256i);
__m256 __lasx_xvffint_s_wu (__m256i);
__m256d __lasx_xvflogb_d (__m256d);
__m256 __lasx_xvflogb_s (__m256);

```

```

__m256d __lasx_xvfmadd_d (__m256d, __m256d, __m256d);
__m256 __lasx_xvfmadd_s (__m256, __m256, __m256);
__m256d __lasx_xvfmaxa_d (__m256d, __m256d);
__m256 __lasx_xvfmaxa_s (__m256, __m256);
__m256d __lasx_xvfmax_d (__m256d, __m256d);
__m256 __lasx_xvfmax_s (__m256, __m256);
__m256d __lasx_xvfmin_d (__m256d, __m256d);
__m256 __lasx_xvfmin_s (__m256, __m256);
__m256d __lasx_xvfmsub_d (__m256d, __m256d, __m256d);
__m256 __lasx_xvfmsub_s (__m256, __m256, __m256);
__m256d __lasx_xvfmul_d (__m256d, __m256d);
__m256 __lasx_xvfmul_s (__m256, __m256);
__m256d __lasx_xvfnmadd_d (__m256d, __m256d, __m256d);
__m256 __lasx_xvfnmadd_s (__m256, __m256, __m256);
__m256d __lasx_xvfnmsub_d (__m256d, __m256d, __m256d);
__m256 __lasx_xvfnmsub_s (__m256, __m256, __m256);
__m256d __lasx_xvfrecip_d (__m256d);
__m256 __lasx_xvfrecip_s (__m256);
__m256d __lasx_xvfrint_d (__m256d);
__m256d __lasx_xvfrintrm_d (__m256d);
__m256 __lasx_xvfrintrm_s (__m256);
__m256d __lasx_xvfrintrne_d (__m256d);
__m256 __lasx_xvfrintrne_s (__m256);
__m256d __lasx_xvfrintrp_d (__m256d);
__m256 __lasx_xvfrintrp_s (__m256);
__m256d __lasx_xvfrintrz_d (__m256d);
__m256 __lasx_xvfrintrz_s (__m256);
__m256 __lasx_xvfrint_s (__m256);
__m256d __lasx_xvfrsqrt_d (__m256d);
__m256 __lasx_xvfrsqrt_s (__m256);
__m256i __lasx_xvfrstp_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvfrstp_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvfrstpi_b (__m256i, __m256i, imm0_31);
__m256i __lasx_xvfrstpi_h (__m256i, __m256i, imm0_31);
__m256d __lasx_xvfsqrt_d (__m256d);
__m256 __lasx_xvfsqrt_s (__m256);
__m256d __lasx_xvfsub_d (__m256d, __m256d);
__m256 __lasx_xvfsub_s (__m256, __m256);
__m256i __lasx_xvftint_l_s (__m256i);
__m256i __lasx_xvftint_l_d (__m256d);
__m256i __lasx_xvftintl_l_s (__m256i);
__m256i __lasx_xvftint_lu_d (__m256d);
__m256i __lasx_xvftintrmh_l_s (__m256i);
__m256i __lasx_xvftintrm_l_d (__m256d);
__m256i __lasx_xvftintrml_l_s (__m256i);
__m256i __lasx_xvftintrm_w_d (__m256d, __m256d);
__m256i __lasx_xvftintrm_w_s (__m256i);
__m256i __lasx_xvftintrneh_l_s (__m256i);
__m256i __lasx_xvftintrne_l_d (__m256d);
__m256i __lasx_xvftintrnel_l_s (__m256i);
__m256i __lasx_xvftintrne_w_d (__m256d, __m256d);
__m256i __lasx_xvftintrne_w_s (__m256i);
__m256i __lasx_xvftintrph_l_s (__m256i);
__m256i __lasx_xvftintrp_l_d (__m256d);
__m256i __lasx_xvftintrpl_l_s (__m256i);
__m256i __lasx_xvftintrp_w_d (__m256d, __m256d);

```

```

__m256i __lasx_xvftintrp_w_s (__m256);
__m256i __lasx_xvftintrzh_l_s (__m256);
__m256i __lasx_xvftintrz_l_d (__m256d);
__m256i __lasx_xvftintrzl_l_s (__m256);
__m256i __lasx_xvftintrz_lu_d (__m256d);
__m256i __lasx_xvftintrz_w_d (__m256d, __m256d);
__m256i __lasx_xvftintrz_w_s (__m256);
__m256i __lasx_xvftintrz_wu_s (__m256);
__m256i __lasx_xvftint_w_d (__m256d, __m256d);
__m256i __lasx_xvftint_w_s (__m256);
__m256i __lasx_xvftint_wu_s (__m256);
__m256i __lasx_xvhaddw_du_wu (__m256i, __m256i);
__m256i __lasx_xvhaddw_d_w (__m256i, __m256i);
__m256i __lasx_xvhaddw_h_b (__m256i, __m256i);
__m256i __lasx_xvhaddw_hu_bu (__m256i, __m256i);
__m256i __lasx_xvhaddw_q_d (__m256i, __m256i);
__m256i __lasx_xvhaddw_qu_du (__m256i, __m256i);
__m256i __lasx_xvhaddw_w_h (__m256i, __m256i);
__m256i __lasx_xvhaddw_wu_hu (__m256i, __m256i);
__m256i __lasx_xvhsubw_du_wu (__m256i, __m256i);
__m256i __lasx_xvhsubw_d_w (__m256i, __m256i);
__m256i __lasx_xvhsubw_h_b (__m256i, __m256i);
__m256i __lasx_xvhsubw_hu_bu (__m256i, __m256i);
__m256i __lasx_xvhsubw_q_d (__m256i, __m256i);
__m256i __lasx_xvhsubw_qu_du (__m256i, __m256i);
__m256i __lasx_xvhsubw_w_h (__m256i, __m256i);
__m256i __lasx_xvhsubw_wu_hu (__m256i, __m256i);
__m256i __lasx_xvilvh_b (__m256i, __m256i);
__m256i __lasx_xvilvh_d (__m256i, __m256i);
__m256i __lasx_xvilvh_h (__m256i, __m256i);
__m256i __lasx_xvilvh_w (__m256i, __m256i);
__m256i __lasx_xvilvl_b (__m256i, __m256i);
__m256i __lasx_xvilvl_d (__m256i, __m256i);
__m256i __lasx_xvilvl_h (__m256i, __m256i);
__m256i __lasx_xvilvl_w (__m256i, __m256i);
__m256i __lasx_xvinsgr2vr_d (__m256i, long int, imm0_3);
__m256i __lasx_xvinsgr2vr_w (__m256i, int, imm0_7);
__m256i __lasx_xvinsve0_d (__m256i, __m256i, imm0_3);
__m256i __lasx_xvinsve0_w (__m256i, __m256i, imm0_7);
__m256i __lasx_xvld (void *, imm_n2048_2047);
__m256i __lasx_xvldi (imm_n1024_1023);
__m256i __lasx_xvldrepl_b (void *, imm_n2048_2047);
__m256i __lasx_xvldrepl_d (void *, imm_n256_255);
__m256i __lasx_xvldrepl_h (void *, imm_n1024_1023);
__m256i __lasx_xvldrepl_w (void *, imm_n512_511);
__m256i __lasx_xvldx (void *, long int);
__m256i __lasx_xvmadd_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvmadd_d (__m256i, __m256i, __m256i);
__m256i __lasx_xvmadd_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvmadd_w (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_d_w (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_d_wu (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_d_wu_w (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_h_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_h_bu (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_h_bu_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_q_d (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_q_du (__m256i, __m256i, __m256i);

```

```

__m256i __lasx_xvmaddwev_q_du_d (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_w_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_w_hu (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwev_w_hu_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_d_w (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_d_wu (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_d_wu_w (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_h_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_h_bu (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_h_bu_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_q_d (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_q_du (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_q_du_d (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_w_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_w_hu (__m256i, __m256i, __m256i);
__m256i __lasx_xvmaddwod_w_hu_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvmax_b (__m256i, __m256i);
__m256i __lasx_xvmax_bu (__m256i, __m256i);
__m256i __lasx_xvmax_d (__m256i, __m256i);
__m256i __lasx_xvmax_du (__m256i, __m256i);
__m256i __lasx_xvmax_h (__m256i, __m256i);
__m256i __lasx_xvmax_hu (__m256i, __m256i);
__m256i __lasx_xvmaxi_b (__m256i, imm_n16_15);
__m256i __lasx_xvmaxi_bu (__m256i, imm0_31);
__m256i __lasx_xvmaxi_d (__m256i, imm_n16_15);
__m256i __lasx_xvmaxi_du (__m256i, imm0_31);
__m256i __lasx_xvmaxi_h (__m256i, imm_n16_15);
__m256i __lasx_xvmaxi_hu (__m256i, imm0_31);
__m256i __lasx_xvmaxi_w (__m256i, imm_n16_15);
__m256i __lasx_xvmaxi_wu (__m256i, imm0_31);
__m256i __lasx_xvmax_w (__m256i, __m256i);
__m256i __lasx_xvmax_wu (__m256i, __m256i);
__m256i __lasx_xvmin_b (__m256i, __m256i);
__m256i __lasx_xvmin_bu (__m256i, __m256i);
__m256i __lasx_xvmin_d (__m256i, __m256i);
__m256i __lasx_xvmin_du (__m256i, __m256i);
__m256i __lasx_xvmin_h (__m256i, __m256i);
__m256i __lasx_xvmin_hu (__m256i, __m256i);
__m256i __lasx_xvmini_b (__m256i, imm_n16_15);
__m256i __lasx_xvmini_bu (__m256i, imm0_31);
__m256i __lasx_xvmini_d (__m256i, imm_n16_15);
__m256i __lasx_xvmini_du (__m256i, imm0_31);
__m256i __lasx_xvmini_h (__m256i, imm_n16_15);
__m256i __lasx_xvmini_hu (__m256i, imm0_31);
__m256i __lasx_xvmini_w (__m256i, imm_n16_15);
__m256i __lasx_xvmini_wu (__m256i, imm0_31);
__m256i __lasx_xvmin_w (__m256i, __m256i);
__m256i __lasx_xvmin_wu (__m256i, __m256i);
__m256i __lasx_xvmod_b (__m256i, __m256i);
__m256i __lasx_xvmod_bu (__m256i, __m256i);
__m256i __lasx_xvmod_d (__m256i, __m256i);
__m256i __lasx_xvmod_du (__m256i, __m256i);
__m256i __lasx_xvmod_h (__m256i, __m256i);
__m256i __lasx_xvmod_hu (__m256i, __m256i);
__m256i __lasx_xvmod_w (__m256i, __m256i);
__m256i __lasx_xvmod_wu (__m256i, __m256i);
__m256i __lasx_xvmskgez_b (__m256i);
__m256i __lasx_xvmskltz_b (__m256i);

```

```

__m256i __lasx_xvmskltz_d (__m256i);
__m256i __lasx_xvmskltz_h (__m256i);
__m256i __lasx_xvmskltz_w (__m256i);
__m256i __lasx_xvmsknz_b (__m256i);
__m256i __lasx_xvmsub_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvmsub_d (__m256i, __m256i, __m256i);
__m256i __lasx_xvmsub_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvmsub_w (__m256i, __m256i, __m256i);
__m256i __lasx_xvmuh_b (__m256i, __m256i);
__m256i __lasx_xvmuh_bu (__m256i, __m256i);
__m256i __lasx_xvmuh_d (__m256i, __m256i);
__m256i __lasx_xvmuh_du (__m256i, __m256i);
__m256i __lasx_xvmuh_h (__m256i, __m256i);
__m256i __lasx_xvmuh_hu (__m256i, __m256i);
__m256i __lasx_xvmuh_w (__m256i, __m256i);
__m256i __lasx_xvmuh_wu (__m256i, __m256i);
__m256i __lasx_xvmul_b (__m256i, __m256i);
__m256i __lasx_xvmul_d (__m256i, __m256i);
__m256i __lasx_xvmul_h (__m256i, __m256i);
__m256i __lasx_xvmul_w (__m256i, __m256i);
__m256i __lasx_xvmulwev_d_w (__m256i, __m256i);
__m256i __lasx_xvmulwev_d_wu (__m256i, __m256i);
__m256i __lasx_xvmulwev_d_wu_w (__m256i, __m256i);
__m256i __lasx_xvmulwev_h_b (__m256i, __m256i);
__m256i __lasx_xvmulwev_h_bu (__m256i, __m256i);
__m256i __lasx_xvmulwev_h_bu_b (__m256i, __m256i);
__m256i __lasx_xvmulwev_q_d (__m256i, __m256i);
__m256i __lasx_xvmulwev_q_du (__m256i, __m256i);
__m256i __lasx_xvmulwev_q_du_d (__m256i, __m256i);
__m256i __lasx_xvmulwev_w_h (__m256i, __m256i);
__m256i __lasx_xvmulwev_w_hu (__m256i, __m256i);
__m256i __lasx_xvmulwev_w_hu_h (__m256i, __m256i);
__m256i __lasx_xvmulwod_d_w (__m256i, __m256i);
__m256i __lasx_xvmulwod_d_wu (__m256i, __m256i);
__m256i __lasx_xvmulwod_d_wu_w (__m256i, __m256i);
__m256i __lasx_xvmulwod_h_b (__m256i, __m256i);
__m256i __lasx_xvmulwod_h_bu (__m256i, __m256i);
__m256i __lasx_xvmulwod_h_bu_b (__m256i, __m256i);
__m256i __lasx_xvmulwod_q_d (__m256i, __m256i);
__m256i __lasx_xvmulwod_q_du (__m256i, __m256i);
__m256i __lasx_xvmulwod_q_du_d (__m256i, __m256i);
__m256i __lasx_xvmulwod_w_h (__m256i, __m256i);
__m256i __lasx_xvmulwod_w_hu (__m256i, __m256i);
__m256i __lasx_xvmulwod_w_hu_h (__m256i, __m256i);
__m256i __lasx_xvneg_b (__m256i);
__m256i __lasx_xvneg_d (__m256i);
__m256i __lasx_xvneg_h (__m256i);
__m256i __lasx_xvneg_w (__m256i);
__m256i __lasx_xvnori_b (__m256i, imm0_255);
__m256i __lasx_xvnor_v (__m256i, __m256i);
__m256i __lasx_xvori_b (__m256i, imm0_255);
__m256i __lasx_xvorn_v (__m256i, __m256i);
__m256i __lasx_xvor_v (__m256i, __m256i);
__m256i __lasx_xvpackev_b (__m256i, __m256i);
__m256i __lasx_xvpackev_d (__m256i, __m256i);
__m256i __lasx_xvpackev_h (__m256i, __m256i);
__m256i __lasx_xvpackev_w (__m256i, __m256i);
__m256i __lasx_xvpackod_b (__m256i, __m256i);

```

```

__m256i __lasx_xvpackod_d (__m256i, __m256i);
__m256i __lasx_xvpackod_h (__m256i, __m256i);
__m256i __lasx_xvpackod_w (__m256i, __m256i);
__m256i __lasx_xvpcnt_b (__m256i);
__m256i __lasx_xvpcnt_d (__m256i);
__m256i __lasx_xvpcnt_h (__m256i);
__m256i __lasx_xvpcnt_w (__m256i);
__m256i __lasx_xvpermi_d (__m256i, imm0_255);
__m256i __lasx_xvpermi_q (__m256i, __m256i, imm0_255);
__m256i __lasx_xvpermi_w (__m256i, __m256i, imm0_255);
__m256i __lasx_xvperm_w (__m256i, __m256i);
__m256i __lasx_xvpickve_b (__m256i, __m256i);
__m256i __lasx_xvpickve_d (__m256i, __m256i);
__m256i __lasx_xvpickve_h (__m256i, __m256i);
__m256i __lasx_xvpickve_w (__m256i, __m256i);
__m256i __lasx_xvpickod_b (__m256i, __m256i);
__m256i __lasx_xvpickod_d (__m256i, __m256i);
__m256i __lasx_xvpickod_h (__m256i, __m256i);
__m256i __lasx_xvpickod_w (__m256i, __m256i);
long int __lasx_xvpickve2gr_d (__m256i, imm0_3);
unsigned long int __lasx_xvpickve2gr_du (__m256i, imm0_3);
int __lasx_xvpickve2gr_w (__m256i, imm0_7);
unsigned int __lasx_xvpickve2gr_wu (__m256i, imm0_7);
__m256i __lasx_xvpickve_d (__m256i, imm0_3);
__m256d __lasx_xvpickve_d_f (__m256d, imm0_3);
__m256i __lasx_xvpickve_w (__m256i, imm0_7);
__m256 __lasx_xvpickve_w_f (__m256, imm0_7);
__m256i __lasx_xvrepl128vei_b (__m256i, imm0_15);
__m256i __lasx_xvrepl128vei_d (__m256i, imm0_1);
__m256i __lasx_xvrepl128vei_h (__m256i, imm0_7);
__m256i __lasx_xvrepl128vei_w (__m256i, imm0_3);
__m256i __lasx_xvreplgr2vr_b (int);
__m256i __lasx_xvreplgr2vr_d (long int);
__m256i __lasx_xvreplgr2vr_h (int);
__m256i __lasx_xvreplgr2vr_w (int);
__m256i __lasx_xvrepli_b (imm_n512_511);
__m256i __lasx_xvrepli_d (imm_n512_511);
__m256i __lasx_xvrepli_h (imm_n512_511);
__m256i __lasx_xvrepli_w (imm_n512_511);
__m256i __lasx_xvreplve0_b (__m256i);
__m256i __lasx_xvreplve0_d (__m256i);
__m256i __lasx_xvreplve0_h (__m256i);
__m256i __lasx_xvreplve0_q (__m256i);
__m256i __lasx_xvreplve0_w (__m256i);
__m256i __lasx_xvreplve_b (__m256i, int);
__m256i __lasx_xvreplve_d (__m256i, int);
__m256i __lasx_xvreplve_h (__m256i, int);
__m256i __lasx_xvreplve_w (__m256i, int);
__m256i __lasx_xvrotr_b (__m256i, __m256i);
__m256i __lasx_xvrotr_d (__m256i, __m256i);
__m256i __lasx_xvrotr_h (__m256i, __m256i);
__m256i __lasx_xvrotri_b (__m256i, imm0_7);
__m256i __lasx_xvrotri_d (__m256i, imm0_63);
__m256i __lasx_xvrotri_h (__m256i, imm0_15);
__m256i __lasx_xvrotri_w (__m256i, imm0_31);
__m256i __lasx_xvrotr_w (__m256i, __m256i);
__m256i __lasx_xvsadd_b (__m256i, __m256i);
__m256i __lasx_xvsadd_bu (__m256i, __m256i);

```

```

__m256i __lasx_xvsadd_d (__m256i, __m256i);
__m256i __lasx_xvsadd_du (__m256i, __m256i);
__m256i __lasx_xvsadd_h (__m256i, __m256i);
__m256i __lasx_xvsadd_hu (__m256i, __m256i);
__m256i __lasx_xvsadd_w (__m256i, __m256i);
__m256i __lasx_xvsadd_wu (__m256i, __m256i);
__m256i __lasx_xvsat_b (__m256i, imm0_7);
__m256i __lasx_xvsat_bu (__m256i, imm0_7);
__m256i __lasx_xvsat_d (__m256i, imm0_63);
__m256i __lasx_xvsat_du (__m256i, imm0_63);
__m256i __lasx_xvsat_h (__m256i, imm0_15);
__m256i __lasx_xvsat_hu (__m256i, imm0_15);
__m256i __lasx_xvsat_w (__m256i, imm0_31);
__m256i __lasx_xvsat_wu (__m256i, imm0_31);
__m256i __lasx_xvseq_b (__m256i, __m256i);
__m256i __lasx_xvseq_d (__m256i, __m256i);
__m256i __lasx_xvseq_h (__m256i, __m256i);
__m256i __lasx_xvseqi_b (__m256i, imm_n16_15);
__m256i __lasx_xvseqi_d (__m256i, imm_n16_15);
__m256i __lasx_xvseqi_h (__m256i, imm_n16_15);
__m256i __lasx_xvseqi_w (__m256i, imm_n16_15);
__m256i __lasx_xvseq_w (__m256i, __m256i);
__m256i __lasx_xvshuf4i_b (__m256i, imm0_255);
__m256i __lasx_xvshuf4i_d (__m256i, __m256i, imm0_255);
__m256i __lasx_xvshuf4i_h (__m256i, imm0_255);
__m256i __lasx_xvshuf4i_w (__m256i, imm0_255);
__m256i __lasx_xvshuf_b (__m256i, __m256i, __m256i);
__m256i __lasx_xvshuf_d (__m256i, __m256i, __m256i);
__m256i __lasx_xvshuf_h (__m256i, __m256i, __m256i);
__m256i __lasx_xvshuf_w (__m256i, __m256i, __m256i);
__m256i __lasx_xvsigncov_b (__m256i, __m256i);
__m256i __lasx_xvsigncov_d (__m256i, __m256i);
__m256i __lasx_xvsigncov_h (__m256i, __m256i);
__m256i __lasx_xvsigncov_w (__m256i, __m256i);
__m256i __lasx_xvsle_b (__m256i, __m256i);
__m256i __lasx_xvsle_bu (__m256i, __m256i);
__m256i __lasx_xvsle_d (__m256i, __m256i);
__m256i __lasx_xvsle_du (__m256i, __m256i);
__m256i __lasx_xvsle_h (__m256i, __m256i);
__m256i __lasx_xvsle_hu (__m256i, __m256i);
__m256i __lasx_xvslei_b (__m256i, imm_n16_15);
__m256i __lasx_xvslei_bu (__m256i, imm0_31);
__m256i __lasx_xvslei_d (__m256i, imm_n16_15);
__m256i __lasx_xvslei_du (__m256i, imm0_31);
__m256i __lasx_xvslei_h (__m256i, imm_n16_15);
__m256i __lasx_xvslei_hu (__m256i, imm0_31);
__m256i __lasx_xvslei_w (__m256i, imm_n16_15);
__m256i __lasx_xvslei_wu (__m256i, imm0_31);
__m256i __lasx_xvsle_w (__m256i, __m256i);
__m256i __lasx_xvsle_wu (__m256i, __m256i);
__m256i __lasx_xvsll_b (__m256i, __m256i);
__m256i __lasx_xvsll_d (__m256i, __m256i);
__m256i __lasx_xvsll_h (__m256i, __m256i);
__m256i __lasx_xvslli_b (__m256i, imm0_7);
__m256i __lasx_xvslli_d (__m256i, imm0_63);
__m256i __lasx_xvslli_h (__m256i, imm0_15);
__m256i __lasx_xvslli_w (__m256i, imm0_31);
__m256i __lasx_xvsll_w (__m256i, __m256i);

```

```

__m256i __lasx_xvslwllwil_du_wu (__m256i, imm0_31);
__m256i __lasx_xvslwllwil_d_w (__m256i, imm0_31);
__m256i __lasx_xvslwllwil_h_b (__m256i, imm0_7);
__m256i __lasx_xvslwllwil_hu_bu (__m256i, imm0_7);
__m256i __lasx_xvslwllwil_w_h (__m256i, imm0_15);
__m256i __lasx_xvslwllwil_wu_hu (__m256i, imm0_15);
__m256i __lasx_xvslt_b (__m256i, __m256i);
__m256i __lasx_xvslt_bu (__m256i, __m256i);
__m256i __lasx_xvslt_d (__m256i, __m256i);
__m256i __lasx_xvslt_du (__m256i, __m256i);
__m256i __lasx_xvslt_h (__m256i, __m256i);
__m256i __lasx_xvslt_hu (__m256i, __m256i);
__m256i __lasx_xvslti_b (__m256i, imm_n16_15);
__m256i __lasx_xvslti_bu (__m256i, imm0_31);
__m256i __lasx_xvslti_d (__m256i, imm_n16_15);
__m256i __lasx_xvslti_du (__m256i, imm0_31);
__m256i __lasx_xvslti_h (__m256i, imm_n16_15);
__m256i __lasx_xvslti_hu (__m256i, imm0_31);
__m256i __lasx_xvslti_w (__m256i, imm_n16_15);
__m256i __lasx_xvslti_wu (__m256i, imm0_31);
__m256i __lasx_xvslt_w (__m256i, __m256i);
__m256i __lasx_xvslt_wu (__m256i, __m256i);
__m256i __lasx_xvsra_b (__m256i, __m256i);
__m256i __lasx_xvsra_d (__m256i, __m256i);
__m256i __lasx_xvsra_h (__m256i, __m256i);
__m256i __lasx_xvsrai_b (__m256i, imm0_7);
__m256i __lasx_xvsrai_d (__m256i, imm0_63);
__m256i __lasx_xvsrai_h (__m256i, imm0_15);
__m256i __lasx_xvsrai_w (__m256i, imm0_31);
__m256i __lasx_xvsran_b_h (__m256i, __m256i);
__m256i __lasx_xvsran_h_w (__m256i, __m256i);
__m256i __lasx_xvsrani_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvsrani_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvsrani_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvsrani_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvsran_w_d (__m256i, __m256i);
__m256i __lasx_xvsrar_b (__m256i, __m256i);
__m256i __lasx_xvsrar_d (__m256i, __m256i);
__m256i __lasx_xvsrar_h (__m256i, __m256i);
__m256i __lasx_xvsrari_b (__m256i, imm0_7);
__m256i __lasx_xvsrari_d (__m256i, imm0_63);
__m256i __lasx_xvsrari_h (__m256i, imm0_15);
__m256i __lasx_xvsrari_w (__m256i, imm0_31);
__m256i __lasx_xvsrarn_b_h (__m256i, __m256i);
__m256i __lasx_xvsrarn_h_w (__m256i, __m256i);
__m256i __lasx_xvsrarni_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvsrarni_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvsrarni_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvsrarni_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvsrarn_w_d (__m256i, __m256i);
__m256i __lasx_xvsrar_w (__m256i, __m256i);
__m256i __lasx_xvsra_w (__m256i, __m256i);
__m256i __lasx_xvsrl_b (__m256i, __m256i);
__m256i __lasx_xvsrl_d (__m256i, __m256i);
__m256i __lasx_xvsrl_h (__m256i, __m256i);
__m256i __lasx_xvsrli_b (__m256i, imm0_7);
__m256i __lasx_xvsrli_d (__m256i, imm0_63);
__m256i __lasx_xvsrli_h (__m256i, imm0_15);

```

```

__m256i __lasx_xvsrli_w (__m256i, imm0_31);
__m256i __lasx_xvsrln_b_h (__m256i, __m256i);
__m256i __lasx_xvsrln_h_w (__m256i, __m256i);
__m256i __lasx_xvsrlni_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvsrlni_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvsrlni_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvsrlni_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvsrln_w_d (__m256i, __m256i);
__m256i __lasx_xvsrlr_b (__m256i, __m256i);
__m256i __lasx_xvsrlr_d (__m256i, __m256i);
__m256i __lasx_xvsrlr_h (__m256i, __m256i);
__m256i __lasx_xvsrlri_b (__m256i, imm0_7);
__m256i __lasx_xvsrlri_d (__m256i, imm0_63);
__m256i __lasx_xvsrlri_h (__m256i, imm0_15);
__m256i __lasx_xvsrlri_w (__m256i, imm0_31);
__m256i __lasx_xvsrlrn_b_h (__m256i, __m256i);
__m256i __lasx_xvsrlrn_h_w (__m256i, __m256i);
__m256i __lasx_xvsrlrni_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvsrlrni_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvsrlrni_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvsrlrni_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvsrlrn_w_d (__m256i, __m256i);
__m256i __lasx_xvsrlr_w (__m256i, __m256i);
__m256i __lasx_xvsrl_w (__m256i, __m256i);
__m256i __lasx_xvssran_b_h (__m256i, __m256i);
__m256i __lasx_xvssran_bu_h (__m256i, __m256i);
__m256i __lasx_xvssran_hu_w (__m256i, __m256i);
__m256i __lasx_xvssran_h_w (__m256i, __m256i);
__m256i __lasx_xvssrani_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvssrani_bu_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvssrani_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrani_du_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrani_hu_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrani_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrani_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssrani_wu_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssran_w_d (__m256i, __m256i);
__m256i __lasx_xvssran_wu_d (__m256i, __m256i);
__m256i __lasx_xvssrarn_b_h (__m256i, __m256i);
__m256i __lasx_xvssrarn_bu_h (__m256i, __m256i);
__m256i __lasx_xvssrarn_hu_w (__m256i, __m256i);
__m256i __lasx_xvssrarn_h_w (__m256i, __m256i);
__m256i __lasx_xvssrarni_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvssrarni_bu_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvssrarni_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrarni_du_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrarni_hu_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrarni_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrarni_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssrarni_wu_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssrarn_w_d (__m256i, __m256i);
__m256i __lasx_xvssrarn_wu_d (__m256i, __m256i);
__m256i __lasx_xvssrln_b_h (__m256i, __m256i);
__m256i __lasx_xvssrln_bu_h (__m256i, __m256i);
__m256i __lasx_xvssrln_hu_w (__m256i, __m256i);
__m256i __lasx_xvssrln_h_w (__m256i, __m256i);
__m256i __lasx_xvssrlni_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvssrlni_bu_h (__m256i, __m256i, imm0_15);

```

```

__m256i __lasx_xvssrlni_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrlni_du_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrlni_hu_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrlni_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrlni_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssrlni_wu_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssrln_w_d (__m256i, __m256i);
__m256i __lasx_xvssrln_wu_d (__m256i, __m256i);
__m256i __lasx_xvssrlrn_b_h (__m256i, __m256i);
__m256i __lasx_xvssrlrn_bu_h (__m256i, __m256i);
__m256i __lasx_xvssrlrn_hu_w (__m256i, __m256i);
__m256i __lasx_xvssrlrn_h_w (__m256i, __m256i);
__m256i __lasx_xvssrlrni_b_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvssrlrni_bu_h (__m256i, __m256i, imm0_15);
__m256i __lasx_xvssrlrni_d_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrlrni_du_q (__m256i, __m256i, imm0_127);
__m256i __lasx_xvssrlrni_hu_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrlrni_h_w (__m256i, __m256i, imm0_31);
__m256i __lasx_xvssrlrni_w_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssrlrni_wu_d (__m256i, __m256i, imm0_63);
__m256i __lasx_xvssrlrn_w_d (__m256i, __m256i);
__m256i __lasx_xvssrlrn_wu_d (__m256i, __m256i);
__m256i __lasx_xvssub_b (__m256i, __m256i);
__m256i __lasx_xvssub_bu (__m256i, __m256i);
__m256i __lasx_xvssub_d (__m256i, __m256i);
__m256i __lasx_xvssub_du (__m256i, __m256i);
__m256i __lasx_xvssub_h (__m256i, __m256i);
__m256i __lasx_xvssub_hu (__m256i, __m256i);
__m256i __lasx_xvssub_w (__m256i, __m256i);
__m256i __lasx_xvssub_wu (__m256i, __m256i);
void __lasx_xvst (__m256i, void *, imm_n2048_2047);
void __lasx_xvstelm_b (__m256i, void *, imm_n128_127, imm0_31);
void __lasx_xvstelm_d (__m256i, void *, imm_n128_127, imm0_3);
void __lasx_xvstelm_h (__m256i, void *, imm_n128_127, imm0_15);
void __lasx_xvstelm_w (__m256i, void *, imm_n128_127, imm0_7);
void __lasx_xvstx (__m256i, void *, long int);
__m256i __lasx_xvsub_b (__m256i, __m256i);
__m256i __lasx_xvsub_d (__m256i, __m256i);
__m256i __lasx_xvsub_h (__m256i, __m256i);
__m256i __lasx_xvsubi_bu (__m256i, imm0_31);
__m256i __lasx_xvsubi_du (__m256i, imm0_31);
__m256i __lasx_xvsubi_hu (__m256i, imm0_31);
__m256i __lasx_xvsubi_wu (__m256i, imm0_31);
__m256i __lasx_xvsub_q (__m256i, __m256i);
__m256i __lasx_xvsub_w (__m256i, __m256i);
__m256i __lasx_xvsubwev_d_w (__m256i, __m256i);
__m256i __lasx_xvsubwev_d_wu (__m256i, __m256i);
__m256i __lasx_xvsubwev_h_b (__m256i, __m256i);
__m256i __lasx_xvsubwev_h_bu (__m256i, __m256i);
__m256i __lasx_xvsubwev_q_d (__m256i, __m256i);
__m256i __lasx_xvsubwev_q_du (__m256i, __m256i);
__m256i __lasx_xvsubwev_w_h (__m256i, __m256i);
__m256i __lasx_xvsubwev_w_hu (__m256i, __m256i);
__m256i __lasx_xvsubwod_d_w (__m256i, __m256i);
__m256i __lasx_xvsubwod_d_wu (__m256i, __m256i);
__m256i __lasx_xvsubwod_h_b (__m256i, __m256i);
__m256i __lasx_xvsubwod_h_bu (__m256i, __m256i);
__m256i __lasx_xvsubwod_q_d (__m256i, __m256i);

```

```

__m256i __lasx_xvsubwod_q_du (__m256i, __m256i);
__m256i __lasx_xvsubwod_w_h (__m256i, __m256i);
__m256i __lasx_xvsubwod_w_hu (__m256i, __m256i);
__m256i __lasx_xvxori_b (__m256i, imm0_255);
__m256i __lasx_xvxor_v (__m256i, __m256i);

```

7.13.14.3 Directly-mapped ASX Division Builtin Functions

These intrinsic functions are available by including `lasxintrin.h` and using `-mfrecipe` and `-mlasx`.

```

__m256d __lasx_xvfrecipe_d (__m256d);
__m256 __lasx_xvfrecipe_s (__m256);
__m256d __lasx_xvfrsqrt_d (__m256d);
__m256 __lasx_xvfrsqrt_s (__m256);

```

7.13.14.4 Directly-mapped SX and ASX Conversion Builtin Functions

For convenience, the `lsxintrin.h` file was imported into `lasxintrin.h` and 18 new interface functions for 128 and 256 vector conversions were added, using the `-mlasx` option.

```

__m256 __lasx_cast_128_s (__m128);
__m256d __lasx_cast_128_d (__m128d);
__m256i __lasx_cast_128 (__m128i);
__m256 __lasx_concat_128_s (__m128, __m128);
__m256d __lasx_concat_128_d (__m128d, __m128d);
__m256i __lasx_concat_128 (__m128i, __m128i);
__m128 __lasx_extract_128_lo_s (__m256);
__m128 __lasx_extract_128_hi_s (__m256);
__m128d __lasx_extract_128_lo_d (__m256d);
__m128d __lasx_extract_128_hi_d (__m256d);
__m128i __lasx_extract_128_lo (__m256i);
__m128i __lasx_extract_128_hi (__m256i);
__m256 __lasx_insert_128_lo_s (__m256, __m128);
__m256 __lasx_insert_128_hi_s (__m256, __m128);
__m256d __lasx_insert_128_lo_d (__m256d, __m128d);
__m256d __lasx_insert_128_hi_d (__m256d, __m128d);
__m256i __lasx_insert_128_lo (__m256i, __m128i);
__m256i __lasx_insert_128_hi (__m256i, __m128i);

```

When gcc does not support interfaces for 128 and 256 conversions, use the following code for equivalent substitution.

```

#ifndef __loongarch_asx_sx_conv

#include <lasxintrin.h>
#include <lsxintrin.h>
__m256 inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_cast_128_s (__m128 src)
{
    __m256 dest;
    asm (" : "=f"(dest) : "0"(src));
    return dest;
}

__m256d inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_cast_128_d (__m128d src)
{

```

```

    __m256d dest;
    asm (" : "=f"(dest) : "0"(src));
    return dest;
}

__m256i inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_cast_128 (__m128i src)
{
    __m256i dest;
    asm (" : "=f"(dest) : "0"(src));
    return dest;
}

__m256 inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_concat_128_s (__m128 src1, __m128 src2)
{
    __m256 dest;
    asm ("xvpermi.q %u0,%u2,0x02\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

__m256d inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_concat_128_d (__m128d src1, __m128d src2)
{
    __m256d dest;
    asm ("xvpermi.q %u0,%u2,0x02\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

__m256i inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_concat_128 (__m128i src1, __m128i src2)
{
    __m256i dest;
    asm ("xvpermi.q %u0,%u2,0x02\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

__m128 inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_extract_128_lo_s (__m256 src)
{
    __m128 dest;
    asm (" : "=f"(dest) : "0"(src));
    return dest;
}

__m128d inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_extract_128_lo_d (__m256d src)
{
    __m128d dest;
    asm (" : "=f"(dest) : "0"(src));
    return dest;
}

```

```

__m128i inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_extract_128_lo (__m256i src)
{
    __m128i dest;
    asm (" : "=f"(dest) : "0"(src));
    return dest;
}

__m128 inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_extract_128_hi_s (__m256 src)
{
    __m128 dest;
    asm ("xvpermi.d %u0,%u1,0xe\n"
        : "=f"(dest)
        : "f"(src));
    return dest;
}

__m128d inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_extract_128_hi_d (__m256d src)
{
    __m128d dest;
    asm ("xvpermi.d %u0,%u1,0xe\n"
        : "=f"(dest)
        : "f"(src));
    return dest;
}

__m128i inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_extract_128_hi (__m256i src)
{
    __m128i dest;
    asm ("xvpermi.d %u0,%u1,0xe\n"
        : "=f"(dest)
        : "f"(src));
    return dest;
}

__m256 inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_insert_128_lo_s (__m256 src1, __m128 src2)
{
    __m256 dest;
    asm ("xvpermi.q %u0,%u2,0x30\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

__m256d inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_insert_128_lo_d (__m256d a, __m128d b)
{
    __m256d dest;
    asm ("xvpermi.q %u0,%u2,0x30\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

```

```

__m256i inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_insert_128_lo (__m256i src1, __m128i src2)
{
    __m256i dest;
    asm ("xvpermi.q %u0,%u2,0x30\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

__m256 inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_insert_128_hi_s (__m256 src1, __m128 src2)
{
    __m256 dest;
    asm ("xvpermi.q %u0,%u2,0x02\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

__m256d inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_insert_128_hi_d (__m256d src1, __m128d src2)
{
    __m256d dest;
    asm ("xvpermi.q %u0,%u2,0x02\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}

__m256i inline __attribute__((__gnu_inline__, __always_inline__, __artificial__))
__lasx_insert_128_hi (__m256i src1, __m128i src2)
{
    __m256i dest;
    asm ("xvpermi.q %u0,%u2,0x02\n"
        : "=f"(dest)
        : "0"(src1), "f"(src2));
    return dest;
}
#endif

```

7.13.15 MIPS DSP Built-in Functions

The MIPS DSP Application-Specific Extension (ASE) includes new instructions that are designed to improve the performance of DSP and media applications. It provides instructions that operate on packed 8-bit/16-bit integer data, Q7, Q15 and Q31 fractional data.

GCC supports MIPS DSP operations using both the generic vector extensions (see Section 7.8 [Vector Extensions], page 816) and a collection of MIPS-specific built-in functions. Both kinds of support are enabled by the `-mdsp` command-line option.

Revision 2 of the ASE was introduced in the second half of 2006. This revision adds extra instructions to the original ASE, but is otherwise backwards-compatible with it. You can select revision 2 using the command-line option `-mdspr2`; this option implies `-mdsp`.

The SCOUNT and POS bits of the DSP control register are global. The WRDSP, EXTPDP, EXTPDPV and MTHLIP instructions modify the SCOUNT and POS bits. During optimization, the compiler does not delete these instructions and it does not delete calls to functions containing these instructions.

At present, GCC only provides support for operations on 32-bit vectors. The vector type associated with 8-bit integer data is usually called `v4i8`, the vector type associated with Q7 is usually called `v4q7`, the vector type associated with 16-bit integer data is usually called `v2i16`, and the vector type associated with Q15 is usually called `v2q15`. They can be defined in C as follows:

```
typedef signed char v4i8 __attribute__((vector_size(4)));
typedef signed char v4q7 __attribute__((vector_size(4)));
typedef short v2i16 __attribute__((vector_size(4)));
typedef short v2q15 __attribute__((vector_size(4)));
```

`v4i8`, `v4q7`, `v2i16` and `v2q15` values are initialized in the same way as aggregates. For example:

```
v4i8 a = {1, 2, 3, 4};
v4i8 b;
b = (v4i8) {5, 6, 7, 8};

v2q15 c = {0x0fcb, 0x3a75};
v2q15 d;
d = (v2q15) {0.1234 * 0x1.0p15, 0.4567 * 0x1.0p15};
```

Note: The CPU's endianness determines the order in which values are packed. On little-endian targets, the first value is the least significant and the last value is the most significant. The opposite order applies to big-endian targets. For example, the code above sets the lowest byte of `a` to 1 on little-endian targets and 4 on big-endian targets.

Note: Q7, Q15 and Q31 values must be initialized with their integer representation. As shown in this example, the integer representation of a Q7 value can be obtained by multiplying the fractional value by `0x1.0p7`. The equivalent for Q15 values is to multiply by `0x1.0p15`. The equivalent for Q31 values is to multiply by `0x1.0p31`.

The table below lists the `v4i8` and `v2q15` operations for which hardware support exists. `a` and `b` are `v4i8` values, and `c` and `d` are `v2q15` values.

C code	MIPS instruction
<code>a + b</code>	<code>addu.qb</code>
<code>c + d</code>	<code>addq.ph</code>
<code>a - b</code>	<code>subu.qb</code>
<code>c - d</code>	<code>subq.ph</code>

The table below lists the `v2i16` operation for which hardware support exists for the DSP ASE REV 2. `e` and `f` are `v2i16` values.

C code	MIPS instruction
<code>e * f</code>	<code>mul.ph</code>

It is easier to describe the DSP built-in functions if we first define the following types:

```
typedef int q31;
typedef int i32;
typedef unsigned int ui32;
typedef long long a64;
```

`q31` and `i32` are actually the same as `int`, but we use `q31` to indicate a Q31 fractional value and `i32` to indicate a 32-bit integer value. Similarly, `a64` is the same as `long long`, but we use `a64` to indicate values that are placed in one of the four DSP accumulators (`$ac0`, `$ac1`, `$ac2` or `$ac3`).

Also, some built-in functions prefer or require immediate numbers as parameters, because the corresponding DSP instructions accept both immediate numbers and register operands, or accept immediate numbers only. The immediate parameters are listed as follows.

```
imm0_3: 0 to 3.
imm0_7: 0 to 7.
imm0_15: 0 to 15.
imm0_31: 0 to 31.
imm0_63: 0 to 63.
imm0_255: 0 to 255.
imm_n32_31: -32 to 31.
imm_n512_511: -512 to 511.
```

The following built-in functions map directly to a particular MIPS DSP instruction. Please refer to the architecture specification for details on what each instruction does.

```
v2q15 __builtin_mips_addq_ph (v2q15, v2q15);
v2q15 __builtin_mips_addq_s_ph (v2q15, v2q15);
q31 __builtin_mips_addq_s_w (q31, q31);
v4i8 __builtin_mips_addu_qb (v4i8, v4i8);
v4i8 __builtin_mips_addu_s_qb (v4i8, v4i8);
v2q15 __builtin_mips_subq_ph (v2q15, v2q15);
v2q15 __builtin_mips_subq_s_ph (v2q15, v2q15);
q31 __builtin_mips_subq_s_w (q31, q31);
v4i8 __builtin_mips_subu_qb (v4i8, v4i8);
v4i8 __builtin_mips_subu_s_qb (v4i8, v4i8);
i32 __builtin_mips_addsc (i32, i32);
i32 __builtin_mips_addwc (i32, i32);
i32 __builtin_mips_modsub (i32, i32);
i32 __builtin_mips_raddu_w_qb (v4i8);
v2q15 __builtin_mips_absq_s_ph (v2q15);
q31 __builtin_mips_absq_s_w (q31);
v4i8 __builtin_mips_precrq_qb_ph (v2q15, v2q15);
v2q15 __builtin_mips_precrq_ph_w (q31, q31);
v2q15 __builtin_mips_precrq_rs_ph_w (q31, q31);
v4i8 __builtin_mips_precrq_s_qb_ph (v2q15, v2q15);
q31 __builtin_mips_preceq_w_phl (v2q15);
q31 __builtin_mips_preceq_w_phr (v2q15);
v2q15 __builtin_mips_precequ_ph_qbl (v4i8);
v2q15 __builtin_mips_precequ_ph_qbr (v4i8);
v2q15 __builtin_mips_precequ_ph_qbla (v4i8);
v2q15 __builtin_mips_precequ_ph_qbra (v4i8);
v2q15 __builtin_mips_preceu_ph_qbl (v4i8);
v2q15 __builtin_mips_preceu_ph_qbr (v4i8);
v2q15 __builtin_mips_preceu_ph_qbla (v4i8);
v2q15 __builtin_mips_preceu_ph_qbra (v4i8);
v4i8 __builtin_mips_shll_qb (v4i8, imm0_7);
v4i8 __builtin_mips_shll_qb (v4i8, i32);
v2q15 __builtin_mips_shll_ph (v2q15, imm0_15);
v2q15 __builtin_mips_shll_ph (v2q15, i32);
v2q15 __builtin_mips_shll_s_ph (v2q15, imm0_15);
v2q15 __builtin_mips_shll_s_ph (v2q15, i32);
q31 __builtin_mips_shll_s_w (q31, imm0_31);
q31 __builtin_mips_shll_s_w (q31, i32);
```

```

v4i8 __builtin_mips_shrl_qb (v4i8, imm0_7);
v4i8 __builtin_mips_shrl_qb (v4i8, i32);
v2q15 __builtin_mips_shra_ph (v2q15, imm0_15);
v2q15 __builtin_mips_shra_ph (v2q15, i32);
v2q15 __builtin_mips_shra_r_ph (v2q15, imm0_15);
v2q15 __builtin_mips_shra_r_ph (v2q15, i32);
q31 __builtin_mips_shra_r_w (q31, imm0_31);
q31 __builtin_mips_shra_r_w (q31, i32);
v2q15 __builtin_mips_muleu_s_ph_qbl (v4i8, v2q15);
v2q15 __builtin_mips_muleu_s_ph_qbr (v4i8, v2q15);
v2q15 __builtin_mips_mulq_rs_ph (v2q15, v2q15);
q31 __builtin_mips_muleq_s_w_phl (v2q15, v2q15);
q31 __builtin_mips_muleq_s_w_phr (v2q15, v2q15);
a64 __builtin_mips_dpau_h_qbl (a64, v4i8, v4i8);
a64 __builtin_mips_dpau_h_qbr (a64, v4i8, v4i8);
a64 __builtin_mips_dpsu_h_qbl (a64, v4i8, v4i8);
a64 __builtin_mips_dpsu_h_qbr (a64, v4i8, v4i8);
a64 __builtin_mips_dpaq_s_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpaq_sa_l_w (a64, q31, q31);
a64 __builtin_mips_dpsq_s_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpsq_sa_l_w (a64, q31, q31);
a64 __builtin_mips_mulsaq_s_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_maq_s_w_phl (a64, v2q15, v2q15);
a64 __builtin_mips_maq_s_w_phr (a64, v2q15, v2q15);
a64 __builtin_mips_maq_sa_w_phl (a64, v2q15, v2q15);
a64 __builtin_mips_maq_sa_w_phr (a64, v2q15, v2q15);
i32 __builtin_mips_bitrev (i32);
i32 __builtin_mips_insv (i32, i32);
v4i8 __builtin_mips_repl_qb (imm0_255);
v4i8 __builtin_mips_repl_qb (i32);
v2q15 __builtin_mips_repl_ph (imm_n512_511);
v2q15 __builtin_mips_repl_ph (i32);
void __builtin_mips_cmpu_eq_qb (v4i8, v4i8);
void __builtin_mips_cmpu_lt_qb (v4i8, v4i8);
void __builtin_mips_cmpu_le_qb (v4i8, v4i8);
i32 __builtin_mips_cmpgu_eq_qb (v4i8, v4i8);
i32 __builtin_mips_cmpgu_lt_qb (v4i8, v4i8);
i32 __builtin_mips_cmpgu_le_qb (v4i8, v4i8);
void __builtin_mips_cmp_eq_ph (v2q15, v2q15);
void __builtin_mips_cmp_lt_ph (v2q15, v2q15);
void __builtin_mips_cmp_le_ph (v2q15, v2q15);
v4i8 __builtin_mips_pick_qb (v4i8, v4i8);
v2q15 __builtin_mips_pick_ph (v2q15, v2q15);
v2q15 __builtin_mips_packrl_ph (v2q15, v2q15);
i32 __builtin_mips_extr_w (a64, imm0_31);
i32 __builtin_mips_extr_w (a64, i32);
i32 __builtin_mips_extr_r_w (a64, imm0_31);
i32 __builtin_mips_extr_s_h (a64, i32);
i32 __builtin_mips_extr_rs_w (a64, imm0_31);
i32 __builtin_mips_extr_rs_w (a64, i32);
i32 __builtin_mips_extr_s_h (a64, imm0_31);
i32 __builtin_mips_extr_r_w (a64, i32);
i32 __builtin_mips_extp (a64, imm0_31);
i32 __builtin_mips_extp (a64, i32);
i32 __builtin_mips_extpdp (a64, imm0_31);
i32 __builtin_mips_extpdp (a64, i32);
a64 __builtin_mips_shilo (a64, imm_n32_31);
a64 __builtin_mips_shilo (a64, i32);

```

```

a64 __builtin_mips_mthlip (a64, i32);
void __builtin_mips_wrdsb (i32, imm0_63);
i32 __builtin_mips_rddsb (imm0_63);
i32 __builtin_mips_lbus (void *, i32);
i32 __builtin_mips_lhx (void *, i32);
i32 __builtin_mips_lwx (void *, i32);
a64 __builtin_mips_ldx (void *, i32); /* MIPS64 only */
i32 __builtin_mips_bposge32 (void);
a64 __builtin_mips_madd (a64, i32, i32);
a64 __builtin_mips_maddu (a64, ui32, ui32);
a64 __builtin_mips_msub (a64, i32, i32);
a64 __builtin_mips_msubu (a64, ui32, ui32);
a64 __builtin_mips_mult (i32, i32);
a64 __builtin_mips_multu (ui32, ui32);

```

The following built-in functions map directly to a particular MIPS DSP REV 2 instruction. Please refer to the architecture specification for details on what each instruction does.

```

v4q7 __builtin_mips_absq_s_qb (v4q7);
v2i16 __builtin_mips_addu_ph (v2i16, v2i16);
v2i16 __builtin_mips_addu_s_ph (v2i16, v2i16);
v4i8 __builtin_mips_adduh_qb (v4i8, v4i8);
v4i8 __builtin_mips_adduh_r_qb (v4i8, v4i8);
i32 __builtin_mips_append (i32, i32, imm0_31);
i32 __builtin_mips_balign (i32, i32, imm0_3);
i32 __builtin_mips_cmpgdu_eq_qb (v4i8, v4i8);
i32 __builtin_mips_cmpgdu_lt_qb (v4i8, v4i8);
i32 __builtin_mips_cmpgdu_le_qb (v4i8, v4i8);
a64 __builtin_mips_dpa_w_ph (a64, v2i16, v2i16);
a64 __builtin_mips_dps_w_ph (a64, v2i16, v2i16);
v2i16 __builtin_mips_mul_ph (v2i16, v2i16);
v2i16 __builtin_mips_mul_s_ph (v2i16, v2i16);
q31 __builtin_mips_mulq_rs_w (q31, q31);
v2q15 __builtin_mips_mulq_s_ph (v2q15, v2q15);
q31 __builtin_mips_mulq_s_w (q31, q31);
a64 __builtin_mips_mulsa_w_ph (a64, v2i16, v2i16);
v4i8 __builtin_mips_precr_qb_ph (v2i16, v2i16);
v2i16 __builtin_mips_precr_sra_ph_w (i32, i32, imm0_31);
v2i16 __builtin_mips_precr_sra_r_ph_w (i32, i32, imm0_31);
i32 __builtin_mips_prepend (i32, i32, imm0_31);
v4i8 __builtin_mips_shra_qb (v4i8, imm0_7);
v4i8 __builtin_mips_shra_r_qb (v4i8, imm0_7);
v4i8 __builtin_mips_shra_qb (v4i8, i32);
v4i8 __builtin_mips_shra_r_qb (v4i8, i32);
v2i16 __builtin_mips_shrl_ph (v2i16, imm0_15);
v2i16 __builtin_mips_shrl_ph (v2i16, i32);
v2i16 __builtin_mips_subu_ph (v2i16, v2i16);
v2i16 __builtin_mips_subu_s_ph (v2i16, v2i16);
v4i8 __builtin_mips_subuh_qb (v4i8, v4i8);
v4i8 __builtin_mips_subuh_r_qb (v4i8, v4i8);
v2q15 __builtin_mips_addqh_ph (v2q15, v2q15);
v2q15 __builtin_mips_addqh_r_ph (v2q15, v2q15);
q31 __builtin_mips_addqh_w (q31, q31);
q31 __builtin_mips_addqh_r_w (q31, q31);
v2q15 __builtin_mips_subqh_ph (v2q15, v2q15);
v2q15 __builtin_mips_subqh_r_ph (v2q15, v2q15);
q31 __builtin_mips_subqh_w (q31, q31);
q31 __builtin_mips_subqh_r_w (q31, q31);
a64 __builtin_mips_dpax_w_ph (a64, v2i16, v2i16);

```

```

a64 __builtin_mips_dpsx_w_ph (a64, v2i16, v2i16);
a64 __builtin_mips_dpaqx_s_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpaqx_sa_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpsqx_s_w_ph (a64, v2q15, v2q15);
a64 __builtin_mips_dpsqx_sa_w_ph (a64, v2q15, v2q15);

```

7.13.16 MIPS Paired-Single Support

The MIPS64 architecture includes a number of instructions that operate on pairs of single-precision floating-point values. Each pair is packed into a 64-bit floating-point register, with one element being designated the “upper half” and the other being designated the “lower half”.

GCC supports paired-single operations using both the generic vector extensions (see Section 7.8 [Vector Extensions], page 816) and a collection of MIPS-specific built-in functions. Both kinds of support are enabled by the `-mpaired-single` command-line option.

The vector type associated with paired-single values is usually called `v2sf`. It can be defined in C as follows:

```

typedef float v2sf __attribute__((vector_size (8)));
v2sf values are initialized in the same way as aggregates. For example:
v2sf a = {1.5, 9.1};
v2sf b;
float e, f;
b = (v2sf) {e, f};

```

Note: The CPU’s endianness determines which value is stored in the upper half of a register and which value is stored in the lower half. On little-endian targets, the first value is the lower one and the second value is the upper one. The opposite order applies to big-endian targets. For example, the code above sets the lower half of `a` to 1.5 on little-endian targets and 9.1 on big-endian targets.

7.13.17 MIPS Loongson Built-in Functions

GCC provides intrinsics to access the SIMD instructions provided by the ST Microelectronics Loongson-2E and -2F processors. These intrinsics, available after inclusion of the `loongson.h` header file, operate on the following 64-bit vector types:

- `uint8x8_t`, a vector of eight unsigned 8-bit integers;
- `uint16x4_t`, a vector of four unsigned 16-bit integers;
- `uint32x2_t`, a vector of two unsigned 32-bit integers;
- `int8x8_t`, a vector of eight signed 8-bit integers;
- `int16x4_t`, a vector of four signed 16-bit integers;
- `int32x2_t`, a vector of two signed 32-bit integers.

The intrinsics provided are listed below; each is named after the machine instruction to which it corresponds, with suffixes added as appropriate to distinguish intrinsics that expand to the same machine instruction yet have different argument types. Refer to the architecture documentation for a description of the functionality of each instruction.

```

int16x4_t packsswh (int32x2_t s, int32x2_t t);
int8x8_t packsshb (int16x4_t s, int16x4_t t);
uint8x8_t packushb (uint16x4_t s, uint16x4_t t);
uint32x2_t paddw_u (uint32x2_t s, uint32x2_t t);

```

```

uint16x4_t paddh_u (uint16x4_t s, uint16x4_t t);
uint8x8_t paddb_u (uint8x8_t s, uint8x8_t t);
int32x2_t paddw_s (int32x2_t s, int32x2_t t);
int16x4_t paddh_s (int16x4_t s, int16x4_t t);
int8x8_t paddb_s (int8x8_t s, int8x8_t t);
uint64_t paddd_u (uint64_t s, uint64_t t);
int64_t paddd_s (int64_t s, int64_t t);
int16x4_t paddsh (int16x4_t s, int16x4_t t);
int8x8_t paddsb (int8x8_t s, int8x8_t t);
uint16x4_t paddush (uint16x4_t s, uint16x4_t t);
uint8x8_t paddusb (uint8x8_t s, uint8x8_t t);
uint64_t pandn_ud (uint64_t s, uint64_t t);
uint32x2_t pandn_uw (uint32x2_t s, uint32x2_t t);
uint16x4_t pandn_uh (uint16x4_t s, uint16x4_t t);
uint8x8_t pandn_ub (uint8x8_t s, uint8x8_t t);
int64_t pandn_sd (int64_t s, int64_t t);
int32x2_t pandn_sw (int32x2_t s, int32x2_t t);
int16x4_t pandn_sh (int16x4_t s, int16x4_t t);
int8x8_t pandn_sb (int8x8_t s, int8x8_t t);
uint16x4_t pavgh (uint16x4_t s, uint16x4_t t);
uint8x8_t pavgb (uint8x8_t s, uint8x8_t t);
uint32x2_t pcmpeqw_u (uint32x2_t s, uint32x2_t t);
uint16x4_t pcmpeqh_u (uint16x4_t s, uint16x4_t t);
uint8x8_t pcmpeqb_u (uint8x8_t s, uint8x8_t t);
int32x2_t pcmpeqw_s (int32x2_t s, int32x2_t t);
int16x4_t pcmpeqh_s (int16x4_t s, int16x4_t t);
int8x8_t pcmpeqb_s (int8x8_t s, int8x8_t t);
uint32x2_t pcmpgtw_u (uint32x2_t s, uint32x2_t t);
uint16x4_t pcmpgth_u (uint16x4_t s, uint16x4_t t);
uint8x8_t pcmpgtb_u (uint8x8_t s, uint8x8_t t);
int32x2_t pcmpgtw_s (int32x2_t s, int32x2_t t);
int16x4_t pcmpgth_s (int16x4_t s, int16x4_t t);
int8x8_t pcmpgtb_s (int8x8_t s, int8x8_t t);
uint16x4_t pextrh_u (uint16x4_t s, int field);
int16x4_t pextrh_s (int16x4_t s, int field);
uint16x4_t pinsrh_0_u (uint16x4_t s, uint16x4_t t);
uint16x4_t pinsrh_1_u (uint16x4_t s, uint16x4_t t);
uint16x4_t pinsrh_2_u (uint16x4_t s, uint16x4_t t);
uint16x4_t pinsrh_3_u (uint16x4_t s, uint16x4_t t);
int16x4_t pinsrh_0_s (int16x4_t s, int16x4_t t);
int16x4_t pinsrh_1_s (int16x4_t s, int16x4_t t);
int16x4_t pinsrh_2_s (int16x4_t s, int16x4_t t);
int16x4_t pinsrh_3_s (int16x4_t s, int16x4_t t);
int32x2_t pmaddhw (int16x4_t s, int16x4_t t);
int16x4_t pmaxsh (int16x4_t s, int16x4_t t);
uint8x8_t pmaxub (uint8x8_t s, uint8x8_t t);
int16x4_t pminsh (int16x4_t s, int16x4_t t);
uint8x8_t pminub (uint8x8_t s, uint8x8_t t);
uint8x8_t pmovmskb_u (uint8x8_t s);
int8x8_t pmovmskb_s (int8x8_t s);
uint16x4_t pmulhuh (uint16x4_t s, uint16x4_t t);
int16x4_t pmulhh (int16x4_t s, int16x4_t t);
int16x4_t pmullh (int16x4_t s, int16x4_t t);
int64_t pmuluw (uint32x2_t s, uint32x2_t t);
uint8x8_t pasubub (uint8x8_t s, uint8x8_t t);
uint16x4_t biadd (uint8x8_t s);
uint16x4_t psadbh (uint8x8_t s, uint8x8_t t);
uint16x4_t pshufh_u (uint16x4_t dest, uint16x4_t s, uint8_t order);

```

```

int16x4_t pshufh_s (int16x4_t dest, int16x4_t s, uint8_t order);
uint16x4_t psllh_u (uint16x4_t s, uint8_t amount);
int16x4_t psllh_s (int16x4_t s, uint8_t amount);
uint32x2_t psllw_u (uint32x2_t s, uint8_t amount);
int32x2_t psllw_s (int32x2_t s, uint8_t amount);
uint16x4_t psrlh_u (uint16x4_t s, uint8_t amount);
int16x4_t psrlh_s (int16x4_t s, uint8_t amount);
uint32x2_t psrlw_u (uint32x2_t s, uint8_t amount);
int32x2_t psrlw_s (int32x2_t s, uint8_t amount);
uint16x4_t psrah_u (uint16x4_t s, uint8_t amount);
int16x4_t psrah_s (int16x4_t s, uint8_t amount);
uint32x2_t psraw_u (uint32x2_t s, uint8_t amount);
int32x2_t psraw_s (int32x2_t s, uint8_t amount);
uint32x2_t psubw_u (uint32x2_t s, uint32x2_t t);
uint16x4_t psubh_u (uint16x4_t s, uint16x4_t t);
uint8x8_t psubb_u (uint8x8_t s, uint8x8_t t);
int32x2_t psubw_s (int32x2_t s, int32x2_t t);
int16x4_t psubh_s (int16x4_t s, int16x4_t t);
int8x8_t psubb_s (int8x8_t s, int8x8_t t);
uint64_t psubd_u (uint64_t s, uint64_t t);
int64_t psubd_s (int64_t s, int64_t t);
int16x4_t psubsh (int16x4_t s, int16x4_t t);
int8x8_t psubsb (int8x8_t s, int8x8_t t);
uint16x4_t psubush (uint16x4_t s, uint16x4_t t);
uint8x8_t psubusb (uint8x8_t s, uint8x8_t t);
uint32x2_t punpckhwd_u (uint32x2_t s, uint32x2_t t);
uint16x4_t punpckhhw_u (uint16x4_t s, uint16x4_t t);
uint8x8_t punpckhbw_u (uint8x8_t s, uint8x8_t t);
int32x2_t punpckhwd_s (int32x2_t s, int32x2_t t);
int16x4_t punpckhhw_s (int16x4_t s, int16x4_t t);
int8x8_t punpckhbw_s (int8x8_t s, int8x8_t t);
uint32x2_t punpcklwd_u (uint32x2_t s, uint32x2_t t);
uint16x4_t punpcklhw_u (uint16x4_t s, uint16x4_t t);
uint8x8_t punpcklbw_u (uint8x8_t s, uint8x8_t t);
int32x2_t punpcklwd_s (int32x2_t s, int32x2_t t);
int16x4_t punpcklhw_s (int16x4_t s, int16x4_t t);
int8x8_t punpcklbw_s (int8x8_t s, int8x8_t t);

```

7.13.17.1 Paired-Single Arithmetic

The table below lists the v2sf operations for which hardware support exists. a, b and c are v2sf values and x is an integral value.

C code	MIPS instruction
a + b	add.ps
a - b	sub.ps
-a	neg.ps
a * b	mul.ps
a * b + c	madd.ps
a * b - c	msub.ps
-(a * b + c)	nmadd.ps
-(a * b - c)	nmsub.ps
x ? a : b	movn.ps/movz.ps

Note that the multiply-accumulate instructions can be disabled using the command-line option `-mno-fused-madd`.

7.13.17.2 Paired-Single Built-in Functions

The following paired-single functions map directly to a particular MIPS instruction. Please refer to the architecture specification for details on what each instruction does.

```
v2sf __builtin_mips_pll_ps (v2sf, v2sf)
    Pair lower lower (pll.ps).

v2sf __builtin_mips_pul_ps (v2sf, v2sf)
    Pair upper lower (pul.ps).

v2sf __builtin_mips_plu_ps (v2sf, v2sf)
    Pair lower upper (plu.ps).

v2sf __builtin_mips_puu_ps (v2sf, v2sf)
    Pair upper upper (puu.ps).

v2sf __builtin_mips_cvt_ps_s (float, float)
    Convert pair to paired single (cvt.ps.s).

float __builtin_mips_cvt_s_pl (v2sf)
    Convert pair lower to single (cvt.s.pl).

float __builtin_mips_cvt_s_pu (v2sf)
    Convert pair upper to single (cvt.s.pu).

v2sf __builtin_mips_abs_ps (v2sf)
    Absolute value (abs.ps).

v2sf __builtin_mips_alnv_ps (v2sf, v2sf, int)
    Align variable (alnv.ps).
```

Note: The value of the third parameter must be 0 or 4 modulo 8, otherwise the result is unpredictable. Please read the instruction description for details.

The following multi-instruction functions are also available. In each case, *cond* can be any of the 16 floating-point conditions: *f*, *un*, *eq*, *ueq*, *olt*, *ult*, *ole*, *ule*, *sf*, *nge*, *seq*, *ngl*, *lt*, *nge*, *le* or *ngt*.

```
v2sf __builtin_mips_movt_c_cond_ps (v2sf a, v2sf b, v2sf c, v2sf d)
v2sf __builtin_mips_movf_c_cond_ps (v2sf a, v2sf b, v2sf c, v2sf d)
    Conditional move based on floating-point comparison (c.cond.ps,
    movt.ps/movf.ps).
```

The *movt* functions return the value *x* computed by:

```
c.cond.ps cc,a,b
mov.ps x,c
movt.ps x,d,cc
```

The *movf* functions are similar but use *movf.ps* instead of *movt.ps*.

```
int __builtin_mips_upper_c_cond_ps (v2sf a, v2sf b)
int __builtin_mips_lower_c_cond_ps (v2sf a, v2sf b)
    Comparison of two paired-single values (c.cond.ps, bc1t/bc1f).
```

These functions compare *a* and *b* using *c.cond.ps* and return either the upper or lower half of the result. For example:

```
v2sf a, b;
```

```

    if (__builtin_mips_upper_c_eq_ps (a, b))
        upper_halves_are_equal ();
    else
        upper_halves_are_unequal ();

    if (__builtin_mips_lower_c_eq_ps (a, b))
        lower_halves_are_equal ();
    else
        lower_halves_are_unequal ();

```

7.13.17.3 MIPS-3D Built-in Functions

The MIPS-3D Application-Specific Extension (ASE) includes additional paired-single instructions that are designed to improve the performance of 3D graphics operations. Support for these instructions is controlled by the `-mips3d` command-line option.

The functions listed below map directly to a particular MIPS-3D instruction. Please refer to the architecture specification for more details on what each instruction does.

```

v2sf __builtin_mips_addr_ps (v2sf, v2sf)
    Reduction add (addr.ps).

v2sf __builtin_mips_mulr_ps (v2sf, v2sf)
    Reduction multiply (mulr.ps).

v2sf __builtin_mips_cvt_pw_ps (v2sf)
    Convert paired single to paired word (cvt.pw.ps).

v2sf __builtin_mips_cvt_ps_pw (v2sf)
    Convert paired word to paired single (cvt.ps.pw).

float __builtin_mips_recip1_s (float)
double __builtin_mips_recip1_d (double)
v2sf __builtin_mips_recip1_ps (v2sf)
    Reduced-precision reciprocal (sequence step 1) (recip1.fmt).

float __builtin_mips_recip2_s (float, float)
double __builtin_mips_recip2_d (double, double)
v2sf __builtin_mips_recip2_ps (v2sf, v2sf)
    Reduced-precision reciprocal (sequence step 2) (recip2.fmt).

float __builtin_mips_rsqr1_s (float)
double __builtin_mips_rsqr1_d (double)
v2sf __builtin_mips_rsqr1_ps (v2sf)
    Reduced-precision reciprocal square root (sequence step 1) (rsqr1.fmt).

float __builtin_mips_rsqr2_s (float, float)
double __builtin_mips_rsqr2_d (double, double)
v2sf __builtin_mips_rsqr2_ps (v2sf, v2sf)
    Reduced-precision reciprocal square root (sequence step 2) (rsqr2.fmt).

```

The following multi-instruction functions are also available. In each case, *cond* can be any of the 16 floating-point conditions: `f`, `un`, `eq`, `ueq`, `olt`, `ult`, `ole`, `ule`, `sf`, `ngle`, `seq`, `ngl`, `lt`, `nge`, `le` or `ngt`.

```
int __builtin_mips_cabs_cond_s (float a, float b)
```

```
int __builtin_mips_cabs_cond_d (double a, double b)
```

Absolute comparison of two scalar values (*cabs.cond.fmt*, *bc1t/bc1f*).

These functions compare *a* and *b* using *cabs.cond.s* or *cabs.cond.d* and return the result as a boolean value. For example:

```
float a, b;
if (__builtin_mips_cabs_eq_s (a, b))
    true ();
else
    false ();
```

```
int __builtin_mips_upper_cabs_cond_ps (v2sf a, v2sf b)
```

```
int __builtin_mips_lower_cabs_cond_ps (v2sf a, v2sf b)
```

Absolute comparison of two paired-single values (*cabs.cond.ps*, *bc1t/bc1f*).

These functions compare *a* and *b* using *cabs.cond.ps* and return either the upper or lower half of the result. For example:

```
v2sf a, b;
if (__builtin_mips_upper_cabs_eq_ps (a, b))
    upper_halves_are_equal ();
else
    upper_halves_are_unequal ();

if (__builtin_mips_lower_cabs_eq_ps (a, b))
    lower_halves_are_equal ();
else
    lower_halves_are_unequal ();
```

```
v2sf __builtin_mips_movt_cabs_cond_ps (v2sf a, v2sf b, v2sf c, v2sf d)
```

```
v2sf __builtin_mips_movf_cabs_cond_ps (v2sf a, v2sf b, v2sf c, v2sf d)
```

Conditional move based on absolute comparison (*cabs.cond.ps*, *movt.ps/movf.ps*).

The *movt* functions return the value *x* computed by:

```
cabs.cond.ps cc,a,b
mov.ps x,c
movt.ps x,d,cc
```

The *movf* functions are similar but use *movf.ps* instead of *movt.ps*.

```
int __builtin_mips_any_c_cond_ps (v2sf a, v2sf b)
```

```
int __builtin_mips_all_c_cond_ps (v2sf a, v2sf b)
```

```
int __builtin_mips_any_cabs_cond_ps (v2sf a, v2sf b)
```

```
int __builtin_mips_all_cabs_cond_ps (v2sf a, v2sf b)
```

Comparison of two paired-single values (*c.cond.ps/cabs.cond.ps*, *bc1any2t/bc1any2f*).

These functions compare *a* and *b* using *c.cond.ps* or *cabs.cond.ps*. The *any* forms return *true* if either result is *true* and the *all* forms return *true* if both results are *true*. For example:

```
v2sf a, b;
if (__builtin_mips_any_c_eq_ps (a, b))
    one_is_true ();
else
    both_are_false ();
```

```

        if (__builtin_mips_all_c_eq_ps (a, b))
            both_are_true ();
        else
            one_is_false ();

int __builtin_mips_any_c_cond_4s (v2sf a, v2sf b, v2sf c, v2sf d)
int __builtin_mips_all_c_cond_4s (v2sf a, v2sf b, v2sf c, v2sf d)
int __builtin_mips_any_cabs_cond_4s (v2sf a, v2sf b, v2sf c, v2sf d)
int __builtin_mips_all_cabs_cond_4s (v2sf a, v2sf b, v2sf c, v2sf d)

```

Comparison of four paired-single values (`c.cond.ps/cabs.cond.ps`, `bc1any4t/bc1any4f`).

These functions use `c.cond.ps` or `cabs.cond.ps` to compare *a* with *b* and to compare *c* with *d*. The `any` forms return `true` if any of the four results are `true` and the `all` forms return `true` if all four results are `true`. For example:

```

v2sf a, b, c, d;
if (__builtin_mips_any_c_eq_4s (a, b, c, d))
    some_are_true ();
else
    all_are_false ();

if (__builtin_mips_all_c_eq_4s (a, b, c, d))
    all_are_true ();
else
    some_are_false ();

```

7.13.18 MIPS SIMD Architecture (MSA) Support

GCC provides intrinsics to access the SIMD instructions provided by the MSA MIPS SIMD Architecture. The interface is made available by including `<msa.h>` and using `-mmsa -mhard-float -mfp64 -mnan=2008`. For each `__builtin_msa_*`, there is a shortened name of the intrinsic, `__msa_*`.

MSA implements 128-bit wide vector registers, operating on 8-, 16-, 32- and 64-bit integer, 16- and 32-bit fixed-point, or 32- and 64-bit floating point data elements. The following vectors typedefs are included in `msa.h`:

- `v16i8`, a vector of sixteen signed 8-bit integers;
- `v16u8`, a vector of sixteen unsigned 8-bit integers;
- `v8i16`, a vector of eight signed 16-bit integers;
- `v8u16`, a vector of eight unsigned 16-bit integers;
- `v4i32`, a vector of four signed 32-bit integers;
- `v4u32`, a vector of four unsigned 32-bit integers;
- `v2i64`, a vector of two signed 64-bit integers;
- `v2u64`, a vector of two unsigned 64-bit integers;
- `v4f32`, a vector of four 32-bit floats;
- `v2f64`, a vector of two 64-bit doubles.

Instructions and corresponding built-ins may have additional restrictions and/or input/output values manipulated:

- `imm0_1`, an integer literal in range 0 to 1;

- `imm0_3`, an integer literal in range 0 to 3;
- `imm0_7`, an integer literal in range 0 to 7;
- `imm0_15`, an integer literal in range 0 to 15;
- `imm0_31`, an integer literal in range 0 to 31;
- `imm0_63`, an integer literal in range 0 to 63;
- `imm0_255`, an integer literal in range 0 to 255;
- `imm_n16_15`, an integer literal in range -16 to 15;
- `imm_n512_511`, an integer literal in range -512 to 511;
- `imm_n1024_1022`, an integer literal in range -512 to 511 left shifted by 1 bit, i.e., -1024, -1022, ..., 1020, 1022;
- `imm_n2048_2044`, an integer literal in range -512 to 511 left shifted by 2 bits, i.e., -2048, -2044, ..., 2040, 2044;
- `imm_n4096_4088`, an integer literal in range -512 to 511 left shifted by 3 bits, i.e., -4096, -4088, ..., 4080, 4088;
- `imm1_4`, an integer literal in range 1 to 4;
- `i32`, `i64`, `u32`, `u64`, `f32`, `f64`, defined as follows:


```

{
    typedef int i32;
    #if __LONG_MAX__ == __LONG_LONG_MAX__
        typedef long i64;
    #else
        typedef long long i64;
    #endif

    typedef unsigned int u32;
    #if __LONG_MAX__ == __LONG_LONG_MAX__
        typedef unsigned long u64;
    #else
        typedef unsigned long long u64;
    #endif

    typedef double f64;
    typedef float f32;
}

```

7.13.18.1 MIPS SIMD Architecture Built-in Functions

The intrinsics provided are listed below; each is named after the machine instruction.

```

v16i8 __builtin_msa_add_a_b (v16i8, v16i8);
v8i16 __builtin_msa_add_a_h (v8i16, v8i16);
v4i32 __builtin_msa_add_a_w (v4i32, v4i32);
v2i64 __builtin_msa_add_a_d (v2i64, v2i64);

v16i8 __builtin_msa_adds_a_b (v16i8, v16i8);
v8i16 __builtin_msa_adds_a_h (v8i16, v8i16);
v4i32 __builtin_msa_adds_a_w (v4i32, v4i32);
v2i64 __builtin_msa_adds_a_d (v2i64, v2i64);

v16i8 __builtin_msa_adds_s_b (v16i8, v16i8);
v8i16 __builtin_msa_adds_s_h (v8i16, v8i16);
v4i32 __builtin_msa_adds_s_w (v4i32, v4i32);

```

```

v2i64 __builtin_msa_adds_s_d (v2i64, v2i64);

v16u8 __builtin_msa_adds_u_b (v16u8, v16u8);
v8u16 __builtin_msa_adds_u_h (v8u16, v8u16);
v4u32 __builtin_msa_adds_u_w (v4u32, v4u32);
v2u64 __builtin_msa_adds_u_d (v2u64, v2u64);

v16i8 __builtin_msa_addv_b (v16i8, v16i8);
v8i16 __builtin_msa_addv_h (v8i16, v8i16);
v4i32 __builtin_msa_addv_w (v4i32, v4i32);
v2i64 __builtin_msa_addv_d (v2i64, v2i64);

v16i8 __builtin_msa_addvi_b (v16i8, imm0_31);
v8i16 __builtin_msa_addvi_h (v8i16, imm0_31);
v4i32 __builtin_msa_addvi_w (v4i32, imm0_31);
v2i64 __builtin_msa_addvi_d (v2i64, imm0_31);

v16u8 __builtin_msa_and_v (v16u8, v16u8);

v16u8 __builtin_msa_andi_b (v16u8, imm0_255);

v16i8 __builtin_msa_asub_s_b (v16i8, v16i8);
v8i16 __builtin_msa_asub_s_h (v8i16, v8i16);
v4i32 __builtin_msa_asub_s_w (v4i32, v4i32);
v2i64 __builtin_msa_asub_s_d (v2i64, v2i64);

v16u8 __builtin_msa_asub_u_b (v16u8, v16u8);
v8u16 __builtin_msa_asub_u_h (v8u16, v8u16);
v4u32 __builtin_msa_asub_u_w (v4u32, v4u32);
v2u64 __builtin_msa_asub_u_d (v2u64, v2u64);

v16i8 __builtin_msa_ave_s_b (v16i8, v16i8);
v8i16 __builtin_msa_ave_s_h (v8i16, v8i16);
v4i32 __builtin_msa_ave_s_w (v4i32, v4i32);
v2i64 __builtin_msa_ave_s_d (v2i64, v2i64);

v16u8 __builtin_msa_ave_u_b (v16u8, v16u8);
v8u16 __builtin_msa_ave_u_h (v8u16, v8u16);
v4u32 __builtin_msa_ave_u_w (v4u32, v4u32);
v2u64 __builtin_msa_ave_u_d (v2u64, v2u64);

v16i8 __builtin_msa_aver_s_b (v16i8, v16i8);
v8i16 __builtin_msa_aver_s_h (v8i16, v8i16);
v4i32 __builtin_msa_aver_s_w (v4i32, v4i32);
v2i64 __builtin_msa_aver_s_d (v2i64, v2i64);

v16u8 __builtin_msa_aver_u_b (v16u8, v16u8);
v8u16 __builtin_msa_aver_u_h (v8u16, v8u16);
v4u32 __builtin_msa_aver_u_w (v4u32, v4u32);
v2u64 __builtin_msa_aver_u_d (v2u64, v2u64);

v16u8 __builtin_msa_bclr_b (v16u8, v16u8);
v8u16 __builtin_msa_bclr_h (v8u16, v8u16);
v4u32 __builtin_msa_bclr_w (v4u32, v4u32);
v2u64 __builtin_msa_bclr_d (v2u64, v2u64);

v16u8 __builtin_msa_bclri_b (v16u8, imm0_7);
v8u16 __builtin_msa_bclri_h (v8u16, imm0_15);

```

```

v4u32 __builtin_msa_bclri_w (v4u32, imm0_31);
v2u64 __builtin_msa_bclri_d (v2u64, imm0_63);

v16u8 __builtin_msa_binsl_b (v16u8, v16u8, v16u8);
v8u16 __builtin_msa_binsl_h (v8u16, v8u16, v8u16);
v4u32 __builtin_msa_binsl_w (v4u32, v4u32, v4u32);
v2u64 __builtin_msa_binsl_d (v2u64, v2u64, v2u64);

v16u8 __builtin_msa_binsli_b (v16u8, v16u8, imm0_7);
v8u16 __builtin_msa_binsli_h (v8u16, v8u16, imm0_15);
v4u32 __builtin_msa_binsli_w (v4u32, v4u32, imm0_31);
v2u64 __builtin_msa_binsli_d (v2u64, v2u64, imm0_63);

v16u8 __builtin_msa_binsr_b (v16u8, v16u8, v16u8);
v8u16 __builtin_msa_binsr_h (v8u16, v8u16, v8u16);
v4u32 __builtin_msa_binsr_w (v4u32, v4u32, v4u32);
v2u64 __builtin_msa_binsr_d (v2u64, v2u64, v2u64);

v16u8 __builtin_msa_binsri_b (v16u8, v16u8, imm0_7);
v8u16 __builtin_msa_binsri_h (v8u16, v8u16, imm0_15);
v4u32 __builtin_msa_binsri_w (v4u32, v4u32, imm0_31);
v2u64 __builtin_msa_binsri_d (v2u64, v2u64, imm0_63);

v16u8 __builtin_msa_bmnz_v (v16u8, v16u8, v16u8);

v16u8 __builtin_msa_bmnzi_b (v16u8, v16u8, imm0_255);

v16u8 __builtin_msa_bmz_v (v16u8, v16u8, v16u8);

v16u8 __builtin_msa_bmzi_b (v16u8, v16u8, imm0_255);

v16u8 __builtin_msa_bneg_b (v16u8, v16u8);
v8u16 __builtin_msa_bneg_h (v8u16, v8u16);
v4u32 __builtin_msa_bneg_w (v4u32, v4u32);
v2u64 __builtin_msa_bneg_d (v2u64, v2u64);

v16u8 __builtin_msa_bnegi_b (v16u8, imm0_7);
v8u16 __builtin_msa_bnegi_h (v8u16, imm0_15);
v4u32 __builtin_msa_bnegi_w (v4u32, imm0_31);
v2u64 __builtin_msa_bnegi_d (v2u64, imm0_63);

i32 __builtin_msa_bnz_b (v16u8);
i32 __builtin_msa_bnz_h (v8u16);
i32 __builtin_msa_bnz_w (v4u32);
i32 __builtin_msa_bnz_d (v2u64);

i32 __builtin_msa_bnz_v (v16u8);

v16u8 __builtin_msa_bsel_v (v16u8, v16u8, v16u8);

v16u8 __builtin_msa_bseli_b (v16u8, v16u8, imm0_255);

v16u8 __builtin_msa_bset_b (v16u8, v16u8);
v8u16 __builtin_msa_bset_h (v8u16, v8u16);
v4u32 __builtin_msa_bset_w (v4u32, v4u32);
v2u64 __builtin_msa_bset_d (v2u64, v2u64);

v16u8 __builtin_msa_bseti_b (v16u8, imm0_7);

```

```

v8u16 __builtin_msa_bseti_h (v8u16, imm0_15);
v4u32 __builtin_msa_bseti_w (v4u32, imm0_31);
v2u64 __builtin_msa_bseti_d (v2u64, imm0_63);

i32 __builtin_msa_bz_b (v16u8);
i32 __builtin_msa_bz_h (v8u16);
i32 __builtin_msa_bz_w (v4u32);
i32 __builtin_msa_bz_d (v2u64);

i32 __builtin_msa_bz_v (v16u8);

v16i8 __builtin_msa_ceq_b (v16i8, v16i8);
v8i16 __builtin_msa_ceq_h (v8i16, v8i16);
v4i32 __builtin_msa_ceq_w (v4i32, v4i32);
v2i64 __builtin_msa_ceq_d (v2i64, v2i64);

v16i8 __builtin_msa_ceqi_b (v16i8, imm_n16_15);
v8i16 __builtin_msa_ceqi_h (v8i16, imm_n16_15);
v4i32 __builtin_msa_ceqi_w (v4i32, imm_n16_15);
v2i64 __builtin_msa_ceqi_d (v2i64, imm_n16_15);

i32 __builtin_msa_cfcmsa (imm0_31);

v16i8 __builtin_msa_cle_s_b (v16i8, v16i8);
v8i16 __builtin_msa_cle_s_h (v8i16, v8i16);
v4i32 __builtin_msa_cle_s_w (v4i32, v4i32);
v2i64 __builtin_msa_cle_s_d (v2i64, v2i64);

v16i8 __builtin_msa_cle_u_b (v16u8, v16u8);
v8i16 __builtin_msa_cle_u_h (v8u16, v8u16);
v4i32 __builtin_msa_cle_u_w (v4u32, v4u32);
v2i64 __builtin_msa_cle_u_d (v2u64, v2u64);

v16i8 __builtin_msa_clei_s_b (v16i8, imm_n16_15);
v8i16 __builtin_msa_clei_s_h (v8i16, imm_n16_15);
v4i32 __builtin_msa_clei_s_w (v4i32, imm_n16_15);
v2i64 __builtin_msa_clei_s_d (v2i64, imm_n16_15);

v16i8 __builtin_msa_clei_u_b (v16u8, imm0_31);
v8i16 __builtin_msa_clei_u_h (v8u16, imm0_31);
v4i32 __builtin_msa_clei_u_w (v4u32, imm0_31);
v2i64 __builtin_msa_clei_u_d (v2u64, imm0_31);

v16i8 __builtin_msa_clt_s_b (v16i8, v16i8);
v8i16 __builtin_msa_clt_s_h (v8i16, v8i16);
v4i32 __builtin_msa_clt_s_w (v4i32, v4i32);
v2i64 __builtin_msa_clt_s_d (v2i64, v2i64);

v16i8 __builtin_msa_clt_u_b (v16u8, v16u8);
v8i16 __builtin_msa_clt_u_h (v8u16, v8u16);
v4i32 __builtin_msa_clt_u_w (v4u32, v4u32);
v2i64 __builtin_msa_clt_u_d (v2u64, v2u64);

v16i8 __builtin_msa_clti_s_b (v16i8, imm_n16_15);
v8i16 __builtin_msa_clti_s_h (v8i16, imm_n16_15);
v4i32 __builtin_msa_clti_s_w (v4i32, imm_n16_15);
v2i64 __builtin_msa_clti_s_d (v2i64, imm_n16_15);

```

```

v16i8 __builtin_msa_clti_u_b (v16u8, imm0_31);
v8i16 __builtin_msa_clti_u_h (v8u16, imm0_31);
v4i32 __builtin_msa_clti_u_w (v4u32, imm0_31);
v2i64 __builtin_msa_clti_u_d (v2u64, imm0_31);

i32 __builtin_msa_copy_s_b (v16i8, imm0_15);
i32 __builtin_msa_copy_s_h (v8i16, imm0_7);
i32 __builtin_msa_copy_s_w (v4i32, imm0_3);
i64 __builtin_msa_copy_s_d (v2i64, imm0_1);

u32 __builtin_msa_copy_u_b (v16i8, imm0_15);
u32 __builtin_msa_copy_u_h (v8i16, imm0_7);
u32 __builtin_msa_copy_u_w (v4i32, imm0_3);
u64 __builtin_msa_copy_u_d (v2i64, imm0_1);

void __builtin_msa_ctcmsa (imm0_31, i32);

v16i8 __builtin_msa_div_s_b (v16i8, v16i8);
v8i16 __builtin_msa_div_s_h (v8i16, v8i16);
v4i32 __builtin_msa_div_s_w (v4i32, v4i32);
v2i64 __builtin_msa_div_s_d (v2i64, v2i64);

v16u8 __builtin_msa_div_u_b (v16u8, v16u8);
v8u16 __builtin_msa_div_u_h (v8u16, v8u16);
v4u32 __builtin_msa_div_u_w (v4u32, v4u32);
v2u64 __builtin_msa_div_u_d (v2u64, v2u64);

v8i16 __builtin_msa_dotp_s_h (v16i8, v16i8);
v4i32 __builtin_msa_dotp_s_w (v8i16, v8i16);
v2i64 __builtin_msa_dotp_s_d (v4i32, v4i32);

v8u16 __builtin_msa_dotp_u_h (v16u8, v16u8);
v4u32 __builtin_msa_dotp_u_w (v8u16, v8u16);
v2u64 __builtin_msa_dotp_u_d (v4u32, v4u32);

v8i16 __builtin_msa_dpadd_s_h (v8i16, v16i8, v16i8);
v4i32 __builtin_msa_dpadd_s_w (v4i32, v8i16, v8i16);
v2i64 __builtin_msa_dpadd_s_d (v2i64, v4i32, v4i32);

v8u16 __builtin_msa_dpadd_u_h (v8u16, v16u8, v16u8);
v4u32 __builtin_msa_dpadd_u_w (v4u32, v8u16, v8u16);
v2u64 __builtin_msa_dpadd_u_d (v2u64, v4u32, v4u32);

v8i16 __builtin_msa_dpsub_s_h (v8i16, v16i8, v16i8);
v4i32 __builtin_msa_dpsub_s_w (v4i32, v8i16, v8i16);
v2i64 __builtin_msa_dpsub_s_d (v2i64, v4i32, v4i32);

v8i16 __builtin_msa_dpsub_u_h (v8i16, v16u8, v16u8);
v4i32 __builtin_msa_dpsub_u_w (v4i32, v8u16, v8u16);
v2i64 __builtin_msa_dpsub_u_d (v2i64, v4u32, v4u32);

v4f32 __builtin_msa_fadd_w (v4f32, v4f32);
v2f64 __builtin_msa_fadd_d (v2f64, v2f64);

v4i32 __builtin_msa_fcaf_w (v4f32, v4f32);
v2i64 __builtin_msa_fcaf_d (v2f64, v2f64);

v4i32 __builtin_msa_fceq_w (v4f32, v4f32);

```

```

v2i64 __builtin_msa_fceq_d (v2f64, v2f64);

v4i32 __builtin_msa_fclass_w (v4f32);
v2i64 __builtin_msa_fclass_d (v2f64);

v4i32 __builtin_msa_fcle_w (v4f32, v4f32);
v2i64 __builtin_msa_fcle_d (v2f64, v2f64);

v4i32 __builtin_msa_fclt_w (v4f32, v4f32);
v2i64 __builtin_msa_fclt_d (v2f64, v2f64);

v4i32 __builtin_msa_fcne_w (v4f32, v4f32);
v2i64 __builtin_msa_fcne_d (v2f64, v2f64);

v4i32 __builtin_msa_fcor_w (v4f32, v4f32);
v2i64 __builtin_msa_fcor_d (v2f64, v2f64);

v4i32 __builtin_msa_fcueq_w (v4f32, v4f32);
v2i64 __builtin_msa_fcueq_d (v2f64, v2f64);

v4i32 __builtin_msa_fcule_w (v4f32, v4f32);
v2i64 __builtin_msa_fcule_d (v2f64, v2f64);

v4i32 __builtin_msa_fcult_w (v4f32, v4f32);
v2i64 __builtin_msa_fcult_d (v2f64, v2f64);

v4i32 __builtin_msa_fcun_w (v4f32, v4f32);
v2i64 __builtin_msa_fcun_d (v2f64, v2f64);

v4i32 __builtin_msa_fcune_w (v4f32, v4f32);
v2i64 __builtin_msa_fcune_d (v2f64, v2f64);

v4f32 __builtin_msa_fdiv_w (v4f32, v4f32);
v2f64 __builtin_msa_fdiv_d (v2f64, v2f64);

v8i16 __builtin_msa_fexdo_h (v4f32, v4f32);
v4f32 __builtin_msa_fexdo_w (v2f64, v2f64);

v4f32 __builtin_msa_fexp2_w (v4f32, v4i32);
v2f64 __builtin_msa_fexp2_d (v2f64, v2i64);

v4f32 __builtin_msa_fexupl_w (v8i16);
v2f64 __builtin_msa_fexupl_d (v4f32);

v4f32 __builtin_msa_fexupr_w (v8i16);
v2f64 __builtin_msa_fexupr_d (v4f32);

v4f32 __builtin_msa_ffint_s_w (v4i32);
v2f64 __builtin_msa_ffint_s_d (v2i64);

v4f32 __builtin_msa_ffint_u_w (v4u32);
v2f64 __builtin_msa_ffint_u_d (v2u64);

v4f32 __builtin_msa_ffql_w (v8i16);
v2f64 __builtin_msa_ffql_d (v4i32);

v4f32 __builtin_msa_ffqr_w (v8i16);
v2f64 __builtin_msa_ffqr_d (v4i32);

```

```
v16i8 __builtin_msa_fill_b (i32);
v8i16 __builtin_msa_fill_h (i32);
v4i32 __builtin_msa_fill_w (i32);
v2i64 __builtin_msa_fill_d (i64);

v4f32 __builtin_msa_flog2_w (v4f32);
v2f64 __builtin_msa_flog2_d (v2f64);

v4f32 __builtin_msa_fmadd_w (v4f32, v4f32, v4f32);
v2f64 __builtin_msa_fmadd_d (v2f64, v2f64, v2f64);

v4f32 __builtin_msa_fmax_w (v4f32, v4f32);
v2f64 __builtin_msa_fmax_d (v2f64, v2f64);

v4f32 __builtin_msa_fmax_a_w (v4f32, v4f32);
v2f64 __builtin_msa_fmax_a_d (v2f64, v2f64);

v4f32 __builtin_msa_fmin_w (v4f32, v4f32);
v2f64 __builtin_msa_fmin_d (v2f64, v2f64);

v4f32 __builtin_msa_fmin_a_w (v4f32, v4f32);
v2f64 __builtin_msa_fmin_a_d (v2f64, v2f64);

v4f32 __builtin_msa_fmsub_w (v4f32, v4f32, v4f32);
v2f64 __builtin_msa_fmsub_d (v2f64, v2f64, v2f64);

v4f32 __builtin_msa_fmul_w (v4f32, v4f32);
v2f64 __builtin_msa_fmul_d (v2f64, v2f64);

v4f32 __builtin_msa_frint_w (v4f32);
v2f64 __builtin_msa_frint_d (v2f64);

v4f32 __builtin_msa_frcp_w (v4f32);
v2f64 __builtin_msa_frcp_d (v2f64);

v4f32 __builtin_msa_frsqrt_w (v4f32);
v2f64 __builtin_msa_frsqrt_d (v2f64);

v4i32 __builtin_msa_fsaf_w (v4f32, v4f32);
v2i64 __builtin_msa_fsaf_d (v2f64, v2f64);

v4i32 __builtin_msa_fseq_w (v4f32, v4f32);
v2i64 __builtin_msa_fseq_d (v2f64, v2f64);

v4i32 __builtin_msa_fsle_w (v4f32, v4f32);
v2i64 __builtin_msa_fsle_d (v2f64, v2f64);

v4i32 __builtin_msa_fslt_w (v4f32, v4f32);
v2i64 __builtin_msa_fslt_d (v2f64, v2f64);

v4i32 __builtin_msa_fsne_w (v4f32, v4f32);
v2i64 __builtin_msa_fsne_d (v2f64, v2f64);

v4i32 __builtin_msa_fsor_w (v4f32, v4f32);
v2i64 __builtin_msa_fsor_d (v2f64, v2f64);

v4f32 __builtin_msa_fsqrt_w (v4f32);
```

```

v2f64 __builtin_msa_fsqrt_d (v2f64);

v4f32 __builtin_msa_fsub_w (v4f32, v4f32);
v2f64 __builtin_msa_fsub_d (v2f64, v2f64);

v4i32 __builtin_msa_fsueq_w (v4f32, v4f32);
v2i64 __builtin_msa_fsueq_d (v2f64, v2f64);

v4i32 __builtin_msa_fsule_w (v4f32, v4f32);
v2i64 __builtin_msa_fsule_d (v2f64, v2f64);

v4i32 __builtin_msa_fsult_w (v4f32, v4f32);
v2i64 __builtin_msa_fsult_d (v2f64, v2f64);

v4i32 __builtin_msa_fsun_w (v4f32, v4f32);
v2i64 __builtin_msa_fsun_d (v2f64, v2f64);

v4i32 __builtin_msa_fsune_w (v4f32, v4f32);
v2i64 __builtin_msa_fsune_d (v2f64, v2f64);

v4i32 __builtin_msa_ftint_s_w (v4f32);
v2i64 __builtin_msa_ftint_s_d (v2f64);

v4u32 __builtin_msa_ftint_u_w (v4f32);
v2u64 __builtin_msa_ftint_u_d (v2f64);

v8i16 __builtin_msa_ftq_h (v4f32, v4f32);
v4i32 __builtin_msa_ftq_w (v2f64, v2f64);

v4i32 __builtin_msa_ftrunc_s_w (v4f32);
v2i64 __builtin_msa_ftrunc_s_d (v2f64);

v4u32 __builtin_msa_ftrunc_u_w (v4f32);
v2u64 __builtin_msa_ftrunc_u_d (v2f64);

v8i16 __builtin_msa_hadd_s_h (v16i8, v16i8);
v4i32 __builtin_msa_hadd_s_w (v8i16, v8i16);
v2i64 __builtin_msa_hadd_s_d (v4i32, v4i32);

v8u16 __builtin_msa_hadd_u_h (v16u8, v16u8);
v4u32 __builtin_msa_hadd_u_w (v8u16, v8u16);
v2u64 __builtin_msa_hadd_u_d (v4u32, v4u32);

v8i16 __builtin_msa_hsub_s_h (v16i8, v16i8);
v4i32 __builtin_msa_hsub_s_w (v8i16, v8i16);
v2i64 __builtin_msa_hsub_s_d (v4i32, v4i32);

v8i16 __builtin_msa_hsub_u_h (v16u8, v16u8);
v4i32 __builtin_msa_hsub_u_w (v8u16, v8u16);
v2i64 __builtin_msa_hsub_u_d (v4u32, v4u32);

v16i8 __builtin_msa_ilvev_b (v16i8, v16i8);
v8i16 __builtin_msa_ilvev_h (v8i16, v8i16);
v4i32 __builtin_msa_ilvev_w (v4i32, v4i32);
v2i64 __builtin_msa_ilvev_d (v2i64, v2i64);

v16i8 __builtin_msa_ilvl_b (v16i8, v16i8);
v8i16 __builtin_msa_ilvl_h (v8i16, v8i16);

```

```

v4i32 __builtin_msa_ilvl_w (v4i32, v4i32);
v2i64 __builtin_msa_ilvl_d (v2i64, v2i64);

v16i8 __builtin_msa_ilvod_b (v16i8, v16i8);
v8i16 __builtin_msa_ilvod_h (v8i16, v8i16);
v4i32 __builtin_msa_ilvod_w (v4i32, v4i32);
v2i64 __builtin_msa_ilvod_d (v2i64, v2i64);

v16i8 __builtin_msa_ilvr_b (v16i8, v16i8);
v8i16 __builtin_msa_ilvr_h (v8i16, v8i16);
v4i32 __builtin_msa_ilvr_w (v4i32, v4i32);
v2i64 __builtin_msa_ilvr_d (v2i64, v2i64);

v16i8 __builtin_msa_insert_b (v16i8, imm0_15, i32);
v8i16 __builtin_msa_insert_h (v8i16, imm0_7, i32);
v4i32 __builtin_msa_insert_w (v4i32, imm0_3, i32);
v2i64 __builtin_msa_insert_d (v2i64, imm0_1, i64);

v16i8 __builtin_msa_insve_b (v16i8, imm0_15, v16i8);
v8i16 __builtin_msa_insve_h (v8i16, imm0_7, v8i16);
v4i32 __builtin_msa_insve_w (v4i32, imm0_3, v4i32);
v2i64 __builtin_msa_insve_d (v2i64, imm0_1, v2i64);

v16i8 __builtin_msa_ld_b (const void *, imm_n512_511);
v8i16 __builtin_msa_ld_h (const void *, imm_n1024_1022);
v4i32 __builtin_msa_ld_w (const void *, imm_n2048_2044);
v2i64 __builtin_msa_ld_d (const void *, imm_n4096_4088);

v16i8 __builtin_msa_ldi_b (imm_n512_511);
v8i16 __builtin_msa_ldi_h (imm_n512_511);
v4i32 __builtin_msa_ldi_w (imm_n512_511);
v2i64 __builtin_msa_ldi_d (imm_n512_511);

v8i16 __builtin_msa_madd_q_h (v8i16, v8i16, v8i16);
v4i32 __builtin_msa_madd_q_w (v4i32, v4i32, v4i32);

v8i16 __builtin_msa_maddr_q_h (v8i16, v8i16, v8i16);
v4i32 __builtin_msa_maddr_q_w (v4i32, v4i32, v4i32);

v16i8 __builtin_msa_maddv_b (v16i8, v16i8, v16i8);
v8i16 __builtin_msa_maddv_h (v8i16, v8i16, v8i16);
v4i32 __builtin_msa_maddv_w (v4i32, v4i32, v4i32);
v2i64 __builtin_msa_maddv_d (v2i64, v2i64, v2i64);

v16i8 __builtin_msa_max_a_b (v16i8, v16i8);
v8i16 __builtin_msa_max_a_h (v8i16, v8i16);
v4i32 __builtin_msa_max_a_w (v4i32, v4i32);
v2i64 __builtin_msa_max_a_d (v2i64, v2i64);

v16i8 __builtin_msa_max_s_b (v16i8, v16i8);
v8i16 __builtin_msa_max_s_h (v8i16, v8i16);
v4i32 __builtin_msa_max_s_w (v4i32, v4i32);
v2i64 __builtin_msa_max_s_d (v2i64, v2i64);

v16u8 __builtin_msa_max_u_b (v16u8, v16u8);
v8u16 __builtin_msa_max_u_h (v8u16, v8u16);
v4u32 __builtin_msa_max_u_w (v4u32, v4u32);
v2u64 __builtin_msa_max_u_d (v2u64, v2u64);

```

```

v16i8 __builtin_msa_maxi_s_b (v16i8, imm_n16_15);
v8i16 __builtin_msa_maxi_s_h (v8i16, imm_n16_15);
v4i32 __builtin_msa_maxi_s_w (v4i32, imm_n16_15);
v2i64 __builtin_msa_maxi_s_d (v2i64, imm_n16_15);

v16u8 __builtin_msa_maxi_u_b (v16u8, imm0_31);
v8u16 __builtin_msa_maxi_u_h (v8u16, imm0_31);
v4u32 __builtin_msa_maxi_u_w (v4u32, imm0_31);
v2u64 __builtin_msa_maxi_u_d (v2u64, imm0_31);

v16i8 __builtin_msa_min_a_b (v16i8, v16i8);
v8i16 __builtin_msa_min_a_h (v8i16, v8i16);
v4i32 __builtin_msa_min_a_w (v4i32, v4i32);
v2i64 __builtin_msa_min_a_d (v2i64, v2i64);

v16i8 __builtin_msa_min_s_b (v16i8, v16i8);
v8i16 __builtin_msa_min_s_h (v8i16, v8i16);
v4i32 __builtin_msa_min_s_w (v4i32, v4i32);
v2i64 __builtin_msa_min_s_d (v2i64, v2i64);

v16u8 __builtin_msa_min_u_b (v16u8, v16u8);
v8u16 __builtin_msa_min_u_h (v8u16, v8u16);
v4u32 __builtin_msa_min_u_w (v4u32, v4u32);
v2u64 __builtin_msa_min_u_d (v2u64, v2u64);

v16i8 __builtin_msa_mini_s_b (v16i8, imm_n16_15);
v8i16 __builtin_msa_mini_s_h (v8i16, imm_n16_15);
v4i32 __builtin_msa_mini_s_w (v4i32, imm_n16_15);
v2i64 __builtin_msa_mini_s_d (v2i64, imm_n16_15);

v16u8 __builtin_msa_mini_u_b (v16u8, imm0_31);
v8u16 __builtin_msa_mini_u_h (v8u16, imm0_31);
v4u32 __builtin_msa_mini_u_w (v4u32, imm0_31);
v2u64 __builtin_msa_mini_u_d (v2u64, imm0_31);

v16i8 __builtin_msa_mod_s_b (v16i8, v16i8);
v8i16 __builtin_msa_mod_s_h (v8i16, v8i16);
v4i32 __builtin_msa_mod_s_w (v4i32, v4i32);
v2i64 __builtin_msa_mod_s_d (v2i64, v2i64);

v16u8 __builtin_msa_mod_u_b (v16u8, v16u8);
v8u16 __builtin_msa_mod_u_h (v8u16, v8u16);
v4u32 __builtin_msa_mod_u_w (v4u32, v4u32);
v2u64 __builtin_msa_mod_u_d (v2u64, v2u64);

v16i8 __builtin_msa_move_v (v16i8);

v8i16 __builtin_msa_msub_q_h (v8i16, v8i16, v8i16);
v4i32 __builtin_msa_msub_q_w (v4i32, v4i32, v4i32);

v8i16 __builtin_msa_msubr_q_h (v8i16, v8i16, v8i16);
v4i32 __builtin_msa_msubr_q_w (v4i32, v4i32, v4i32);

v16i8 __builtin_msa_msubv_b (v16i8, v16i8, v16i8);
v8i16 __builtin_msa_msubv_h (v8i16, v8i16, v8i16);
v4i32 __builtin_msa_msubv_w (v4i32, v4i32, v4i32);
v2i64 __builtin_msa_msubv_d (v2i64, v2i64, v2i64);

```

```

v8i16 __builtin_msa_mul_q_h (v8i16, v8i16);
v4i32 __builtin_msa_mul_q_w (v4i32, v4i32);

v8i16 __builtin_msa_mulr_q_h (v8i16, v8i16);
v4i32 __builtin_msa_mulr_q_w (v4i32, v4i32);

v16i8 __builtin_msa_mulv_b (v16i8, v16i8);
v8i16 __builtin_msa_mulv_h (v8i16, v8i16);
v4i32 __builtin_msa_mulv_w (v4i32, v4i32);
v2i64 __builtin_msa_mulv_d (v2i64, v2i64);

v16i8 __builtin_msa_nloc_b (v16i8);
v8i16 __builtin_msa_nloc_h (v8i16);
v4i32 __builtin_msa_nloc_w (v4i32);
v2i64 __builtin_msa_nloc_d (v2i64);

v16i8 __builtin_msa_nlzc_b (v16i8);
v8i16 __builtin_msa_nlzc_h (v8i16);
v4i32 __builtin_msa_nlzc_w (v4i32);
v2i64 __builtin_msa_nlzc_d (v2i64);

v16u8 __builtin_msa_nor_v (v16u8, v16u8);

v16u8 __builtin_msa_nori_b (v16u8, imm0_255);

v16u8 __builtin_msa_or_v (v16u8, v16u8);

v16u8 __builtin_msa_ori_b (v16u8, imm0_255);

v16i8 __builtin_msa_pckev_b (v16i8, v16i8);
v8i16 __builtin_msa_pckev_h (v8i16, v8i16);
v4i32 __builtin_msa_pckev_w (v4i32, v4i32);
v2i64 __builtin_msa_pckev_d (v2i64, v2i64);

v16i8 __builtin_msa_pckod_b (v16i8, v16i8);
v8i16 __builtin_msa_pckod_h (v8i16, v8i16);
v4i32 __builtin_msa_pckod_w (v4i32, v4i32);
v2i64 __builtin_msa_pckod_d (v2i64, v2i64);

v16i8 __builtin_msa_pcmt_b (v16i8);
v8i16 __builtin_msa_pcmt_h (v8i16);
v4i32 __builtin_msa_pcmt_w (v4i32);
v2i64 __builtin_msa_pcmt_d (v2i64);

v16i8 __builtin_msa_sat_s_b (v16i8, imm0_7);
v8i16 __builtin_msa_sat_s_h (v8i16, imm0_15);
v4i32 __builtin_msa_sat_s_w (v4i32, imm0_31);
v2i64 __builtin_msa_sat_s_d (v2i64, imm0_63);

v16u8 __builtin_msa_sat_u_b (v16u8, imm0_7);
v8u16 __builtin_msa_sat_u_h (v8u16, imm0_15);
v4u32 __builtin_msa_sat_u_w (v4u32, imm0_31);
v2u64 __builtin_msa_sat_u_d (v2u64, imm0_63);

v16i8 __builtin_msa_shf_b (v16i8, imm0_255);
v8i16 __builtin_msa_shf_h (v8i16, imm0_255);
v4i32 __builtin_msa_shf_w (v4i32, imm0_255);

```

```

v16i8 __builtin_msa_sld_b (v16i8, v16i8, i32);
v8i16 __builtin_msa_sld_h (v8i16, v8i16, i32);
v4i32 __builtin_msa_sld_w (v4i32, v4i32, i32);
v2i64 __builtin_msa_sld_d (v2i64, v2i64, i32);

v16i8 __builtin_msa_sldi_b (v16i8, v16i8, imm0_15);
v8i16 __builtin_msa_sldi_h (v8i16, v8i16, imm0_7);
v4i32 __builtin_msa_sldi_w (v4i32, v4i32, imm0_3);
v2i64 __builtin_msa_sldi_d (v2i64, v2i64, imm0_1);

v16i8 __builtin_msa_sll_b (v16i8, v16i8);
v8i16 __builtin_msa_sll_h (v8i16, v8i16);
v4i32 __builtin_msa_sll_w (v4i32, v4i32);
v2i64 __builtin_msa_sll_d (v2i64, v2i64);

v16i8 __builtin_msa_slli_b (v16i8, imm0_7);
v8i16 __builtin_msa_slli_h (v8i16, imm0_15);
v4i32 __builtin_msa_slli_w (v4i32, imm0_31);
v2i64 __builtin_msa_slli_d (v2i64, imm0_63);

v16i8 __builtin_msa_splat_b (v16i8, i32);
v8i16 __builtin_msa_splat_h (v8i16, i32);
v4i32 __builtin_msa_splat_w (v4i32, i32);
v2i64 __builtin_msa_splat_d (v2i64, i32);

v16i8 __builtin_msa_splati_b (v16i8, imm0_15);
v8i16 __builtin_msa_splati_h (v8i16, imm0_7);
v4i32 __builtin_msa_splati_w (v4i32, imm0_3);
v2i64 __builtin_msa_splati_d (v2i64, imm0_1);

v16i8 __builtin_msa_sra_b (v16i8, v16i8);
v8i16 __builtin_msa_sra_h (v8i16, v8i16);
v4i32 __builtin_msa_sra_w (v4i32, v4i32);
v2i64 __builtin_msa_sra_d (v2i64, v2i64);

v16i8 __builtin_msa_srai_b (v16i8, imm0_7);
v8i16 __builtin_msa_srai_h (v8i16, imm0_15);
v4i32 __builtin_msa_srai_w (v4i32, imm0_31);
v2i64 __builtin_msa_srai_d (v2i64, imm0_63);

v16i8 __builtin_msa_srar_b (v16i8, v16i8);
v8i16 __builtin_msa_srar_h (v8i16, v8i16);
v4i32 __builtin_msa_srar_w (v4i32, v4i32);
v2i64 __builtin_msa_srar_d (v2i64, v2i64);

v16i8 __builtin_msa_srari_b (v16i8, imm0_7);
v8i16 __builtin_msa_srari_h (v8i16, imm0_15);
v4i32 __builtin_msa_srari_w (v4i32, imm0_31);
v2i64 __builtin_msa_srari_d (v2i64, imm0_63);

v16i8 __builtin_msa_srl_b (v16i8, v16i8);
v8i16 __builtin_msa_srl_h (v8i16, v8i16);
v4i32 __builtin_msa_srl_w (v4i32, v4i32);
v2i64 __builtin_msa_srl_d (v2i64, v2i64);

v16i8 __builtin_msa_srli_b (v16i8, imm0_7);
v8i16 __builtin_msa_srli_h (v8i16, imm0_15);

```

```

v4i32 __builtin_msa_srli_w (v4i32, imm0_31);
v2i64 __builtin_msa_srli_d (v2i64, imm0_63);

v16i8 __builtin_msa_srlr_b (v16i8, v16i8);
v8i16 __builtin_msa_srlr_h (v8i16, v8i16);
v4i32 __builtin_msa_srlr_w (v4i32, v4i32);
v2i64 __builtin_msa_srlr_d (v2i64, v2i64);

v16i8 __builtin_msa_srlri_b (v16i8, imm0_7);
v8i16 __builtin_msa_srlri_h (v8i16, imm0_15);
v4i32 __builtin_msa_srlri_w (v4i32, imm0_31);
v2i64 __builtin_msa_srlri_d (v2i64, imm0_63);

void __builtin_msa_st_b (v16i8, void *, imm_n512_511);
void __builtin_msa_st_h (v8i16, void *, imm_n1024_1022);
void __builtin_msa_st_w (v4i32, void *, imm_n2048_2044);
void __builtin_msa_st_d (v2i64, void *, imm_n4096_4088);

v16i8 __builtin_msa_subs_s_b (v16i8, v16i8);
v8i16 __builtin_msa_subs_s_h (v8i16, v8i16);
v4i32 __builtin_msa_subs_s_w (v4i32, v4i32);
v2i64 __builtin_msa_subs_s_d (v2i64, v2i64);

v16u8 __builtin_msa_subs_u_b (v16u8, v16u8);
v8u16 __builtin_msa_subs_u_h (v8u16, v8u16);
v4u32 __builtin_msa_subs_u_w (v4u32, v4u32);
v2u64 __builtin_msa_subs_u_d (v2u64, v2u64);

v16u8 __builtin_msa_subsus_u_b (v16u8, v16i8);
v8u16 __builtin_msa_subsus_u_h (v8u16, v8i16);
v4u32 __builtin_msa_subsus_u_w (v4u32, v4i32);
v2u64 __builtin_msa_subsus_u_d (v2u64, v2i64);

v16i8 __builtin_msa_subsuu_s_b (v16u8, v16u8);
v8i16 __builtin_msa_subsuu_s_h (v8u16, v8u16);
v4i32 __builtin_msa_subsuu_s_w (v4u32, v4u32);
v2i64 __builtin_msa_subsuu_s_d (v2u64, v2u64);

v16i8 __builtin_msa_subv_b (v16i8, v16i8);
v8i16 __builtin_msa_subv_h (v8i16, v8i16);
v4i32 __builtin_msa_subv_w (v4i32, v4i32);
v2i64 __builtin_msa_subv_d (v2i64, v2i64);

v16i8 __builtin_msa_subvi_b (v16i8, imm0_31);
v8i16 __builtin_msa_subvi_h (v8i16, imm0_31);
v4i32 __builtin_msa_subvi_w (v4i32, imm0_31);
v2i64 __builtin_msa_subvi_d (v2i64, imm0_31);

v16i8 __builtin_msa_vshf_b (v16i8, v16i8, v16i8);
v8i16 __builtin_msa_vshf_h (v8i16, v8i16, v8i16);
v4i32 __builtin_msa_vshf_w (v4i32, v4i32, v4i32);
v2i64 __builtin_msa_vshf_d (v2i64, v2i64, v2i64);

v16u8 __builtin_msa_xor_v (v16u8, v16u8);

v16u8 __builtin_msa_xori_b (v16u8, imm0_255);

```

7.13.19 Other MIPS Built-in Functions

GCC provides other MIPS-specific built-in functions:

void __builtin_mips_cache (int *op*, const volatile void **addr*)
 Insert a ‘cache’ instruction with operands *op* and *addr*. GCC defines the preprocessor macro `___GCC_HAVE_BUILTIN_MIPS_CACHE` when this function is available.

unsigned int __builtin_mips_get_fcsr (void)
void __builtin_mips_set_fcsr (unsigned int *value*)
 Get and set the contents of the floating-point control and status register (FPU control register 31). These functions are only available in hard-float code but can be called in both MIPS16 and non-MIPS16 contexts.
__builtin_mips_set_fcsr can be used to change any bit of the register except the condition codes, which GCC assumes are preserved.

7.13.20 MSP430 Built-in Functions

GCC provides a couple of special builtin functions to aid in the writing of interrupt handlers in C.

__bic_SR_register_on_exit (int *mask*)
 This clears the indicated bits in the saved copy of the status register currently residing on the stack. This only works inside interrupt handlers and the changes to the status register will only take affect once the handler returns.

__bis_SR_register_on_exit (int *mask*)
 This sets the indicated bits in the saved copy of the status register currently residing on the stack. This only works inside interrupt handlers and the changes to the status register will only take affect once the handler returns.

__delay_cycles (long long *cycles*)
 This inserts an instruction sequence that takes exactly *cycles* cycles (between 0 and about 17E9) to complete. The inserted sequence may use jumps, loops, or no-ops, and does not interfere with any other instructions. Note that *cycles* must be a compile-time constant integer - that is, you must pass a number, not a variable that may be optimized to a constant later. The number of cycles delayed by this builtin is exact.

7.13.21 NDS32 Built-in Functions

These built-in functions are available for the NDS32 target:

void __builtin_nds32_isync (int **addr*) [Built-in Function]
 Insert an ISYNC instruction into the instruction stream where *addr* is an instruction address for serialization.

void __builtin_nds32_isb (void) [Built-in Function]
 Insert an ISB instruction into the instruction stream.

int __builtin_nds32_mfsr (int *sr*) [Built-in Function]
 Return the content of a system register which is mapped by *sr*.

`int __builtin_nds32_mfusr (int usr)` [Built-in Function]
 Return the content of a user space register which is mapped by *usr*.

`void __builtin_nds32_mtsr (int value, int sr)` [Built-in Function]
 Move the *value* to a system register which is mapped by *sr*.

`void __builtin_nds32_mtusr (int value, int usr)` [Built-in Function]
 Move the *value* to a user space register which is mapped by *usr*.

`void __builtin_nds32_setgie_en (void)` [Built-in Function]
 Enable global interrupt.

`void __builtin_nds32_setgie_dis (void)` [Built-in Function]
 Disable global interrupt.

7.13.22 Nvidia PTX Built-in Functions

These built-in functions are available for the Nvidia PTX target:

`unsigned int __builtin_nvptx_brev (unsigned int x)` [Built-in Function]
 Reverse the bit order of a 32-bit unsigned integer.

`unsigned long long __builtin_nvptx_brevll (unsigned long long x)` [Built-in Function]
 Reverse the bit order of a 64-bit unsigned integer.

7.13.23 Basic PowerPC Built-in Functions

This section describes PowerPC built-in functions that do not require the inclusion of any special header files to declare prototypes or provide macro definitions. The sections that follow describe additional PowerPC built-in functions.

7.13.23.1 Basic PowerPC Built-in Functions Available on all Configurations

`void __builtin_cpu_init (void)` [Built-in Function]
 This function is a `nop` on the PowerPC platform and is included solely to maintain API compatibility with the x86 builtins.

`int __builtin_cpu_is (const char *cpuname)` [Built-in Function]
 This function returns a value of 1 if the run-time CPU is of type *cpuname* and returns 0 otherwise.

The `__builtin_cpu_is` function requires GLIBC 2.23 or newer which exports the hardware capability bits. GCC defines the macro `__BUILTIN_CPU_SUPPORTS__` if the `__builtin_cpu_supports` built-in function is fully supported.

If GCC was configured to use a GLIBC before 2.23, the built-in function `__builtin_cpu_is` always returns a 0 and the compiler issues a warning.

The following CPU names can be detected:

‘power10’ IBM POWER10 Server CPU.

‘power9’ IBM POWER9 Server CPU.

‘power8’ IBM POWER8 Server CPU.
 ‘power7’ IBM POWER7 Server CPU.
 ‘power6x’ IBM POWER6 Server CPU (RAW mode).
 ‘power6’ IBM POWER6 Server CPU (Architected mode).
 ‘power5+’ IBM POWER5+ Server CPU.
 ‘power5’ IBM POWER5 Server CPU.
 ‘ppc970’ IBM 970 Server CPU (ie, Apple G5).
 ‘power4’ IBM POWER4 Server CPU.
 ‘ppca2’ IBM A2 64-bit Embedded CPU
 ‘ppc476’ IBM PowerPC 476FP 32-bit Embedded CPU.
 ‘ppc464’ IBM PowerPC 464 32-bit Embedded CPU.
 ‘ppc440’ PowerPC 440 32-bit Embedded CPU.
 ‘ppc405’ PowerPC 405 32-bit Embedded CPU.
 ‘ppc-cell-be’
 IBM PowerPC Cell Broadband Engine Architecture CPU.

Here is an example:

```

#ifdef __BUILTIN_CPU_SUPPORTS__
  if (__builtin_cpu_is ("power8"))
  {
    do_power8 (); // POWER8 specific implementation.
  }
  else
#endif
  {
    do_generic (); // Generic implementation.
  }

```

int __builtin_cpu_supports (const char *feature) [Built-in Function]

This function returns a value of 1 if the run-time CPU supports the HWCAP feature *feature* and returns 0 otherwise.

The `__builtin_cpu_supports` function requires GLIBC 2.23 or newer which exports the hardware capability bits. GCC defines the macro `__BUILTIN_CPU_SUPPORTS__` if the `__builtin_cpu_supports` built-in function is fully supported.

If GCC was configured to use a GLIBC before 2.23, the built-in function `__builtin_cpu_supports` always returns a 0 and the compiler issues a warning.

The following features can be detected:

‘4xxmac’ 4xx CPU has a Multiply Accumulator.
 ‘altivec’ CPU has a SIMD/Vector Unit.
 ‘arch_2_05’
 CPU supports ISA 2.05 (eg, POWER6)

<code>'arch_2_06'</code>	CPU supports ISA 2.06 (eg, POWER7)
<code>'arch_2_07'</code>	CPU supports ISA 2.07 (eg, POWER8)
<code>'arch_3_00'</code>	CPU supports ISA 3.0 (eg, POWER9)
<code>'arch_3_1'</code>	CPU supports ISA 3.1 (eg, POWER10)
<code>'archpmu'</code>	CPU supports the set of compatible performance monitoring events.
<code>'booke'</code>	CPU supports the Embedded ISA category.
<code>'cellbe'</code>	CPU has a CELL broadband engine.
<code>'darn'</code>	CPU supports the darn (deliver a random number) instruction.
<code>'dfp'</code>	CPU has a decimal floating point unit.
<code>'dscr'</code>	CPU supports the data stream control register.
<code>'ebb'</code>	CPU supports event base branching.
<code>'efpdouble'</code>	CPU has a SPE double precision floating point unit.
<code>'efpsingle'</code>	CPU has a SPE single precision floating point unit.
<code>'fpu'</code>	CPU has a floating point unit.
<code>'htm'</code>	CPU has hardware transaction memory instructions.
<code>'htm-nosc'</code>	Kernel aborts hardware transactions when a syscall is made.
<code>'htm-no-suspend'</code>	CPU supports hardware transaction memory but does not support the tsuspend . instruction.
<code>'ic_snoop'</code>	CPU supports icache snooping capabilities.
<code>'ieee128'</code>	CPU supports 128-bit IEEE binary floating point instructions.
<code>'isel'</code>	CPU supports the integer select instruction.
<code>'mma'</code>	CPU supports the matrix-multiply assist instructions.
<code>'mmu'</code>	CPU has a memory management unit.
<code>'notb'</code>	CPU does not have a timebase (eg, 601 and 403gx).
<code>'pa6t'</code>	CPU supports the PA Semi 6T CORE ISA.
<code>'power4'</code>	CPU supports ISA 2.00 (eg, POWER4)
<code>'power5'</code>	CPU supports ISA 2.02 (eg, POWER5)

‘power5+’ CPU supports ISA 2.03 (eg, POWER5+)
 ‘power6x’ CPU supports ISA 2.05 (eg, POWER6) extended opcodes mffgpr and mftgpr.
 ‘ppc32’ CPU supports 32-bit mode execution.
 ‘ppc601’ CPU supports the old POWER ISA (eg, 601)
 ‘ppc64’ CPU supports 64-bit mode execution.
 ‘ppcle’ CPU supports a little-endian mode that uses address swizzling.
 ‘scv’ Kernel supports system call vectored.
 ‘smt’ CPU support simultaneous multi-threading.
 ‘spe’ CPU has a signal processing extension unit.
 ‘tar’ CPU supports the target address register.
 ‘true_le’ CPU supports true little-endian mode.
 ‘ucache’ CPU has unified I/D cache.
 ‘vcrypto’ CPU supports the vector cryptography instructions.
 ‘vsx’ CPU supports the vector-scalar extension.

Here is an example:

```

#ifdef __BUILTIN_CPU_SUPPORTS__
  if (__builtin_cpu_supports ("fpu"))
  {
    asm("fadd %0,%1,%2" : "=d"(dst) : "d"(src1), "d"(src2));
  }
  else
#endif
  {
    dst = __fadd (src1, src2); // Software FP addition function.
  }

```

The following built-in functions are also available on all PowerPC processors:

```

uint64_t __builtin_ppc_get_timebase ();
unsigned long __builtin_ppc_mftb ();
double __builtin_unpack_ibm128 (__ibm128, int);
__ibm128 __builtin_pack_ibm128 (double, double);
double __builtin_mffs (void);
void __builtin_mtfss (const int, double);
void __builtin_mtfssb0 (const int);
void __builtin_mtfssl (const int);
double __builtin_set_fpscr_rn (int);

```

The `__builtin_ppc_get_timebase` and `__builtin_ppc_mftb` functions generate instructions to read the Time Base Register. The `__builtin_ppc_get_timebase` function may generate multiple instructions and always returns the 64 bits of the Time Base Register. The `__builtin_ppc_mftb` function always generates one instruction and returns the Time Base Register value as an unsigned long, throwing away the most significant word on 32-bit environments. The `__builtin_mffs` return the value of the FPSCR register. Note, ISA 3.0 supports the `__builtin_mffsl()` which permits software to read

the control and non-sticky status bits in the FPSCR without the higher latency associated with accessing the sticky status bits. The `__builtin_mtf sf` takes a constant 8-bit integer field mask and a double precision floating point argument and generates the `mtf sf` (extended mnemonic) instruction to write new values to selected fields of the FPSCR. The `__builtin_mtf sb0` and `__builtin_mtf sb1` take the bit to change as an argument. The valid bit range is between 0 and 31. The builtins map to the `mtf sb0` and `mtf sb1` instructions which take the argument and add 32. Hence these instructions only modify the FPSCR[32:63] bits by changing the specified bit to a zero or one respectively.

The `__builtin_set_fpscr_rn` built-in allows changing both of the floating point rounding mode bits and returning the various FPSCR fields before the RN field is updated. The built-in returns a double consisting of the initial value of the FPSCR fields DRN, VE, OE, UE, ZE, XE, NI, and RN bit positions with all other bits set to zero. The built-in argument is a 2-bit value for the new RN field value. The argument can either be an `const int` or stored in a variable. Earlier versions of `__builtin_set_fpscr_rn` returned void. A `__SET_FPSCR_RN_RETURNS_FPSCR__` macro has been added. If defined, then the `__builtin_set_fpscr_rn` built-in returns the FPSCR fields. If not defined, the `__builtin_set_fpscr_rn` does not return a value. If the `-msoft-float` option is used, the `__builtin_set_fpscr_rn` built-in will not return a value.

7.13.23.2 Basic PowerPC Built-in Functions Available on ISA 2.05

The basic built-in functions described in this section are available on the PowerPC family of processors starting with ISA 2.05 or later. Unless specific options are explicitly disabled on the command line, specifying option `-mcpu=power6` has the effect of enabling the `-mpowerpc64`, `-mpowerpc-gpopt`, `-mpowerpc-gfxopt`, `-mmfcrf`, `-mpopcmtb`, `-mfprnd`, `-mcmpb`, `-mhard-dfp`, and `-mrecip-precision` options. Specify the `-maltivec` option explicitly in combination with the above options if desired.

The following functions require option `-mcmpb`.

```
unsigned long long __builtin_cmpb (unsigned long long int, unsigned long long int);
unsigned int __builtin_cmpb (unsigned int, unsigned int);
```

The `__builtin_cmpb` function performs a byte-wise compare on the contents of its two arguments, returning the result of the byte-wise comparison as the returned value. For each byte comparison, the corresponding byte of the return value holds 0xff if the input bytes are equal and 0 if the input bytes are not equal. If either of the arguments to this built-in function is wider than 32 bits, the function call expands into the form that expects `unsigned long long int` arguments which is only available on 64-bit targets.

The following built-in functions are available when hardware decimal floating point (`-mhard-dfp`) is available:

```
void __builtin_set_fpscr_drn(int);
_Decimal64 __builtin_ddedpd (int, _Decimal64);
_Decimal128 __builtin_ddedpdq (int, _Decimal128);
_Decimal64 __builtin_denbcd (int, _Decimal64);
_Decimal128 __builtin_denbcdq (int, _Decimal128);
_Decimal64 __builtin_diex (long long, _Decimal64);
_Decimal128 __builtin_diexq (long long, _Decimal128);
_Decimal64 __builtin_dscli (_Decimal64, int);
_Decimal128 __builtin_dscliq (_Decimal128, int);
_Decimal64 __builtin_dscri (_Decimal64, int);
_Decimal128 __builtin_dscriq (_Decimal128, int);
```

```
long long __builtin_dxex (_Decimal64);
long long __builtin_dxexq (_Decimal128);
_Decimal128 __builtin_pack_dec128 (unsigned long long, unsigned long long);
unsigned long long __builtin_unpack_dec128 (_Decimal128, int);
```

The `__builtin_set_fpscr_drn` builtin allows changing the three decimal floating point rounding mode bits. The argument is a 3-bit value. The argument can either be a `const int` or the value can be stored in a variable.

The builtin uses the ISA 3.0 instruction `mfscdrn` if available. Otherwise the builtin reads the FPSCR, masks the current decimal rounding mode bits out and OR's in the new value.

```
_Decimal64 __builtin_dfp_quantize (_Decimal64, _Decimal64, const int);
_Decimal64 __builtin_dfp_quantize (const int, _Decimal64, const int);
_Decimal128 __builtin_dfp_quantize (_Decimal128, _Decimal128, const int);
_Decimal128 __builtin_dfp_quantize (const int, _Decimal128, const int);
```

The `__builtin_dfp_quantize` built-in, converts and rounds the second argument to the form with the exponent as specified by the first argument based on the rounding mode specified by the third argument. If the first argument is a decimal floating point value, its exponent is used for converting and rounding of the second argument. If the first argument is a 5-bit constant integer value, then the value specifies the exponent to be used when rounding and converting the second argument. The third argument is a two bit constant integer that specifies the rounding mode. The possible modes are: 00 Round to nearest, ties to even; 01 Round toward 0; 10 Round to nearest, ties away from 0; 11 Round according to DRN where DRN is the Decimal Floating point field of the FPSCR.

The following functions require `-mhard-float`, `-mpowerpc-gfxopt`, and `-mpopcmtb` options.

```
double __builtin_recipdiv (double, double);
float __builtin_recipdivf (float, float);
double __builtin_rsqrtd (double);
float __builtin_rsqrtf (float);
```

The `vec_rsqrt`, `__builtin_rsqrt`, and `__builtin_rsqrtf` functions generate multiple instructions to implement the reciprocal sqrt functionality using reciprocal sqrt estimate instructions.

The `__builtin_recipdiv`, and `__builtin_recipdivf` functions generate multiple instructions to implement division using the reciprocal estimate instructions.

The following functions require `-mhard-float` and `-mmultiple` options.

The `__builtin_unpack_longdouble` function takes a long double argument and a compile time constant of 0 or 1. If the constant is 0, the first double within the long double is returned, otherwise the second double is returned. The `__builtin_unpack_longdouble` function is only available if long double uses the IBM extended double representation.

The `__builtin_pack_longdouble` function takes two double arguments and returns a long double value that combines the two arguments. The `__builtin_pack_longdouble` function is only available if long double uses the IBM extended double representation.

The `__builtin_unpack_ibm128` function takes a `__ibm128` argument and a compile time constant of 0 or 1. If the constant is 0, the first double within the `__ibm128` is returned, otherwise the second double is returned.

The `__builtin_pack_ibm128` function takes two double arguments and returns a `__ibm128` value that combines the two arguments.

Additional built-in functions are available for the 64-bit PowerPC family of processors, for efficient use of 128-bit floating point (`__float128`) values.

Vector select

```
vector signed __int128 vec_sel (vector signed __int128,
                                vector signed __int128, vector bool __int128);
vector signed __int128 vec_sel (vector signed __int128,
                                vector signed __int128, vector unsigned __int128);
vector unsigned __int128 vec_sel (vector unsigned __int128,
                                  vector unsigned __int128, vector bool __int128);
vector unsigned __int128 vec_sel (vector unsigned __int128,
                                  vector unsigned __int128, vector unsigned __int128);
vector bool __int128 vec_sel (vector bool __int128,
                              vector bool __int128, vector bool __int128);
vector bool __int128 vec_sel (vector bool __int128,
                              vector bool __int128, vector unsigned __int128);
```

The instance is an extension of the existing overloaded built-in `vec_sel` that is documented in the PVI PR.

```
vector signed __int128 vec_perm (vector signed __int128,
                                 vector signed __int128);
vector unsigned __int128 vec_perm (vector unsigned __int128,
                                   vector unsigned __int128);
```

The instance is an extension of the existing overloaded built-in `vec_perm` that is documented in the PVI PR.

7.13.23.3 Basic PowerPC Built-in Functions Available on ISA 2.06

The basic built-in functions described in this section are available on the PowerPC family of processors starting with ISA 2.05 or later. Unless specific options are explicitly disabled on the command line, specifying option `-mcpu=power7` has the effect of enabling all the same options as for `-mcpu=power6` in addition to the `-maltivec`, `-mpopcntd`, and `-mvsx` options.

The following basic built-in functions require `-mpopcntd`:

```
unsigned int __builtin_addg6s (unsigned int, unsigned int);
long long __builtin_bpermd (long long, long long);
unsigned int __builtin_cbcdd (unsigned int);
unsigned int __builtin_cdtbcd (unsigned int);
long long __builtin_divde (long long, long long);
unsigned long long __builtin_divdeu (unsigned long long, unsigned long long);
int __builtin_divwe (int, int);
unsigned int __builtin_divweu (unsigned int, unsigned int);
vector __int128 __builtin_pack_vector_int128 (long long, long long);
void __builtin_rs6000_speculation_barrier (void);
long long __builtin_unpack_vector_int128 (vector __int128, signed char);
```

Of these, the `__builtin_divde` and `__builtin_divdeu` functions require a 64-bit environment.

The following basic built-in functions, which are also supported on x86 targets, require `-mfloat128`.

```
__float128 __builtin_fabsq (__float128);
__float128 __builtin_copysignq (__float128, __float128);
__float128 __builtin_infq (void);
```

```

__float128 __builtin_huge_valq (void);
__float128 __builtin_nanq (void);
__float128 __builtin_nansq (void);

__float128 __builtin_sqrtf128 (__float128);
__float128 __builtin_fmaf128 (__float128, __float128, __float128);

```

7.13.23.4 Basic PowerPC Built-in Functions Available on ISA 2.07

The basic built-in functions described in this section are available on the PowerPC family of processors starting with ISA 2.07 or later. Unless specific options are explicitly disabled on the command line, specifying option `-mcpu=power8` has the effect of enabling all the same options as for `-mcpu=power7` in addition to the `-mpower8-fusion`, `-mcrypto`, `-mhtm`, `-mqquad-memory`, and `-mqquad-memory-atomic` options.

This section intentionally empty.

7.13.23.5 Basic PowerPC Built-in Functions Available on ISA 3.0

The basic built-in functions described in this section are available on the PowerPC family of processors starting with ISA 3.0 or later. Unless specific options are explicitly disabled on the command line, specifying option `-mcpu=power9` has the effect of enabling all the same options as for `-mcpu=power8` in addition to the `-misel` option.

The following built-in functions are available on Linux 64-bit systems that use the ISA 3.0 instruction set (`-mcpu=power9`):

```

__float128 __builtin_addf128_round_to_odd          [Built-in Function]
    (__float128, __float128)

```

Perform a 128-bit IEEE floating point add using round to odd as the rounding mode.

```

__float128 __builtin_subf128_round_to_odd          [Built-in Function]
    (__float128, __float128)

```

Perform a 128-bit IEEE floating point subtract using round to odd as the rounding mode.

```

__float128 __builtin_mulf128_round_to_odd          [Built-in Function]
    (__float128, __float128)

```

Perform a 128-bit IEEE floating point multiply using round to odd as the rounding mode.

```

__float128 __builtin_divf128_round_to_odd          [Built-in Function]
    (__float128, __float128)

```

Perform a 128-bit IEEE floating point divide using round to odd as the rounding mode.

```

__float128 __builtin_sqrtf128_round_to_odd        [Built-in Function]
    (__float128)

```

Perform a 128-bit IEEE floating point square root using round to odd as the rounding mode.

```
__float128 __builtin_fmaf128_round_to_odd          [Built-in Function]
    (__float128, __float128, __float128)
```

Perform a 128-bit IEEE floating point fused multiply and add operation using round to odd as the rounding mode.

```
double __builtin_truncf128_round_to_odd (__float128) [Built-in Function]
```

Convert a 128-bit IEEE floating point value to double using round to odd as the rounding mode.

The following additional built-in functions are also available for the PowerPC family of processors, starting with ISA 3.0 or later:

```
long long __builtin_darn (void)                    [Built-in Function]
```

```
long long __builtin_darn_raw (void)                [Built-in Function]
```

```
int __builtin_darn_32 (void)                       [Built-in Function]
```

The `__builtin_darn` and `__builtin_darn_raw` functions require a 64-bit environment supporting ISA 3.0 or later. The `__builtin_darn` function provides a 64-bit conditioned random number. The `__builtin_darn_raw` function provides a 64-bit raw random number. The `__builtin_darn_32` function provides a 32-bit conditioned random number.

The following additional built-in functions are also available for the PowerPC family of processors, starting with ISA 3.0 or later:

```
int __builtin_byte_in_set (unsigned char u, unsigned long long set);
int __builtin_byte_in_range (unsigned char u, unsigned int range);
int __builtin_byte_in_either_range (unsigned char u, unsigned int ranges);

int __builtin_dfp_dtstsfi_lt (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_lt (unsigned int comparison, _Decimal128 value);
int __builtin_dfp_dtstsfi_lt_dd (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_lt_td (unsigned int comparison, _Decimal128 value);

int __builtin_dfp_dtstsfi_gt (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_gt (unsigned int comparison, _Decimal128 value);
int __builtin_dfp_dtstsfi_gt_dd (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_gt_td (unsigned int comparison, _Decimal128 value);

int __builtin_dfp_dtstsfi_eq (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_eq (unsigned int comparison, _Decimal128 value);
int __builtin_dfp_dtstsfi_eq_dd (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_eq_td (unsigned int comparison, _Decimal128 value);

int __builtin_dfp_dtstsfi_ov (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_ov (unsigned int comparison, _Decimal128 value);
int __builtin_dfp_dtstsfi_ov_dd (unsigned int comparison, _Decimal64 value);
int __builtin_dfp_dtstsfi_ov_td (unsigned int comparison, _Decimal128 value);

double __builtin_mffsl(void);
```

The `__builtin_byte_in_set` function requires a 64-bit environment supporting ISA 3.0 or later. This function returns a non-zero value if and only if its `u` argument exactly equals one of the eight bytes contained within its 64-bit `set` argument.

The `__builtin_byte_in_range` and `__builtin_byte_in_either_range` require an environment supporting ISA 3.0 or later. For these two functions, the `range` argument is encoded as 4 bytes, organized as `hi_1:lo_1:hi_2:lo_2`. The `__builtin_byte_in_range` function returns a non-zero value if and only if its `u` argument is within the range bounded between `lo_2` and `hi_2` inclusive. The `__builtin_byte_in_either_range` function returns non-zero if and only if its `u` argument is within either the range bounded between `lo_1` and `hi_1` inclusive or the range bounded between `lo_2` and `hi_2` inclusive.

The `__builtin_dfp_dtstsfi_lt` function returns a non-zero value if and only if the number of significant digits of its `value` argument is less than its `comparison` argument. The `__builtin_dfp_dtstsfi_lt_dd` and `__builtin_dfp_dtstsfi_lt_td` functions behave similarly, but require that the type of the `value` argument be `__Decimal64` and `__Decimal128` respectively.

The `__builtin_dfp_dtstsfi_gt` function returns a non-zero value if and only if the number of significant digits of its `value` argument is greater than its `comparison` argument. The `__builtin_dfp_dtstsfi_gt_dd` and `__builtin_dfp_dtstsfi_gt_td` functions behave similarly, but require that the type of the `value` argument be `__Decimal64` and `__Decimal128` respectively.

The `__builtin_dfp_dtstsfi_eq` function returns a non-zero value if and only if the number of significant digits of its `value` argument equals its `comparison` argument. The `__builtin_dfp_dtstsfi_eq_dd` and `__builtin_dfp_dtstsfi_eq_td` functions behave similarly, but require that the type of the `value` argument be `__Decimal64` and `__Decimal128` respectively.

The `__builtin_dfp_dtstsfi_ov` function returns a non-zero value if and only if its `value` argument has an undefined number of significant digits, such as when `value` is an encoding of NaN. The `__builtin_dfp_dtstsfi_ov_dd` and `__builtin_dfp_dtstsfi_ov_td` functions behave similarly, but require that the type of the `value` argument be `__Decimal64` and `__Decimal128` respectively.

The `__builtin_mffsl` uses the ISA 3.0 `mffsl` instruction to read the FPSCR. The instruction is a lower latency version of the `mffs` instruction. If the `mffsl` instruction is not available, then the builtin uses the older `mffs` instruction to read the FPSCR.

7.13.23.6 Basic PowerPC Built-in Functions Available on ISA 3.1

The basic built-in functions described in this section are available on the PowerPC family of processors starting with ISA 3.1. Unless specific options are explicitly disabled on the command line, specifying option `-mcpu=power10` has the effect of enabling all the same options as for `-mcpu=power9`.

The following built-in functions are available on Linux 64-bit systems that use a future architecture instruction set (`-mcpu=power10`):

```
unsigned long long __builtin_cfuged (unsigned long long, unsigned long long) [Built-in Function]
```

Perform a 64-bit centrifuge operation, as if implemented by the `cfuged` instruction.

```
unsigned long long __builtin_cntlzdzm (unsigned long long, unsigned long long) [Built-in Function]
```

Perform a 64-bit count leading zeros operation under mask, as if implemented by the `cntlzdzm` instruction.

`unsigned long long __builtin_cnttzdm (unsigned long long, unsigned long long)` [Built-in Function]

Perform a 64-bit count trailing zeros operation under mask, as if implemented by the `cnttzdm` instruction.

`unsigned long long __builtin_pdepd (unsigned long long, unsigned long long)` [Built-in Function]

Perform a 64-bit parallel bits deposit operation, as if implemented by the `pdepd` instruction.

`unsigned long long __builtin_pextd (unsigned long long, unsigned long long)` [Built-in Function]

Perform a 64-bit parallel bits extract operation, as if implemented by the `pextd` instruction.

`vector signed __int128 vsx_xl_sext (signed long long, signed char *)` [Built-in Function]

`vector signed __int128 vsx_xl_sext (signed long long, signed short *)` [Built-in Function]

`vector signed __int128 vsx_xl_sext (signed long long, signed int *)` [Built-in Function]

`vector signed __int128 vsx_xl_sext (signed long long, signed long long *)` [Built-in Function]

`vector unsigned __int128 vsx_xl_zext (signed long long, unsigned char *)` [Built-in Function]

`vector unsigned __int128 vsx_xl_zext (signed long long, unsigned short *)` [Built-in Function]

`vector unsigned __int128 vsx_xl_zext (signed long long, unsigned int *)` [Built-in Function]

`vector unsigned __int128 vsx_xl_zext (signed long long, unsigned long long *)` [Built-in Function]

Load (and sign extend) to an `__int128` vector, as if implemented by the ISA 3.1 `lxvr bx, lxvr hx, lxvr wx, and lxvr dx` instructions.

`void vec_xst_trunc (vector signed __int128, signed long long, signed char *)` [Built-in Function]

`void vec_xst_trunc (vector signed __int128, signed long long, signed short *)` [Built-in Function]

`void vec_xst_trunc (vector signed __int128, signed long long, signed int *)` [Built-in Function]

`void vec_xst_trunc (vector signed __int128, signed long long, signed long long *)` [Built-in Function]

`void vec_xst_trunc (vector unsigned __int128, signed long long, unsigned char *)` [Built-in Function]

`void vec_xst_trunc (vector unsigned __int128, signed long long, unsigned short *)` [Built-in Function]

`void vec_xst_trunc (vector unsigned __int128, signed long long, unsigned int *)` [Built-in Function]

```
void vec_xst_trunc (vector unsigned __int128, signed [Built-in Function]
                  long long, unsigned long long *)
```

Truncate and store the rightmost element of a vector, as if implemented by the ISA 3.1 `stxvrxb`, `stxvrhx`, `stxvrwx`, and `stxvrdx` instructions.

7.13.24 PowerPC AltiVec/VSX Built-in Functions

GCC provides an interface for the PowerPC family of processors to access the AltiVec operations described in Motorola's AltiVec Programming Interface Manual. The interface is made available by including `<altivec.h>` and using `-maltivec` and `-mabi=altivec`. The interface supports the following vector types.

```
vector unsigned char
vector signed char
vector bool char

vector unsigned short
vector signed short
vector bool short
vector pixel

vector unsigned int
vector signed int
vector bool int
vector float
```

GCC's implementation of the high-level language interface available from C and C++ code differs from Motorola's documentation in several ways.

- A vector constant is a list of constant expressions within curly braces.
- A vector initializer requires no cast if the vector constant is of the same type as the variable it is initializing.
- If `signed` or `unsigned` is omitted, the signedness of the vector type is the default signedness of the base type. The default varies depending on the operating system, so a portable program should always specify the signedness.
- Compiling with `-maltivec` adds keywords `__vector`, `vector`, `__pixel`, `pixel`, `__bool` and `bool`. When compiling ISO C, the context-sensitive substitution of the keywords `vector`, `pixel` and `bool` is disabled. To use them, you must include `<altivec.h>` instead.
- GCC allows using a `typedef` name as the type specifier for a vector type, but only under the following circumstances:
 - When using `__vector` instead of `vector`; for example,


```
typedef signed short int16;
__vector int16 data;
```
 - When using `vector` in keyword-and-define mode; for example,


```
typedef signed short int16;
vector int16 data;
```

Note that keyword-and-define mode is enabled by disabling GNU extensions (e.g., by using `-std=c11`) and including `<altivec.h>`.

- For C, overloaded functions are implemented with macros so the following does not work:

```
vec_add ((vector signed int){1, 2, 3, 4}, foo);
```

Since `vec_add` is a macro, the vector constant in the example is treated as four separate arguments. Wrap the entire argument in parentheses for this to work.

Note: Only the `<altivec.h>` interface is supported. Internally, GCC uses built-in functions to achieve the functionality in the aforementioned header file, but they are not supported and are subject to change without notice.

GCC complies with the Power Vector Intrinsic Programming Reference (PVI PR), which may be found at https://openpowerfoundation.org/?resource_lib=power-vector-intrinsic-programming-reference. Chapter 4 of this document fully documents the vector API interfaces that must be provided by compliant compilers. Programmers should preferentially use the interfaces described therein. However, historically GCC has provided additional interfaces for access to vector instructions. These are briefly described below. Where the PVI PR provides a portable interface, other functions in GCC that provide the same capabilities should be considered deprecated.

The PVI PR documents the following overloaded functions:

<code>vec_abs</code>	<code>vec_absd</code>	<code>vec_abss</code>
<code>vec_add</code>	<code>vec_addc</code>	<code>vec_adde</code>
<code>vec_addec</code>	<code>vec_adds</code>	<code>vec_all_eq</code>
<code>vec_all_ge</code>	<code>vec_all_gt</code>	<code>vec_all_in</code>
<code>vec_all_le</code>	<code>vec_all_lt</code>	<code>vec_all_nan</code>
<code>vec_all_ne</code>	<code>vec_all_nge</code>	<code>vec_all_ngt</code>
<code>vec_all_nle</code>	<code>vec_all_nlt</code>	<code>vec_all_numeric</code>
<code>vec_and</code>	<code>vec_andc</code>	<code>vec_any_eq</code>
<code>vec_any_ge</code>	<code>vec_any_gt</code>	<code>vec_any_le</code>
<code>vec_any_lt</code>	<code>vec_any_nan</code>	<code>vec_any_ne</code>
<code>vec_any_nge</code>	<code>vec_any_ngt</code>	<code>vec_any_nle</code>
<code>vec_any_nlt</code>	<code>vec_any_numeric</code>	<code>vec_any_out</code>
<code>vec_avg</code>	<code>vec_bperm</code>	<code>vec_ceil</code>
<code>vec_cipher_be</code>	<code>vec_cipherlast_be</code>	<code>vec_cmpb</code>
<code>vec_cmpeq</code>	<code>vec_cmpge</code>	<code>vec_cmpgt</code>
<code>vec_cmple</code>	<code>vec_cmplt</code>	<code>vec_cmpne</code>
<code>vec_cmpnez</code>	<code>vec_cntlz</code>	<code>vec_cntlz_lsbb</code>
<code>vec_cnttz</code>	<code>vec_cnttz_lsbb</code>	<code>vec_cpsgn</code>
<code>vec_ctf</code>	<code>vec_cts</code>	<code>vec_ctu</code>
<code>vec_div</code>	<code>vec_double</code>	<code>vec_doublee</code>
<code>vec_doubleh</code>	<code>vec_doublel</code>	<code>vec_doubleo</code>
<code>vec_eqv</code>	<code>vec_expte</code>	<code>vec_extract</code>
<code>vec_extract_exp</code>	<code>vec_extract_fp32_from_</code> <code>shortl</code>	<code>vec_extract_fp32_from_</code> <code>shortl</code>
<code>vec_extract_sig</code>	<code>vec_extract_4b</code>	<code>vec_first_match_index</code>
<code>vec_first_match_or_eos_</code> <code>index</code>	<code>vec_first_mismatch_</code> <code>index</code>	<code>vec_first_mismatch_or_</code> <code>eos_index</code>
<code>vec_float</code>	<code>vec_float2</code>	<code>vec_floate</code>
<code>vec_floato</code>	<code>vec_floor</code>	<code>vec_gb</code>
<code>vec_insert</code>	<code>vec_insert_exp</code>	<code>vec_insert4b</code>

<code>vec_ld</code>	<code>vec_lde</code>	<code>vec_ldl</code>
<code>vec_loge</code>	<code>vec_madd</code>	<code>vec_madds</code>
<code>vec_max</code>	<code>vec_mergee</code>	<code>vec_mergeh</code>
<code>vec_mergel</code>	<code>vec_mergeo</code>	<code>vec_mfvscr</code>
<code>vec_min</code>	<code>vec_mradds</code>	<code>vec_msub</code>
<code>vec_msum</code>	<code>vec_msums</code>	<code>vec_mtvscr</code>
<code>vec_mul</code>	<code>vec_mule</code>	<code>vec_mulo</code>
<code>vec_nabs</code>	<code>vec_nand</code>	<code>vec_ncipher_be</code>
<code>vec_ncipherlast_be</code>	<code>vec_nearbyint</code>	<code>vec_neg</code>
<code>vec_nmadd</code>	<code>vec_nmsub</code>	<code>vec_nor</code>
<code>vec_or</code>	<code>vec_orc</code>	<code>vec_pack</code>
<code>vec_pack_to_short_fp32</code>	<code>vec_packpx</code>	<code>vec_packs</code>
<code>vec_packsu</code>	<code>vec_parity_lsbb</code>	<code>vec_perm</code>
<code>vec_permxor</code>	<code>vec_pmsum_be</code>	<code>vec_popcnt</code>
<code>vec_re</code>	<code>vec_recipdiv</code>	<code>vec_revb</code>
<code>vec_reve</code>	<code>vec_rint</code>	<code>vec_rl</code>
<code>vec_rlmi</code>	<code>vec_rlnm</code>	<code>vec_round</code>
<code>vec_rsqrt</code>	<code>vec_rsqrtc</code>	<code>vec_sbox_be</code>
<code>vec_sel</code>	<code>vec_shasigma_be</code>	<code>vec_signed</code>
<code>vec_signed2</code>	<code>vec_signede</code>	<code>vec_signedo</code>
<code>vec_sl</code>	<code>vec_sld</code>	<code>vec_sldw</code>
<code>vec_sll</code>	<code>vec_slo</code>	<code>vec_slv</code>
<code>vec_splat</code>	<code>vec_splat_s8</code>	<code>vec_splat_s16</code>
<code>vec_splat_s32</code>	<code>vec_splat_u8</code>	<code>vec_splat_u16</code>
<code>vec_splat_u32</code>	<code>vec_splats</code>	<code>vec_sqrt</code>
<code>vec_sr</code>	<code>vec_sra</code>	<code>vec_srl</code>
<code>vec_sro</code>	<code>vec_srv</code>	<code>vec_st</code>
<code>vec_ste</code>	<code>vec_stl</code>	<code>vec_sub</code>
<code>vec_subc</code>	<code>vec_sube</code>	<code>vec_subec</code>
<code>vec_subs</code>	<code>vec_sum2s</code>	<code>vec_sum4s</code>
<code>vec_sums</code>	<code>vec_test_data_class</code>	<code>vec_trunc</code>
<code>vec_unpackh</code>	<code>vec_unpackl</code>	<code>vec_unsigned</code>
<code>vec_unsigned2</code>	<code>vec_unsignede</code>	<code>vec_unsignedo</code>
<code>vec_xl</code>	<code>vec_xl_be</code>	<code>vec_xl_len</code>
<code>vec_xl_len_r</code>	<code>vec_xor</code>	<code>vec_xst</code>
<code>vec_xst_be</code>	<code>vec_xst_len</code>	<code>vec_xst_len_r</code>

7.13.24.1 PowerPC AltiVec Built-in Functions on ISA 2.05

The following interfaces are supported for the generic and specific AltiVec operations and the AltiVec predicates. In cases where there is a direct mapping between generic and specific operations, only the generic names are shown here, although the specific operations can also be used.

Arguments that are documented as `const int` require literal integral values within the range required for that operation.

Only functions excluded from the PVI PR are listed here.

```

void vec_dss (const int);

void vec_dssall (void);

void vec_dst (const vector unsigned char *, int, const int);
void vec_dst (const vector signed char *, int, const int);
void vec_dst (const vector bool char *, int, const int);
void vec_dst (const vector unsigned short *, int, const int);
void vec_dst (const vector signed short *, int, const int);
void vec_dst (const vector bool short *, int, const int);
void vec_dst (const vector pixel *, int, const int);
void vec_dst (const vector unsigned int *, int, const int);
void vec_dst (const vector signed int *, int, const int);
void vec_dst (const vector bool int *, int, const int);
void vec_dst (const vector float *, int, const int);
void vec_dst (const unsigned char *, int, const int);
void vec_dst (const signed char *, int, const int);
void vec_dst (const unsigned short *, int, const int);
void vec_dst (const short *, int, const int);
void vec_dst (const unsigned int *, int, const int);
void vec_dst (const int *, int, const int);
void vec_dst (const float *, int, const int);

void vec_dstst (const vector unsigned char *, int, const int);
void vec_dstst (const vector signed char *, int, const int);
void vec_dstst (const vector bool char *, int, const int);
void vec_dstst (const vector unsigned short *, int, const int);
void vec_dstst (const vector signed short *, int, const int);
void vec_dstst (const vector bool short *, int, const int);
void vec_dstst (const vector pixel *, int, const int);
void vec_dstst (const vector unsigned int *, int, const int);
void vec_dstst (const vector signed int *, int, const int);
void vec_dstst (const vector bool int *, int, const int);
void vec_dstst (const vector float *, int, const int);
void vec_dstst (const unsigned char *, int, const int);
void vec_dstst (const signed char *, int, const int);
void vec_dstst (const unsigned short *, int, const int);
void vec_dstst (const short *, int, const int);
void vec_dstst (const unsigned int *, int, const int);
void vec_dstst (const int *, int, const int);
void vec_dstst (const unsigned long *, int, const int);
void vec_dstst (const long *, int, const int);
void vec_dstst (const float *, int, const int);

void vec_dststt (const vector unsigned char *, int, const int);
void vec_dststt (const vector signed char *, int, const int);
void vec_dststt (const vector bool char *, int, const int);
void vec_dststt (const vector unsigned short *, int, const int);
void vec_dststt (const vector signed short *, int, const int);
void vec_dststt (const vector bool short *, int, const int);
void vec_dststt (const vector pixel *, int, const int);
void vec_dststt (const vector unsigned int *, int, const int);
void vec_dststt (const vector signed int *, int, const int);
void vec_dststt (const vector bool int *, int, const int);
void vec_dststt (const vector float *, int, const int);
void vec_dststt (const unsigned char *, int, const int);
void vec_dststt (const signed char *, int, const int);
void vec_dststt (const unsigned short *, int, const int);

```

```

void vec_dststt (const short *, int, const int);
void vec_dststt (const unsigned int *, int, const int);
void vec_dststt (const int *, int, const int);
void vec_dststt (const float *, int, const int);

void vec_dstt (const vector unsigned char *, int, const int);
void vec_dstt (const vector signed char *, int, const int);
void vec_dstt (const vector bool char *, int, const int);
void vec_dstt (const vector unsigned short *, int, const int);
void vec_dstt (const vector signed short *, int, const int);
void vec_dstt (const vector bool short *, int, const int);
void vec_dstt (const vector pixel *, int, const int);
void vec_dstt (const vector unsigned int *, int, const int);
void vec_dstt (const vector signed int *, int, const int);
void vec_dstt (const vector bool int *, int, const int);
void vec_dstt (const vector float *, int, const int);
void vec_dstt (const vector unsigned char *, int, const int);
void vec_dstt (const vector signed char *, int, const int);
void vec_dstt (const vector unsigned short *, int, const int);
void vec_dstt (const vector signed short *, int, const int);
void vec_dstt (const vector bool short *, int, const int);
void vec_dstt (const vector bool int *, int, const int);
void vec_dstt (const vector float *, int, const int);

vector signed char vec_lvebx (int, char *);
vector unsigned char vec_lvebx (int, unsigned char *);

vector signed short vec_lvehx (int, short *);
vector unsigned short vec_lvehx (int, unsigned short *);

vector float vec_lviewx (int, float *);
vector signed int vec_lviewx (int, int *);
vector unsigned int vec_lviewx (int, unsigned int *);

vector unsigned char vec_lvsl (int, const unsigned char *);
vector unsigned char vec_lvsl (int, const signed char *);
vector unsigned char vec_lvsl (int, const unsigned short *);
vector unsigned char vec_lvsl (int, const short *);
vector unsigned char vec_lvsl (int, const unsigned int *);
vector unsigned char vec_lvsl (int, const int *);
vector unsigned char vec_lvsl (int, const float *);

vector unsigned char vec_lvsl (int, const unsigned char *);
vector unsigned char vec_lvsl (int, const signed char *);
vector unsigned char vec_lvsl (int, const unsigned short *);
vector unsigned char vec_lvsl (int, const short *);
vector unsigned char vec_lvsl (int, const unsigned int *);
vector unsigned char vec_lvsl (int, const int *);
vector unsigned char vec_lvsl (int, const float *);

void vec_stvebx (vector signed char, int, signed char *);
void vec_stvebx (vector unsigned char, int, unsigned char *);
void vec_stvebx (vector bool char, int, signed char *);
void vec_stvebx (vector bool char, int, unsigned char *);

void vec_stvehx (vector signed short, int, short *);
void vec_stvehx (vector unsigned short, int, unsigned short *);
void vec_stvehx (vector bool short, int, short *);

```

```

void vec_stvehx (vector bool short, int, unsigned short *);

void vec_stviewx (vector float, int, float *);
void vec_stviewx (vector signed int, int, int *);
void vec_stviewx (vector unsigned int, int, unsigned int *);
void vec_stviewx (vector bool int, int, int *);
void vec_stviewx (vector bool int, int, unsigned int *);

vector float vec_vaddfp (vector float, vector float);

vector signed char vec_vaddsbs (vector bool char, vector signed char);
vector signed char vec_vaddsbs (vector signed char, vector bool char);
vector signed char vec_vaddsbs (vector signed char, vector signed char);

vector signed short vec_vaddshs (vector bool short, vector signed short);
vector signed short vec_vaddshs (vector signed short, vector bool short);
vector signed short vec_vaddshs (vector signed short, vector signed short);

vector signed int vec_vaddsws (vector bool int, vector signed int);
vector signed int vec_vaddsws (vector signed int, vector bool int);
vector signed int vec_vaddsws (vector signed int, vector signed int);

vector signed char vec_vaddubm (vector bool char, vector signed char);
vector signed char vec_vaddubm (vector signed char, vector bool char);
vector signed char vec_vaddubm (vector signed char, vector signed char);
vector unsigned char vec_vaddubm (vector bool char, vector unsigned char);
vector unsigned char vec_vaddubm (vector unsigned char, vector bool char);
vector unsigned char vec_vaddubm (vector unsigned char, vector unsigned char);

vector unsigned char vec_vaddubs (vector bool char, vector unsigned char);
vector unsigned char vec_vaddubs (vector unsigned char, vector bool char);
vector unsigned char vec_vaddubs (vector unsigned char, vector unsigned char);

vector signed short vec_vadduhm (vector bool short, vector signed short);
vector signed short vec_vadduhm (vector signed short, vector bool short);
vector signed short vec_vadduhm (vector signed short, vector signed short);
vector unsigned short vec_vadduhm (vector bool short, vector unsigned short);
vector unsigned short vec_vadduhm (vector unsigned short, vector bool short);
vector unsigned short vec_vadduhm (vector unsigned short, vector unsigned short);

vector unsigned short vec_vadduhs (vector bool short, vector unsigned short);
vector unsigned short vec_vadduhs (vector unsigned short, vector bool short);
vector unsigned short vec_vadduhs (vector unsigned short, vector unsigned short);

vector signed int vec_vadduwm (vector bool int, vector signed int);
vector signed int vec_vadduwm (vector signed int, vector bool int);
vector signed int vec_vadduwm (vector signed int, vector signed int);
vector unsigned int vec_vadduwm (vector bool int, vector unsigned int);
vector unsigned int vec_vadduwm (vector unsigned int, vector bool int);
vector unsigned int vec_vadduwm (vector unsigned int, vector unsigned int);

vector unsigned int vec_vadduws (vector bool int, vector unsigned int);
vector unsigned int vec_vadduws (vector unsigned int, vector bool int);
vector unsigned int vec_vadduws (vector unsigned int, vector unsigned int);

vector signed char vec_vavgsh (vector signed char, vector signed char);

vector signed short vec_vavgsh (vector signed short, vector signed short);

```

```

vector signed int vec_vavgsw (vector signed int, vector signed int);

vector unsigned char vec_vavgub (vector unsigned char, vector unsigned char);

vector unsigned short vec_vavguh (vector unsigned short, vector unsigned short);

vector unsigned int vec_vavguw (vector unsigned int, vector unsigned int);

vector float vec_vcfsx (vector signed int, const int);

vector float vec_vcfux (vector unsigned int, const int);

vector bool int vec_vcmpeqfp (vector float, vector float);

vector bool char vec_vcmpequb (vector signed char, vector signed char);
vector bool char vec_vcmpequb (vector unsigned char, vector unsigned char);

vector bool short vec_vcmpequh (vector signed short, vector signed short);
vector bool short vec_vcmpequh (vector unsigned short, vector unsigned short);

vector bool int vec_vcmpequw (vector signed int, vector signed int);
vector bool int vec_vcmpequw (vector unsigned int, vector unsigned int);

vector bool int vec_vcmpgtfp (vector float, vector float);

vector bool char vec_vcmpgtfb (vector signed char, vector signed char);

vector bool short vec_vcmpgtsh (vector signed short, vector signed short);

vector bool int vec_vcmpgtsw (vector signed int, vector signed int);

vector bool char vec_vcmpgtub (vector unsigned char, vector unsigned char);

vector bool short vec_vcmpgtuh (vector unsigned short, vector unsigned short);

vector bool int vec_vcmpgtuw (vector unsigned int, vector unsigned int);

vector float vec_vmaxfp (vector float, vector float);

vector signed char vec_vmaxsb (vector bool char, vector signed char);
vector signed char vec_vmaxsb (vector signed char, vector bool char);
vector signed char vec_vmaxsb (vector signed char, vector signed char);

vector signed short vec_vmaxsh (vector bool short, vector signed short);
vector signed short vec_vmaxsh (vector signed short, vector bool short);
vector signed short vec_vmaxsh (vector signed short, vector signed short);

vector signed int vec_vmaxsw (vector bool int, vector signed int);
vector signed int vec_vmaxsw (vector signed int, vector bool int);
vector signed int vec_vmaxsw (vector signed int, vector signed int);

vector unsigned char vec_vmaxub (vector bool char, vector unsigned char);
vector unsigned char vec_vmaxub (vector unsigned char, vector bool char);
vector unsigned char vec_vmaxub (vector unsigned char, vector unsigned char);

vector unsigned short vec_vmaxuh (vector bool short, vector unsigned short);
vector unsigned short vec_vmaxuh (vector unsigned short, vector bool short);

```

```

vector unsigned short vec_vmaxuh (vector unsigned short, vector unsigned short);

vector unsigned int vec_vmaxuw (vector bool int, vector unsigned int);
vector unsigned int vec_vmaxuw (vector unsigned int, vector bool int);
vector unsigned int vec_vmaxuw (vector unsigned int, vector unsigned int);

vector float vec_vminfp (vector float, vector float);

vector signed char vec_vminsb (vector bool char, vector signed char);
vector signed char vec_vminsb (vector signed char, vector bool char);
vector signed char vec_vminsb (vector signed char, vector signed char);

vector signed short vec_vminsh (vector bool short, vector signed short);
vector signed short vec_vminsh (vector signed short, vector bool short);
vector signed short vec_vminsh (vector signed short, vector signed short);

vector signed int vec_vminsw (vector bool int, vector signed int);
vector signed int vec_vminsw (vector signed int, vector bool int);
vector signed int vec_vminsw (vector signed int, vector signed int);

vector unsigned char vec_vminub (vector bool char, vector unsigned char);
vector unsigned char vec_vminub (vector unsigned char, vector bool char);
vector unsigned char vec_vminub (vector unsigned char, vector unsigned char);

vector unsigned short vec_vminuh (vector bool short, vector unsigned short);
vector unsigned short vec_vminuh (vector unsigned short, vector bool short);
vector unsigned short vec_vminuh (vector unsigned short, vector unsigned short);

vector unsigned int vec_vminuw (vector bool int, vector unsigned int);
vector unsigned int vec_vminuw (vector unsigned int, vector bool int);
vector unsigned int vec_vminuw (vector unsigned int, vector unsigned int);

vector bool char vec_vmrghb (vector bool char, vector bool char);
vector signed char vec_vmrghb (vector signed char, vector signed char);
vector unsigned char vec_vmrghb (vector unsigned char, vector unsigned char);

vector bool short vec_vmrghh (vector bool short, vector bool short);
vector signed short vec_vmrghh (vector signed short, vector signed short);
vector unsigned short vec_vmrghh (vector unsigned short, vector unsigned short);
vector pixel vec_vmrghh (vector pixel, vector pixel);

vector float vec_vmrghw (vector float, vector float);
vector bool int vec_vmrghw (vector bool int, vector bool int);
vector signed int vec_vmrghw (vector signed int, vector signed int);
vector unsigned int vec_vmrghw (vector unsigned int, vector unsigned int);

vector bool char vec_vmrglb (vector bool char, vector bool char);
vector signed char vec_vmrglb (vector signed char, vector signed char);
vector unsigned char vec_vmrglb (vector unsigned char, vector unsigned char);

vector bool short vec_vmrglh (vector bool short, vector bool short);
vector signed short vec_vmrglh (vector signed short, vector signed short);
vector unsigned short vec_vmrglh (vector unsigned short, vector unsigned short);
vector pixel vec_vmrglh (vector pixel, vector pixel);

vector float vec_vmrglw (vector float, vector float);
vector signed int vec_vmrglw (vector signed int, vector signed int);
vector unsigned int vec_vmrglw (vector unsigned int, vector unsigned int);

```

```

vector bool int vec_vmrglw (vector bool int, vector bool int);

vector signed int vec_vmsummbm (vector signed char, vector unsigned char,
                                vector signed int);

vector signed int vec_vmsumshm (vector signed short, vector signed short,
                                vector signed int);

vector signed int vec_vmsumshs (vector signed short, vector signed short,
                                vector signed int);

vector unsigned int vec_vmsumubm (vector unsigned char, vector unsigned char,
                                vector unsigned int);

vector unsigned int vec_vmsumuhm (vector unsigned short, vector unsigned short,
                                vector unsigned int);

vector unsigned int vec_vmsumuhs (vector unsigned short, vector unsigned short,
                                vector unsigned int);

vector signed short vec_vmulesb (vector signed char, vector signed char);

vector signed int vec_vmulesh (vector signed short, vector signed short);

vector unsigned short vec_vmuleub (vector unsigned char, vector unsigned char);

vector unsigned int vec_vmuleuh (vector unsigned short, vector unsigned short);

vector signed short vec_vmulosb (vector signed char, vector signed char);

vector signed int vec_vmulosh (vector signed short, vector signed short);

vector unsigned short vec_vmuloub (vector unsigned char, vector unsigned char);

vector unsigned int vec_vmulouh (vector unsigned short, vector unsigned short);

vector signed char vec_vpkshss (vector signed short, vector signed short);

vector unsigned char vec_vpkshus (vector signed short, vector signed short);

vector signed short vec_vpkswss (vector signed int, vector signed int);

vector unsigned short vec_vpkswus (vector signed int, vector signed int);

vector bool char vec_vpkuhum (vector bool short, vector bool short);
vector signed char vec_vpkuhum (vector signed short, vector signed short);
vector unsigned char vec_vpkuhum (vector unsigned short, vector unsigned short);

vector unsigned char vec_vpkuhus (vector unsigned short, vector unsigned short);

vector bool short vec_vpkuwum (vector bool int, vector bool int);
vector signed short vec_vpkuwum (vector signed int, vector signed int);
vector unsigned short vec_vpkuwum (vector unsigned int, vector unsigned int);

vector unsigned short vec_vpkuwus (vector unsigned int, vector unsigned int);

vector signed char vec_vrlb (vector signed char, vector unsigned char);
vector unsigned char vec_vrlb (vector unsigned char, vector unsigned char);

```

```

vector signed short vec_vrlh (vector signed short, vector unsigned short);
vector unsigned short vec_vrlh (vector unsigned short, vector unsigned short);

vector signed int vec_vrlw (vector signed int, vector unsigned int);
vector unsigned int vec_vrlw (vector unsigned int, vector unsigned int);

vector signed char vec_vslb (vector signed char, vector unsigned char);
vector unsigned char vec_vslb (vector unsigned char, vector unsigned char);

vector signed short vec_vslh (vector signed short, vector unsigned short);
vector unsigned short vec_vslh (vector unsigned short, vector unsigned short);

vector signed int vec_vslw (vector signed int, vector unsigned int);
vector unsigned int vec_vslw (vector unsigned int, vector unsigned int);

vector signed char vec_vspltb (vector signed char, const int);
vector unsigned char vec_vspltb (vector unsigned char, const int);
vector bool char vec_vspltb (vector bool char, const int);

vector bool short vec_vsplth (vector bool short, const int);
vector signed short vec_vsplth (vector signed short, const int);
vector unsigned short vec_vsplth (vector unsigned short, const int);
vector pixel vec_vsplth (vector pixel, const int);

vector float vec_vspltw (vector float, const int);
vector signed int vec_vspltw (vector signed int, const int);
vector unsigned int vec_vspltw (vector unsigned int, const int);
vector bool int vec_vspltw (vector bool int, const int);

vector signed char vec_vsrab (vector signed char, vector unsigned char);
vector unsigned char vec_vsrab (vector unsigned char, vector unsigned char);

vector signed short vec_vsrh (vector signed short, vector unsigned short);
vector unsigned short vec_vsrh (vector unsigned short, vector unsigned short);

vector signed int vec_vsrw (vector signed int, vector unsigned int);
vector unsigned int vec_vsrw (vector unsigned int, vector unsigned int);

vector signed char vec_vsrb (vector signed char, vector unsigned char);
vector unsigned char vec_vsrb (vector unsigned char, vector unsigned char);

vector signed short vec_vsrh (vector signed short, vector unsigned short);
vector unsigned short vec_vsrh (vector unsigned short, vector unsigned short);

vector signed int vec_vsrw (vector signed int, vector unsigned int);
vector unsigned int vec_vsrw (vector unsigned int, vector unsigned int);

vector float vec_vsubfp (vector float, vector float);

vector signed char vec_vsubsbs (vector bool char, vector signed char);
vector signed char vec_vsubsbs (vector signed char, vector bool char);
vector signed char vec_vsubsbs (vector signed char, vector signed char);

vector signed short vec_vsubshs (vector bool short, vector signed short);
vector signed short vec_vsubshs (vector signed short, vector bool short);
vector signed short vec_vsubshs (vector signed short, vector signed short);

```

```

vector signed int vec_vsubsws (vector bool int, vector signed int);
vector signed int vec_vsubsws (vector signed int, vector bool int);
vector signed int vec_vsubsws (vector signed int, vector signed int);

vector signed char vec_vsububm (vector bool char, vector signed char);
vector signed char vec_vsububm (vector signed char, vector bool char);
vector signed char vec_vsububm (vector signed char, vector signed char);
vector unsigned char vec_vsububm (vector bool char, vector unsigned char);
vector unsigned char vec_vsububm (vector unsigned char, vector bool char);
vector unsigned char vec_vsububm (vector unsigned char, vector unsigned char);

vector unsigned char vec_vsububs (vector bool char, vector unsigned char);
vector unsigned char vec_vsububs (vector unsigned char, vector bool char);
vector unsigned char vec_vsububs (vector unsigned char, vector unsigned char);

vector signed short vec_vsubuhm (vector bool short, vector signed short);
vector signed short vec_vsubuhm (vector signed short, vector bool short);
vector signed short vec_vsubuhm (vector signed short, vector signed short);
vector unsigned short vec_vsubuhm (vector bool short, vector unsigned short);
vector unsigned short vec_vsubuhm (vector unsigned short, vector bool short);
vector unsigned short vec_vsubuhm (vector unsigned short, vector unsigned short);

vector unsigned short vec_vsubuhs (vector bool short, vector unsigned short);
vector unsigned short vec_vsubuhs (vector unsigned short, vector bool short);
vector unsigned short vec_vsubuhs (vector unsigned short, vector unsigned short);

vector signed int vec_vsubuwm (vector bool int, vector signed int);
vector signed int vec_vsubuwm (vector signed int, vector bool int);
vector signed int vec_vsubuwm (vector signed int, vector signed int);
vector unsigned int vec_vsubuwm (vector bool int, vector unsigned int);
vector unsigned int vec_vsubuwm (vector unsigned int, vector bool int);
vector unsigned int vec_vsubuwm (vector unsigned int, vector unsigned int);

vector unsigned int vec_vsubuws (vector bool int, vector unsigned int);
vector unsigned int vec_vsubuws (vector unsigned int, vector bool int);
vector unsigned int vec_vsubuws (vector unsigned int, vector unsigned int);

vector signed int vec_vsum4sbs (vector signed char, vector signed int);

vector signed int vec_vsum4shs (vector signed short, vector signed int);

vector unsigned int vec_vsum4ubs (vector unsigned char, vector unsigned int);

vector unsigned int vec_vupkhp (vector pixel);

vector bool short vec_vupkhsb (vector bool char);
vector signed short vec_vupkhsb (vector signed char);

vector bool int vec_vupkhsh (vector bool short);
vector signed int vec_vupkhsh (vector signed short);

vector unsigned int vec_vupklpx (vector pixel);

vector bool short vec_vupklsb (vector bool char);
vector signed short vec_vupklsb (vector signed char);

vector bool int vec_vupklsh (vector bool short);
vector signed int vec_vupklsh (vector signed short);

```

7.13.24.2 PowerPC AltiVec Built-in Functions Available on ISA 2.06

The AltiVec built-in functions described in this section are available on the PowerPC family of processors starting with ISA 2.06 or later. These are normally enabled by adding `-mvsx` to the command line.

When `-mvsx` is used, the following additional vector types are implemented.

```
vector unsigned __int128
vector signed __int128
vector unsigned long long int
vector signed long long int
vector double
```

The long long types are only implemented for 64-bit code generation.

Only functions excluded from the PVIPR are listed here.

```
void vec_dst (const unsigned long *, int, const int);
void vec_dst (const long *, int, const int);

void vec_dststt (const unsigned long *, int, const int);
void vec_dststt (const long *, int, const int);

void vec_dstt (const unsigned long *, int, const int);
void vec_dstt (const long *, int, const int);

vector unsigned char vec_lvsl (int, const unsigned long *);
vector unsigned char vec_lvsl (int, const long *);

vector unsigned char vec_lvsr (int, const unsigned long *);
vector unsigned char vec_lvsr (int, const long *);

vector unsigned char vec_lvsl (int, const double *);
vector unsigned char vec_lvsr (int, const double *);

vector double vec_vsx_ld (int, const vector double *);
vector double vec_vsx_ld (int, const double *);
vector float vec_vsx_ld (int, const vector float *);
vector float vec_vsx_ld (int, const float *);
vector bool int vec_vsx_ld (int, const vector bool int *);
vector signed int vec_vsx_ld (int, const vector signed int *);
vector signed int vec_vsx_ld (int, const int *);
vector signed int vec_vsx_ld (int, const long *);
vector unsigned int vec_vsx_ld (int, const vector unsigned int *);
vector unsigned int vec_vsx_ld (int, const unsigned int *);
vector unsigned int vec_vsx_ld (int, const unsigned long *);
vector bool short vec_vsx_ld (int, const vector bool short *);
vector pixel vec_vsx_ld (int, const vector pixel *);
vector signed short vec_vsx_ld (int, const vector signed short *);
vector signed short vec_vsx_ld (int, const short *);
vector unsigned short vec_vsx_ld (int, const vector unsigned short *);
vector unsigned short vec_vsx_ld (int, const unsigned short *);
vector bool char vec_vsx_ld (int, const vector bool char *);
vector signed char vec_vsx_ld (int, const vector signed char *);
vector signed char vec_vsx_ld (int, const signed char *);
vector unsigned char vec_vsx_ld (int, const vector unsigned char *);
vector unsigned char vec_vsx_ld (int, const unsigned char *);

void vec_vsx_st (vector double, int, vector double *);
void vec_vsx_st (vector double, int, double *);
```

```

void vec_vsx_st (vector float, int, vector float *);
void vec_vsx_st (vector float, int, float *);
void vec_vsx_st (vector signed int, int, vector signed int *);
void vec_vsx_st (vector signed int, int, int *);
void vec_vsx_st (vector unsigned int, int, vector unsigned int *);
void vec_vsx_st (vector unsigned int, int, unsigned int *);
void vec_vsx_st (vector bool int, int, vector bool int *);
void vec_vsx_st (vector bool int, int, unsigned int *);
void vec_vsx_st (vector bool int, int, int *);
void vec_vsx_st (vector signed short, int, vector signed short *);
void vec_vsx_st (vector signed short, int, short *);
void vec_vsx_st (vector unsigned short, int, vector unsigned short *);
void vec_vsx_st (vector unsigned short, int, unsigned short *);
void vec_vsx_st (vector bool short, int, vector bool short *);
void vec_vsx_st (vector bool short, int, unsigned short *);
void vec_vsx_st (vector pixel, int, vector pixel *);
void vec_vsx_st (vector pixel, int, unsigned short *);
void vec_vsx_st (vector pixel, int, short *);
void vec_vsx_st (vector bool short, int, short *);
void vec_vsx_st (vector signed char, int, vector signed char *);
void vec_vsx_st (vector signed char, int, signed char *);
void vec_vsx_st (vector unsigned char, int, vector unsigned char *);
void vec_vsx_st (vector unsigned char, int, unsigned char *);
void vec_vsx_st (vector bool char, int, vector bool char *);
void vec_vsx_st (vector bool char, int, unsigned char *);
void vec_vsx_st (vector bool char, int, signed char *);

vector double vec_xxpermdi (vector double, vector double, const int);
vector float vec_xxpermdi (vector float, vector float, const int);
vector __int128 vec_xxpermdi (vector __int128,
                             vector __int128, const int);
vector __uint128 vec_xxpermdi (vector __uint128,
                              vector __uint128, const int);
vector long long vec_xxpermdi (vector long long, vector long long, const int);
vector unsigned long long vec_xxpermdi (vector unsigned long long,
                                         vector unsigned long long, const int);
vector int vec_xxpermdi (vector int, vector int, const int);
vector unsigned int vec_xxpermdi (vector unsigned int,
                                  vector unsigned int, const int);
vector short vec_xxpermdi (vector short, vector short, const int);
vector unsigned short vec_xxpermdi (vector unsigned short,
                                    vector unsigned short, const int);
vector signed char vec_xxpermdi (vector signed char, vector signed char,
                                const int);
vector unsigned char vec_xxpermdi (vector unsigned char,
                                   vector unsigned char, const int);

vector double vec_xxsldi (vector double, vector double, int);
vector float vec_xxsldi (vector float, vector float, int);
vector long long vec_xxsldi (vector long long, vector long long, int);
vector unsigned long long vec_xxsldi (vector unsigned long long,
                                       vector unsigned long long, int);
vector int vec_xxsldi (vector int, vector int, int);
vector unsigned int vec_xxsldi (vector unsigned int, vector unsigned int, int);
vector short vec_xxsldi (vector short, vector short, int);
vector unsigned short vec_xxsldi (vector unsigned short,
                                  vector unsigned short, int);
vector signed char vec_xxsldi (vector signed char, vector signed char, int);

```

```
vector unsigned char vec_xxsldi (vector unsigned char,
                                vector unsigned char, int);
```

Note that the ‘`vec_ld`’ and ‘`vec_st`’ built-in functions always generate the AltiVec ‘`LVX`’ and ‘`STVX`’ instructions even if the VSX instruction set is available. The ‘`vec_vsx_ld`’ and ‘`vec_vsx_st`’ built-in functions always generate the VSX ‘`LXVD2X`’, ‘`LXVW4X`’, ‘`STXVD2X`’, and ‘`STXVW4X`’ instructions.

```
vector signed long long vec_signedo (vector float);
vector signed long long vec_signede (vector float);
vector unsigned long long vec_unsignedo (vector float);
vector unsigned long long vec_unsignede (vector float);
```

The overloaded built-ins `vec_signedo` and `vec_signede` are additional extensions to the built-ins as documented in the PVI PR.

7.13.24.3 PowerPC AltiVec Built-in Functions Available on ISA 2.07

If the ISA 2.07 additions to the vector/scalar (power8-vector) instruction set are available, the following additional functions are available for both 32-bit and 64-bit targets. For 64-bit targets, you can use *vector long* instead of *vector long long*, *vector bool long* instead of *vector bool long long*, and *vector unsigned long* instead of *vector unsigned long long*.

Only functions excluded from the PVI PR are listed here.

```
vector long long vec_vaddudm (vector long long, vector long long);
vector long long vec_vaddudm (vector bool long long, vector long long);
vector long long vec_vaddudm (vector long long, vector bool long long);
vector unsigned long long vec_vaddudm (vector unsigned long long,
                                       vector unsigned long long);
vector unsigned long long vec_vaddudm (vector bool unsigned long long,
                                       vector unsigned long long);
vector unsigned long long vec_vaddudm (vector unsigned long long,
                                       vector bool unsigned long long);

vector long long vec_vclz (vector long long);
vector unsigned long long vec_vclz (vector unsigned long long);
vector int vec_vclz (vector int);
vector unsigned int vec_vclz (vector int);
vector short vec_vclz (vector short);
vector unsigned short vec_vclz (vector unsigned short);
vector signed char vec_vclz (vector signed char);
vector unsigned char vec_vclz (vector unsigned char);

vector signed char vec_vclzb (vector signed char);
vector unsigned char vec_vclzb (vector unsigned char);

vector long long vec_vclzd (vector long long);
vector unsigned long long vec_vclzd (vector unsigned long long);

vector short vec_vclzh (vector short);
vector unsigned short vec_vclzh (vector unsigned short);

vector int vec_vclzw (vector int);
vector unsigned int vec_vclzw (vector int);

vector signed char vec_vgbdd (vector signed char);
vector unsigned char vec_vgbdd (vector unsigned char);
```

```

vector long long vec_vmaxsd (vector long long, vector long long);

vector unsigned long long vec_vmaxud (vector unsigned long long,
                                       unsigned vector long long);

vector long long vec_vminsd (vector long long, vector long long);

vector unsigned long long vec_vminud (vector long long, vector long long);

vector int vec_vpksdss (vector long long, vector long long);
vector unsigned int vec_vpksdss (vector long long, vector long long);

vector unsigned int vec_vp kudus (vector unsigned long long,
                                  vector unsigned long long);

vector int vec_vp kudum (vector long long, vector long long);
vector unsigned int vec_vp kudum (vector unsigned long long,
                                  vector unsigned long long);
vector bool int vec_vp kudum (vector bool long long, vector bool long long);

vector long long vec_vpopcnt (vector long long);
vector unsigned long long vec_vpopcnt (vector unsigned long long);
vector int vec_vpopcnt (vector int);
vector unsigned int vec_vpopcnt (vector int);
vector short vec_vpopcnt (vector short);
vector unsigned short vec_vpopcnt (vector unsigned short);
vector signed char vec_vpopcnt (vector signed char);
vector unsigned char vec_vpopcnt (vector unsigned char);

vector signed char vec_vpopcntb (vector signed char);
vector unsigned char vec_vpopcntb (vector unsigned char);

vector long long vec_vpopcntd (vector long long);
vector unsigned long long vec_vpopcntd (vector unsigned long long);

vector short vec_vpopcnth (vector short);
vector unsigned short vec_vpopcnth (vector unsigned short);

vector int vec_vpopcntw (vector int);
vector unsigned int vec_vpopcntw (vector int);

vector long long vec_vrld (vector long long, vector unsigned long long);
vector unsigned long long vec_vrld (vector unsigned long long,
                                    vector unsigned long long);

vector long long vec_vsld (vector long long, vector unsigned long long);
vector long long vec_vsld (vector unsigned long long,
                          vector unsigned long long);

vector long long vec_vsrld (vector long long, vector unsigned long long);
vector unsigned long long vec_vsrld (vector unsigned long long,
                                     vector unsigned long long);

vector long long vec_vsrld (vector long long, vector unsigned long long);
vector unsigned long long char vec_vsrld (vector unsigned long long,
                                           vector unsigned long long);

vector long long vec_vsubudm (vector long long, vector long long);

```

```

vector long long vec_vsubudm (vector bool long long, vector long long);
vector long long vec_vsubudm (vector long long, vector bool long long);
vector unsigned long long vec_vsubudm (vector unsigned long long,
                                         vector unsigned long long);
vector unsigned long long vec_vsubudm (vector bool long long,
                                         vector unsigned long long);
vector unsigned long long vec_vsubudm (vector unsigned long long,
                                         vector bool long long);

vector long long vec_vupkhs (vector int);
vector unsigned long long vec_vupkhs (vector unsigned int);

vector long long vec_vupkls (vector int);
vector unsigned long long vec_vupkls (vector int);

```

If the ISA 2.07 additions to the vector/scalar (power8-vector) instruction set are available, the following additional functions are available for 64-bit targets. New vector types (*vector __int128* and *vector __uint128*) are available to hold the *__int128* and *__uint128* types to use these builtins.

The normal vector extract, and set operations work on *vector __int128* and *vector __uint128* types, but the index value must be 0.

Only functions excluded from the PVI PR are listed here.

```

vector __int128 vec_vaddcuq (vector __int128, vector __int128);
vector __uint128 vec_vaddcuq (vector __uint128, vector __uint128);

vector __int128 vec_vadduqm (vector __int128, vector __int128);
vector __uint128 vec_vadduqm (vector __uint128, vector __uint128);

vector __int128 vec_vaddecuq (vector __int128, vector __int128,
                              vector __int128);
vector __uint128 vec_vaddecuq (vector __uint128, vector __uint128,
                              vector __uint128);

vector __int128 vec_vaddeuqm (vector __int128, vector __int128,
                              vector __int128);
vector __uint128 vec_vaddeuqm (vector __uint128, vector __uint128,
                              vector __uint128);

vector __int128 vec_vsubecuq (vector __int128, vector __int128,
                              vector __int128);
vector __uint128 vec_vsubecuq (vector __uint128, vector __uint128,
                              vector __uint128);

vector __int128 vec_vsubeuqm (vector __int128, vector __int128,
                              vector __int128);
vector __uint128 vec_vsubeuqm (vector __uint128, vector __uint128,
                              vector __uint128);

vector __int128 vec_vsubcuq (vector __int128, vector __int128);
vector __uint128 vec_vsubcuq (vector __uint128, vector __uint128);

__int128 vec_vsubuqm (__int128, __int128);
__uint128 vec_vsubuqm (__uint128, __uint128);

vector __int128 __builtin_bcdadd (vector __int128, vector __int128, const int);
vector unsigned char __builtin_bcdadd (vector unsigned char, vector unsigned char,
                                       const int);

```

```

int __builtin_bcdadd_lt (vector __int128, vector __int128, const int);
int __builtin_bcdadd_lt (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdadd_eq (vector __int128, vector __int128, const int);
int __builtin_bcdadd_eq (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdadd_gt (vector __int128, vector __int128, const int);
int __builtin_bcdadd_gt (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdadd_ov (vector __int128, vector __int128, const int);
int __builtin_bcdadd_ov (vector unsigned char, vector unsigned char, const int);

vector __int128 __builtin_bcdsub (vector __int128, vector __int128, const int);
vector unsigned char __builtin_bcdsub (vector unsigned char, vector unsigned char,
                                     const int);

int __builtin_bcdsub_le (vector __int128, vector __int128, const int);
int __builtin_bcdsub_le (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdsub_lt (vector __int128, vector __int128, const int);
int __builtin_bcdsub_lt (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdsub_eq (vector __int128, vector __int128, const int);
int __builtin_bcdsub_eq (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdsub_gt (vector __int128, vector __int128, const int);
int __builtin_bcdsub_gt (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdsub_ge (vector __int128, vector __int128, const int);
int __builtin_bcdsub_ge (vector unsigned char, vector unsigned char, const int);
int __builtin_bcdsub_ov (vector __int128, vector __int128, const int);
int __builtin_bcdsub_ov (vector unsigned char, vector unsigned char, const int);

```

7.13.24.4 PowerPC AltiVec Built-in Functions Available on ISA 3.0

The following additional built-in functions are also available for the PowerPC family of processors, starting with ISA 3.0 (`-mcpu=power9`) or later.

Only instructions excluded from the PVI PR are listed here.

```

unsigned int scalar_extract_exp (double source);
unsigned long long int scalar_extract_exp (__ieee128 source);

unsigned long long int scalar_extract_sig (double source);
unsigned __int128 scalar_extract_sig (__ieee128 source);

double scalar_insert_exp (unsigned long long int significand,
                          unsigned long long int exponent);
double scalar_insert_exp (double significand, unsigned long long int exponent);

ieee_128 scalar_insert_exp (unsigned __int128 significand,
                            unsigned long long int exponent);
ieee_128 scalar_insert_exp (ieee_128 significand, unsigned long long int exponent);
vector ieee_128 scalar_insert_exp (vector unsigned __int128 significand,
                                   vector unsigned long long int exponent);
vector unsigned long long scalar_extract_exp_to_vec (ieee_128);
vector unsigned __int128 scalar_extract_sig_to_vec (ieee_128);

int scalar_cmp_exp_gt (double arg1, double arg2);
int scalar_cmp_exp_lt (double arg1, double arg2);
int scalar_cmp_exp_eq (double arg1, double arg2);
int scalar_cmp_exp_unordered (double arg1, double arg2);

bool scalar_test_data_class (float source, const int condition);
bool scalar_test_data_class (double source, const int condition);
bool scalar_test_data_class (__ieee128 source, const int condition);

```

```

bool scalar_test_neg (float source);
bool scalar_test_neg (double source);
bool scalar_test_neg (__ieee128 source);

```

The `scalar_extract_exp` with a 64-bit source argument function requires an environment supporting ISA 3.0 or later. The `scalar_extract_exp` with a 128-bit source argument and `scalar_extract_sig` functions require a 64-bit environment supporting ISA 3.0 or later. The `scalar_extract_exp` and `scalar_extract_sig` built-in functions return the significand and the biased exponent value respectively of their `source` arguments. When supplied with a 64-bit `source` argument, the result returned by `scalar_extract_sig` has the 0x0010000000000000 bit set if the function's `source` argument is in normalized form. Otherwise, this bit is set to 0. When supplied with a 128-bit `source` argument, the 0x00010000000000000000000000000000 bit of the result is treated similarly. Note that the sign of the significand is not represented in the result returned from the `scalar_extract_sig` function. Use the `scalar_test_neg` function to test the sign of its `double` argument.

The `scalar_insert_exp` functions require a 64-bit environment supporting ISA 3.0 or later. When supplied with a 64-bit first argument, the `scalar_insert_exp` built-in function returns a double-precision floating point value that is constructed by assembling the values of its `significand` and `exponent` arguments. The sign of the result is copied from the most significant bit of the `significand` argument. The significand and exponent components of the result are composed of the least significant 11 bits of the `exponent` argument and the least significant 52 bits of the `significand` argument respectively.

When supplied with a 128-bit first argument, the `scalar_insert_exp` built-in function returns a quad-precision IEEE floating point value if the two arguments were scalar. If the two arguments are vectors, the return value is a vector IEEE floating point value. The sign bit of the result is copied from the most significant bit of the `significand` argument. The significand and exponent components of the result are composed of the least significant 15 bits of the `exponent` argument (element 0 on big-endian and element 1 on little-endian) and the least significant 112 bits of the `significand` argument respectively. Note, the `significand` is the scalar argument or in the case of vector arguments, `significand` is element 0 for big-endian and element 1 for little-endian.

The `scalar_extract_exp_to_vec`, and `scalar_extract_sig_to_vec` are similar to `scalar_extract_exp`, `scalar_extract_sig` except they return a vector result of type unsigned long long and unsigned __int128 respectively.

The `scalar_cmp_exp_gt`, `scalar_cmp_exp_lt`, `scalar_cmp_exp_eq`, and `scalar_cmp_exp_unordered` built-in functions return a non-zero value if `arg1` is greater than, less than, equal to, or not comparable to `arg2` respectively. The arguments are not comparable if one or the other equals NaN (not a number).

The `scalar_test_data_class` built-in function returns 1 if any of the condition tests enabled by the value of the `condition` variable are true, and 0 otherwise. The `condition` argument must be a compile-time constant integer with value not exceeding 127. The `condition` argument is encoded as a bitmask with each bit enabling the testing of a different condition, as characterized by the following:

```

0x40    Test for NaN
0x20    Test for +Infinity
0x10    Test for -Infinity
0x08    Test for +Zero

```

```

0x04    Test for -Zero
0x02    Test for +Denormal
0x01    Test for -Denormal

```

The `scalar_test_neg` built-in function returns 1 if its `source` argument holds a negative value, 0 otherwise.

The following built-in functions are also available for the PowerPC family of processors, starting with ISA 3.0 or later (`-mcpu=power9`). These string functions are described separately in order to group the descriptions closer to the function prototypes.

Only functions excluded from the PVI PR are listed here.

```

int vec_all_nez (vector signed char, vector signed char);
int vec_all_nez (vector unsigned char, vector unsigned char);
int vec_all_nez (vector signed short, vector signed short);
int vec_all_nez (vector unsigned short, vector unsigned short);
int vec_all_nez (vector signed int, vector signed int);
int vec_all_nez (vector unsigned int, vector unsigned int);

int vec_any_eqz (vector signed char, vector signed char);
int vec_any_eqz (vector unsigned char, vector unsigned char);
int vec_any_eqz (vector signed short, vector signed short);
int vec_any_eqz (vector unsigned short, vector unsigned short);
int vec_any_eqz (vector signed int, vector signed int);
int vec_any_eqz (vector unsigned int, vector unsigned int);

signed char vec_xlx (unsigned int index, vector signed char data);
unsigned char vec_xlx (unsigned int index, vector unsigned char data);
signed short vec_xlx (unsigned int index, vector signed short data);
unsigned short vec_xlx (unsigned int index, vector unsigned short data);
signed int vec_xlx (unsigned int index, vector signed int data);
unsigned int vec_xlx (unsigned int index, vector unsigned int data);
float vec_xlx (unsigned int index, vector float data);

signed char vec_xrx (unsigned int index, vector signed char data);
unsigned char vec_xrx (unsigned int index, vector unsigned char data);
signed short vec_xrx (unsigned int index, vector signed short data);
unsigned short vec_xrx (unsigned int index, vector unsigned short data);
signed int vec_xrx (unsigned int index, vector signed int data);
unsigned int vec_xrx (unsigned int index, vector unsigned int data);
float vec_xrx (unsigned int index, vector float data);

```

The `vec_all_nez`, `vec_any_eqz`, and `vec_cmpnez` perform pairwise comparisons between the elements at the same positions within their two vector arguments. The `vec_all_nez` function returns a non-zero value if and only if all pairwise comparisons are not equal and no element of either vector argument contains a zero. The `vec_any_eqz` function returns a non-zero value if and only if at least one pairwise comparison is equal or if at least one element of either vector argument contains a zero. The `vec_cmpnez` function returns a vector of the same type as its two arguments, within which each element consists of all ones to denote that either the corresponding elements of the incoming arguments are not equal or that at least one of the corresponding elements contains zero. Otherwise, the element of the returned vector contains all zeros.

The `vec_xlx` and `vec_xrx` functions extract the single element selected by the `index` argument from the vector represented by the `data` argument. The `index` argument always specifies a byte offset, regardless of the size of the vector element. With `vec_xlx`, `index` is the offset of the first byte of the element to be extracted. With `vec_xrx`, `index` represents

the last byte of the element to be extracted, measured from the right end of the vector. In other words, the last byte of the element to be extracted is found at position `(15 - index)`. There is no requirement that `index` be a multiple of the vector element size. However, if the size of the vector element added to `index` is greater than 15, the content of the returned value is undefined.

The following functions are also available if the ISA 3.0 instruction set additions (`-mcpu=power9`) are available.

Only functions excluded from the PVI PR are listed here.

```
vector long long vec_vctz (vector long long);
vector unsigned long long vec_vctz (vector unsigned long long);
vector int vec_vctz (vector int);
vector unsigned int vec_vctz (vector int);
vector short vec_vctz (vector short);
vector unsigned short vec_vctz (vector unsigned short);
vector signed char vec_vctz (vector signed char);
vector unsigned char vec_vctz (vector unsigned char);

vector signed char vec_vctzb (vector signed char);
vector unsigned char vec_vctzb (vector unsigned char);

vector long long vec_vctzd (vector long long);
vector unsigned long long vec_vctzd (vector unsigned long long);

vector short vec_vctzh (vector short);
vector unsigned short vec_vctzh (vector unsigned short);

vector int vec_vctzw (vector int);
vector unsigned int vec_vctzw (vector int);

vector int vec_vpptyb (vector int);
vector unsigned int vec_vpptyb (vector unsigned int);
vector long long vec_vpptyb (vector long long);
vector unsigned long long vec_vpptyb (vector unsigned long long);

vector int vec_vpptybw (vector int);
vector unsigned int vec_vpptybw (vector unsigned int);

vector long long vec_vpptybd (vector long long);
vector unsigned long long vec_vpptybd (vector unsigned long long);
```

On 64-bit targets, if the ISA 3.0 additions (`-mcpu=power9`) are available:

```
vector long vec_vpptyb (vector long);
vector unsigned long vec_vpptyb (vector unsigned long);
vector __int128 vec_vpptyb (vector __int128);
vector __uint128 vec_vpptyb (vector __uint128);

vector long vec_vpptybd (vector long);
vector unsigned long vec_vpptybd (vector unsigned long);

vector __int128 vec_vpptybq (vector __int128);
vector __uint128 vec_vpptybd (vector __uint128);
```

The following built-in functions are available for the PowerPC family of processors, starting with ISA 3.0 or later (`-mcpu=power9`).

Only functions excluded from the PVI PR are listed here.

```
__vector unsigned char
```

```

vec_absdb (__vector unsigned char arg1, __vector unsigned char arg2);
__vector unsigned short
vec_absdh (__vector unsigned short arg1, __vector unsigned short arg2);
__vector unsigned int
vec_absdw (__vector unsigned int arg1, __vector unsigned int arg2);

```

The `vec_absd`, `vec_absdb`, `vec_absdh`, and `vec_absdw` built-in functions each computes the absolute differences of the pairs of vector elements supplied in its two vector arguments, placing the absolute differences into the corresponding elements of the vector result.

The following built-in functions are available for the PowerPC family of processors, starting with ISA 3.0 or later (`-mcpu=power9`):

```

vector unsigned int vec_vrlnm (vector unsigned int, vector unsigned int);
vector unsigned long long vec_vrlnm (vector unsigned long long,
                                     vector unsigned long long);

```

The result of `vec_vrlnm` is obtained by rotating each element of the first argument vector left and ANDing it with a mask. The second argument vector contains the mask beginning in bits 11:15, the mask end in bits 19:23, and the shift count in bits 27:31, of each element.

If the cryptographic instructions are enabled (`-mcrypto` or `-mcpu=power8`), the following builtins are enabled.

Only functions excluded from the PVIPR are listed here.

```

vector unsigned long long __builtin_crypto_vsbox (vector unsigned long long);

vector unsigned long long __builtin_crypto_vcipher (vector unsigned long long,
                                                    vector unsigned long long);

vector unsigned long long __builtin_crypto_vcipherlast
    (vector unsigned long long,
     vector unsigned long long);

vector unsigned long long __builtin_crypto_vncipher (vector unsigned long long,
                                                    vector unsigned long long);

vector unsigned long long __builtin_crypto_vncipherlast (vector unsigned long long,
                                                         vector unsigned long long);

vector unsigned char __builtin_crypto_vpermxor (vector unsigned char,
                                                vector unsigned char,
                                                vector unsigned char);

vector unsigned short __builtin_crypto_vpermxor (vector unsigned short,
                                                  vector unsigned short,
                                                  vector unsigned short);

vector unsigned int __builtin_crypto_vpermxor (vector unsigned int,
                                                vector unsigned int,
                                                vector unsigned int);

vector unsigned long long __builtin_crypto_vpermxor (vector unsigned long long,
                                                      vector unsigned long long,
                                                      vector unsigned long long);

vector unsigned char __builtin_crypto_vpmsumb (vector unsigned char,
                                                vector unsigned char);

vector unsigned short __builtin_crypto_vpmsumh (vector unsigned short,

```

```

vector unsigned short);

vector unsigned int __builtin_crypto_vpmsumw (vector unsigned int,
                                              vector unsigned int);

vector unsigned long long __builtin_crypto_vpmsumd (vector unsigned long long,
                                                    vector unsigned long long);

vector unsigned long long __builtin_crypto_vshasigmad (vector unsigned long long,
                                                       int, int);

vector unsigned int __builtin_crypto_vshasigmaw (vector unsigned int, int, int);

```

The second argument to `__builtin_crypto_vshasigmad` and `__builtin_crypto_vshasigmaw` must be a constant integer that is 0 or 1. The third argument to these built-in functions must be a constant integer in the range of 0 to 15.

The following sign extension builtins are provided:

```

vector signed int vec_signexti (vector signed char a);
vector signed long long vec_signextll (vector signed char a);
vector signed int vec_signexti (vector signed short a);
vector signed long long vec_signextll (vector signed short a);
vector signed long long vec_signextll (vector signed int a);
vector signed long long vec_signextq (vector signed long long a);

```

Each element of the result is produced by sign-extending the element of the input vector that would fall in the least significant portion of the result element. For example, a sign-extension of a vector signed char to a vector signed long long will sign extend the rightmost byte of each doubleword.

7.13.24.5 PowerPC AltiVec Built-in Functions Available on ISA 3.1

The following additional built-in functions are also available for the PowerPC family of processors, starting with ISA 3.1 (`-mcpu=power10`):

```

int vec_test_lsbb_all_ones (vector signed char);
int vec_test_lsbb_all_ones (vector unsigned char);
int vec_test_lsbb_all_ones (vector bool char);

```

The builtin `vec_test_lsbb_all_ones` returns 1 if the least significant bit in each byte is equal to 1. It returns 0 otherwise.

```

int vec_test_lsbb_all_zeros (vector signed char);
int vec_test_lsbb_all_zeros (vector unsigned char);
int vec_test_lsbb_all_zeros (vector bool char);

```

The builtin `vec_test_lsbb_all_zeros` returns 1 if the least significant bit in each byte is equal to zero. It returns 0 otherwise.

```

vector unsigned long long int
vec_cfuge (vector unsigned long long int, vector unsigned long long int);

```

Perform a vector centrifuge operation, as if implemented by the `vcfuged` instruction.

```

vector unsigned long long int
vec_cntlzm (vector unsigned long long int, vector unsigned long long int);

```

Perform a vector count leading zeros under bit mask operation, as if implemented by the `vclzdm` instruction.

```

vector unsigned long long int
vec_cnttzm (vector unsigned long long int, vector unsigned long long int);

```

Perform a vector count trailing zeros under bit mask operation, as if implemented by the **vctzdm** instruction.

```
vector signed char
vec_clrl (vector signed char a, unsigned int n);
vector unsigned char
vec_clrl (vector unsigned char a, unsigned int n);
```

Clear the left-most $(16 - n)$ bytes of vector argument **a**, as if implemented by the **vclrlb** instruction on a big-endian target and by the **vclrrb** instruction on a little-endian target. A value of **n** that is greater than 16 is treated as if it equaled 16.

```
vector signed char
vec_clrr (vector signed char a, unsigned int n);
vector unsigned char
vec_clrr (vector unsigned char a, unsigned int n);
```

Clear the right-most $(16 - n)$ bytes of vector argument **a**, as if implemented by the **vclrrb** instruction on a big-endian target and by the **vclrlb** instruction on a little-endian target. A value of **n** that is greater than 16 is treated as if it equaled 16.

```
vector unsigned long long int
vec_gnb (vector unsigned __int128, const unsigned char);
```

Perform a 128-bit vector gather operation, as if implemented by the **vgnb** instruction. The second argument must be a literal integer value between 2 and 7 inclusive.

Vector Extract

```
vector unsigned long long int
vec_extractl (vector unsigned char, vector unsigned char, unsigned int);
vector unsigned long long int
vec_extractl (vector unsigned short, vector unsigned short, unsigned int);
vector unsigned long long int
vec_extractl (vector unsigned int, vector unsigned int, unsigned int);
vector unsigned long long int
vec_extractl (vector unsigned long long, vector unsigned long long, unsigned int);
```

Extract an element from two concatenated vectors starting at the given byte index in natural-endian order, and place it zero-extended in doubleword 1 of the result according to natural element order. If the byte index is out of range for the data type, the intrinsic will be rejected. For little-endian, this output will match the placement by the hardware instruction, i.e., `dword[0]` in RTL notation. For big-endian, an additional instruction is needed to move it from the "left" doubleword to the "right" one. For little-endian, semantics matching the **vextdubvr_x**, **vextduhvr_x**, **vextduwvr_x** instruction will be generated, while for big-endian, semantics matching the **vextdubvl_x**, **vextduhvl_x**, **vextduwvl_x** instructions will be generated. Note that some fairly anomalous results can be generated if the byte index is not aligned on an element boundary for the element being extracted. This is a limitation of the bi-endian vector programming model is consistent with the limitation on **vec_perm**.

```
vector unsigned long long int
vec_extracth (vector unsigned char, vector unsigned char, unsigned int);
vector unsigned long long int
vec_extracth (vector unsigned short, vector unsigned short,
              unsigned int);
vector unsigned long long int
vec_extracth (vector unsigned int, vector unsigned int, unsigned int);
vector unsigned long long int
vec_extracth (vector unsigned long long, vector unsigned long long,
```

```
unsigned int);
```

Extract an element from two concatenated vectors starting at the given byte index. The index is based on big endian order for a little endian system. Similarly, the index is based on little endian order for a big endian system. The extracted elements are zero-extended and put in doubleword 1 according to natural element order. If the byte index is out of range for the data type, the intrinsic will be rejected. For little-endian, this output will match the placement by the hardware instruction (`vextdubvrx`, `vextduhvr`, `vextduwvr`, `vextddvr`) i.e., `dword[0]` in RTL notation. For big-endian, an additional instruction is needed to move it from the "left" doubleword to the "right" one. For little-endian, semantics matching the `vextdubvlx`, `vextduhvlx`, `vextduwvlx` instructions will be generated, while for big-endian, semantics matching the `vextdubvr`, `vextduhvr`, `vextduwvr` instructions will be generated. Note that some fairly anomalous results can be generated if the byte index is not aligned on the element boundary for the element being extracted. This is a limitation of the bi-endian vector programming model consistent with the limitation on `vec_perm`.

```
vector unsigned long long int
```

```
vec_pdep (vector unsigned long long int, vector unsigned long long int);
```

Perform a vector parallel bits deposit operation, as if implemented by the `vpdepd` instruction.

Vector Insert

```
vector unsigned char
```

```
vec_insertl (unsigned char, vector unsigned char, unsigned int);
```

```
vector unsigned short
```

```
vec_insertl (unsigned short, vector unsigned short, unsigned int);
```

```
vector unsigned int
```

```
vec_insertl (unsigned int, vector unsigned int, unsigned int);
```

```
vector unsigned long long
```

```
vec_insertl (unsigned long long, vector unsigned long long,
```

```
    unsigned int);
```

```
vector unsigned char
```

```
vec_insertl (vector unsigned char, vector unsigned char, unsigned int);
```

```
vector unsigned short
```

```
vec_insertl (vector unsigned short, vector unsigned short,
```

```
    unsigned int);
```

```
vector unsigned int
```

```
vec_insertl (vector unsigned int, vector unsigned int, unsigned int);
```

Let `src` be the first argument, when the first argument is a scalar, or the rightmost element of the left doubleword of the first argument, when the first argument is a vector. Insert the source into the destination at the position given by the third argument, using natural element order in the second argument. The rest of the second argument is unchanged. If the byte index is greater than 14 for halfwords, greater than 12 for words, or greater than 8 for doublewords the result is undefined. For little-endian, the generated code will be semantically equivalent to `vins[bhwd]rx` instructions. Similarly for big-endian it will be semantically equivalent to `vins[bhwd]lx`. Note that some fairly anomalous results can be generated if the byte index is not aligned on an element boundary for the type of element being inserted.

```
vector unsigned char
```

```
vec_inserth (unsigned char, vector unsigned char, unsigned int);
```

```
vector unsigned short
```

```
vec_inserth (unsigned short, vector unsigned short, unsigned int);
```

```
vector unsigned int
```

```
vec.inserth (unsigned int, vector unsigned int, unsigned int);
vector unsigned long long
vec.inserth (unsigned long long, vector unsigned long long,
             unsigned int);
vector unsigned char
vec.inserth (vector unsigned char, vector unsigned char, unsigned int);
vector unsigned short
vec.inserth (vector unsigned short, vector unsigned short,
             unsigned int);
vector unsigned int
vec.inserth (vector unsigned int, vector unsigned int, unsigned int);
```

Let `src` be the first argument, when the first argument is a scalar, or the rightmost element of the first argument, when the first argument is a vector. Insert `src` into the second argument at the position identified by the third argument, using opposite element order in the second argument, and leaving the rest of the second argument unchanged. If the byte index is greater than 14 for halfwords, 12 for words, or 8 for doublewords, the intrinsic will be rejected. Note that the underlying hardware instruction uses the same register for the second argument and the result. For little-endian, the code generation will be semantically equivalent to `vins[bhwd]lx`, while for big-endian it will be semantically equivalent to `vins[bhwd]rx`. Note that some fairly anomalous results can be generated if the byte index is not aligned on an element boundary for the sort of element being inserted.

Vector Replace Element

```
vector signed int vec_replace_elt (vector signed int, signed int,
    const int);
vector unsigned int vec_replace_elt (vector unsigned int,
    unsigned int, const int);
vector float vec_replace_elt (vector float, float, const int);
vector signed long long vec_replace_elt (vector signed long long,
    signed long long, const int);
vector unsigned long long vec_replace_elt (vector unsigned long long,
    unsigned long long, const int);
vector double rec_replace_elt (vector double, double, const int);
```

The third argument (constrained to $[0,3]$) identifies the natural-endian element number of the first argument that will be replaced by the second argument to produce the result. The other elements of the first argument will remain unchanged in the result.

If it's desirable to insert a word at an unaligned position, use `vec_replace_unaligned` instead.

Vector Replace Unaligned

```
vector<unsigned char> vec_replace_unaligned (vector<unsigned char>,
signed int, const int);
vector<unsigned char> vec_replace_unaligned (vector<unsigned char>,
unsigned int, const int);
vector<unsigned char> vec_replace_unaligned (vector<unsigned char>,
float, const int);
vector<unsigned char> vec_replace_unaligned (vector<unsigned char>,
signed long long, const int);
vector<unsigned char> vec_replace_unaligned (vector<unsigned char>,
unsigned long long, const int);
vector<unsigned char> vec_replace_unaligned (vector<unsigned char>,
double, const int);
```

The second argument replaces a portion of the first argument to produce the result, with the rest of the first argument unchanged in the result. The third argument identifies the byte index (using left-to-right, or big-endian order) where the high-order byte of the second argument will be placed, with the remaining bytes of the second argument placed naturally "to the right" of the high-order byte.

The programmer is responsible for understanding the endianness issues involved with the first argument and the result.

Vector Shift Left Double Bit Immediate

```
vector signed char vec_sldb (vector signed char, vector signed char,
    const unsigned int);
vector unsigned char vec_sldb (vector unsigned char,
    vector unsigned char, const unsigned int);
vector signed short vec_sldb (vector signed short, vector signed short,
    const unsigned int);
vector unsigned short vec_sldb (vector unsigned short,
    vector unsigned short, const unsigned int);
vector signed int vec_sldb (vector signed int, vector signed int,
    const unsigned int);
vector unsigned int vec_sldb (vector unsigned int, vector unsigned int,
    const unsigned int);
vector signed long long vec_sldb (vector signed long long,
    vector signed long long, const unsigned int);
vector unsigned long long vec_sldb (vector unsigned long long,
    vector unsigned long long, const unsigned int);
vector signed __int128 vec_sldb (vector signed __int128,
    vector signed __int128, const unsigned int);
vector unsigned __int128 vec_sldb (vector unsigned __int128,
    vector unsigned __int128, const unsigned int);
```

Shift the combined input vectors left by the amount specified by the low-order three bits of the third argument, and return the leftmost remaining 128 bits. Code using this instruction must be endian-aware.

Vector Shift Right Double Bit Immediate

```
vector signed char vec_srdh (vector signed char, vector signed char,
    const unsigned int);
vector unsigned char vec_srdh (vector unsigned char, vector unsigned char,
    const unsigned int);
vector signed short vec_srdh (vector signed short, vector signed short,
    const unsigned int);
vector unsigned short vec_srdh (vector unsigned short, vector unsigned short,
    const unsigned int);
vector signed int vec_srdh (vector signed int, vector signed int,
    const unsigned int);
vector unsigned int vec_srdh (vector unsigned int, vector unsigned int,
    const unsigned int);
vector signed long long vec_srdh (vector signed long long,
    vector signed long long, const unsigned int);
vector unsigned long long vec_srdh (vector unsigned long long,
    vector unsigned long long, const unsigned int);
vector signed __int128 vec_srdh (vector signed __int128,
    vector signed __int128, const unsigned int);
vector unsigned __int128 vec_srdh (vector unsigned __int128,
    vector unsigned __int128, const unsigned int);
```

Shift the combined input vectors right by the amount specified by the low-order three bits of the third argument, and return the remaining 128 bits. Code using this built-in must be endian-aware.

Vector Splat

```
vector signed int vec_splati (const signed int);
vector float vec_splati (const float);
```

Splat a 32-bit immediate into a vector of words.

```
vector double vec_splatid (const float);
```

Convert a single precision floating-point value to double-precision and splat the result to a vector of double-precision floats.

```
vector signed int vec_splati_ins (vector signed int,
    const unsigned int, const signed int);
vector unsigned int vec_splati_ins (vector unsigned int,
    const unsigned int, const unsigned int);
vector float vec_splati_ins (vector float, const unsigned int,
    const float);
```

Argument 2 must be either 0 or 1. Splat the value of argument 3 into the word identified by argument 2 of each doubleword of argument 1 and return the result. The other words of argument 1 are unchanged.

Vector Blend Variable

```
vector signed char vec_blendv (vector signed char, vector signed char,
    vector unsigned char);
vector unsigned char vec_blendv (vector unsigned char,
    vector unsigned char, vector unsigned char);
vector signed short vec_blendv (vector signed short,
    vector signed short, vector unsigned short);
vector unsigned short vec_blendv (vector unsigned short,
    vector unsigned short, vector unsigned short);
vector signed int vec_blendv (vector signed int, vector signed int,
    vector unsigned int);
vector unsigned int vec_blendv (vector unsigned int,
    vector unsigned int, vector unsigned int);
vector signed long long vec_blendv (vector signed long long,
    vector signed long long, vector unsigned long long);
vector unsigned long long vec_blendv (vector unsigned long long,
    vector unsigned long long, vector unsigned long long);
vector float vec_blendv (vector float, vector float,
    vector unsigned int);
vector double vec_blendv (vector double, vector double,
    vector unsigned long long);
```

Blend the first and second argument vectors according to the sign bits of the corresponding elements of the third argument vector. This is similar to the `vsel` and `xxsel` instructions but for bigger elements.

Vector Permute Extended

```
vector signed char vec_permx (vector signed char, vector signed char,
    vector unsigned char, const int);
vector unsigned char vec_permx (vector unsigned char,
    vector unsigned char, vector unsigned char, const int);
vector signed short vec_permx (vector signed short,
    vector signed short, vector unsigned char, const int);
vector unsigned short vec_permx (vector unsigned short,
    vector unsigned short, vector unsigned char, const int);
```

```

vector signed int vec_permx (vector signed int, vector signed int,
    vector unsigned char, const int);
vector unsigned int vec_permx (vector unsigned int,
    vector unsigned char, const int);
vector signed long long vec_permx (vector signed long long,
    vector signed long long, vector unsigned char, const int);
vector unsigned long long vec_permx (vector unsigned long long,
    vector unsigned long long, vector unsigned char, const int);
vector float (vector float, vector float, vector unsigned char,
    const int);
vector double (vector double, vector double, vector unsigned char,
    const int);

```

Perform a partial permute of the first two arguments, which form a 32-byte section of an emulated vector up to 256 bytes wide, using the partial permute control vector in the third argument. The fourth argument (constrained to values of 0-7) identifies which 32-byte section of the emulated vector is contained in the first two arguments.

```

vector unsigned long long int
vec_pext (vector unsigned long long int, vector unsigned long long int);

```

Perform a vector parallel bit extract operation, as if implemented by the `vpextd` instruction.

```

vector unsigned char vec_stril (vector unsigned char);
vector signed char vec_stril (vector signed char);
vector unsigned short vec_stril (vector unsigned short);
vector signed short vec_stril (vector signed short);

```

Isolate the left-most non-zero elements of the incoming vector argument, replacing all elements to the right of the left-most zero element found within the argument with zero. The typical implementation uses the `vstribl` or `vstrihl` instruction on big-endian targets and uses the `vstribr` or `vstrihr` instruction on little-endian targets.

```

int vec_stril_p (vector unsigned char);
int vec_stril_p (vector signed char);
int short vec_stril_p (vector unsigned short);
int vec_stril_p (vector signed short);

```

Return a non-zero value if and only if the argument contains a zero element. The typical implementation uses the `vstribl` or `vstrihl` instruction on big-endian targets and uses the `vstribr` or `vstrihr` instruction on little-endian targets. Choose this built-in to check for presence of zero element if the same argument is also passed to `vec_stril`.

```

vector unsigned char vec_strir (vector unsigned char);
vector signed char vec_strir (vector signed char);
vector unsigned short vec_strir (vector unsigned short);
vector signed short vec_strir (vector signed short);

```

Isolate the right-most non-zero elements of the incoming vector argument, replacing all elements to the left of the right-most zero element found within the argument with zero. The typical implementation uses the `vstribr` or `vstrihr` instruction on big-endian targets and uses the `vstribl` or `vstrihl` instruction on little-endian targets.

```

int vec_strir_p (vector unsigned char);
int vec_strir_p (vector signed char);
int short vec_strir_p (vector unsigned short);
int vec_strir_p (vector signed short);

```

Return a non-zero value if and only if the argument contains a zero element. The typical implementation uses the `vstribr` or `vstrihr` instruction on big-endian targets and uses

the `vstrl` or `vstribl` instruction on little-endian targets. Choose this built-in to check for presence of zero element if the same argument is also passed to `vec_strir`.

```
vector unsigned char
vec_ternarylogic (vector unsigned char, vector unsigned char,
                  vector unsigned char, const unsigned int);
vector unsigned short
vec_ternarylogic (vector unsigned short, vector unsigned short,
                  vector unsigned short, const unsigned int);
vector unsigned int
vec_ternarylogic (vector unsigned int, vector unsigned int,
                  vector unsigned int, const unsigned int);
vector unsigned long long int
vec_ternarylogic (vector unsigned long long int, vector unsigned long long int,
                  vector unsigned long long int, const unsigned int);
vector unsigned __int128
vec_ternarylogic (vector unsigned __int128, vector unsigned __int128,
                  vector unsigned __int128, const unsigned int);
```

Perform a 128-bit vector evaluate operation, as if implemented by the `xxeval` instruction. The fourth argument must be a literal integer value between 0 and 255 inclusive.

```
vector unsigned char vec_genpcvm (vector unsigned char, const int);
vector unsigned short vec_genpcvm (vector unsigned short, const int);
vector unsigned int vec_genpcvm (vector unsigned int, const int);
vector unsigned long long int vec_genpcvm (vector unsigned long long int,
                                           const int);
```

Vector Integer Multiply/Divide/Modulo

```
vector signed int
vec_mulh (vector signed int a, vector signed int b);
vector unsigned int
vec_mulh (vector unsigned int a, vector unsigned int b);
```

For each integer value *i* from 0 to 3, do the following. The integer value in word element *i* of *a* is multiplied by the integer value in word element *i* of *b*. The high-order 32 bits of the 64-bit product are placed into word element *i* of the vector returned.

```
vector signed long long
vec_mulh (vector signed long long a, vector signed long long b);
vector unsigned long long
vec_mulh (vector unsigned long long a, vector unsigned long long b);
```

For each integer value *i* from 0 to 1, do the following. The integer value in doubleword element *i* of *a* is multiplied by the integer value in doubleword element *i* of *b*. The high-order 64 bits of the 128-bit product are placed into doubleword element *i* of the vector returned.

```
vector unsigned long long
vec_mul (vector unsigned long long a, vector unsigned long long b);
vector signed long long
vec_mul (vector signed long long a, vector signed long long b);
```

For each integer value *i* from 0 to 1, do the following. The integer value in doubleword element *i* of *a* is multiplied by the integer value in doubleword element *i* of *b*. The low-order 64 bits of the 128-bit product are placed into doubleword element *i* of the vector returned.

```
vector signed int
vec_div (vector signed int a, vector signed int b);
vector unsigned int
vec_div (vector unsigned int a, vector unsigned int b);
```

For each integer value *i* from 0 to 3, do the following. The integer in word element *i* of *a* is divided by the integer in word element *i* of *b*. The unique integer quotient is placed into the word element *i* of the vector returned. If an attempt is made to perform any of the divisions $\langle \text{anything} \rangle \div 0$ then the quotient is undefined.

```
vector signed long long
vec_div (vector signed long long a, vector signed long long b);
vector unsigned long long
vec_div (vector unsigned long long a, vector unsigned long long b);
```

For each integer value *i* from 0 to 1, do the following. The integer in doubleword element *i* of *a* is divided by the integer in doubleword element *i* of *b*. The unique integer quotient is placed into the doubleword element *i* of the vector returned. If an attempt is made to perform any of the divisions $0x8000_0000_0000_0000 \div -1$ or $\langle \text{anything} \rangle \div 0$ then the quotient is undefined.

```
vector signed int
vec_dive (vector signed int a, vector signed int b);
vector unsigned int
vec_dive (vector unsigned int a, vector unsigned int b);
```

For each integer value *i* from 0 to 3, do the following. The integer in word element *i* of *a* is shifted left by 32 bits, then divided by the integer in word element *i* of *b*. The unique integer quotient is placed into the word element *i* of the vector returned. If the quotient cannot be represented in 32 bits, or if an attempt is made to perform any of the divisions $\langle \text{anything} \rangle \div 0$ then the quotient is undefined.

```
vector signed long long
vec_dive (vector signed long long a, vector signed long long b);
vector unsigned long long
vec_dive (vector unsigned long long a, vector unsigned long long b);
```

For each integer value *i* from 0 to 1, do the following. The integer in doubleword element *i* of *a* is shifted left by 64 bits, then divided by the integer in doubleword element *i* of *b*. The unique integer quotient is placed into the doubleword element *i* of the vector returned. If the quotient cannot be represented in 64 bits, or if an attempt is made to perform $\langle \text{anything} \rangle \div 0$ then the quotient is undefined.

```
vector signed int
vec_mod (vector signed int a, vector signed int b);
vector unsigned int
vec_mod (vector unsigned int a, vector unsigned int b);
```

For each integer value *i* from 0 to 3, do the following. The integer in word element *i* of *a* is divided by the integer in word element *i* of *b*. The unique integer remainder is placed into the word element *i* of the vector returned. If an attempt is made to perform any of the divisions $0x8000_0000 \div -1$ or $\langle \text{anything} \rangle \div 0$ then the remainder is undefined.

```
vector signed long long
vec_mod (vector signed long long a, vector signed long long b);
vector unsigned long long
vec_mod (vector unsigned long long a, vector unsigned long long b);
```

For each integer value *i* from 0 to 1, do the following. The integer in doubleword element *i* of *a* is divided by the integer in doubleword element *i* of *b*. The unique integer remainder is placed into the doubleword element *i* of the vector returned. If an attempt is made to perform $\langle \text{anything} \rangle \div 0$ then the remainder is undefined.

Generate PCV from specified Mask size, as if implemented by the `xxgenpcvbm`, `xxgenpcvbm`, `xxgenpcvwm` instructions, where immediate value is either 0, 1, 2 or 3.

Returns a vector containing a 128-bit integer result of multiplying the odd doubleword elements of the two inputs.

```
vector unsigned __int128 vec_div (vector unsigned __int128,
                                vector unsigned __int128);
vector signed __int128 vec_div (vector signed __int128,
                                vector signed __int128);
```

Returns the result of dividing the first operand by the second operand. An attempt to divide any value by zero or to divide the most negative signed 128-bit integer by negative one results in an undefined value.

```
vector unsigned __int128 vec_dive (vector unsigned __int128,
                                   vector unsigned __int128);
vector signed __int128 vec_dive (vector signed __int128,
                                 vector signed __int128);
```

The result is produced by shifting the first input left by 128 bits and dividing by the second. If an attempt is made to divide by zero or the result is larger than 128 bits, the result is undefined.

```
vector unsigned __int128 vec_mod (vector unsigned __int128,
                                  vector unsigned __int128);
vector signed __int128 vec_mod (vector signed __int128,
                                vector signed __int128);
```

The result is the modulo result of dividing the first input by the second input.

The following builtins perform 128-bit vector comparisons. The `vec_all_xx`, `vec_any_xx`, and `vec_cmpxx`, where `xx` is one of the operations `eq`, `ne`, `gt`, `lt`, `ge`, `le` perform pairwise comparisons between the elements at the same positions within their two vector arguments. The `vec_all_xx` function returns a non-zero value if and only if all pairwise comparisons are true. The `vec_any_xx` function returns a non-zero value if and only if at least one pairwise comparison is true. The `vec_cmpxx` function returns a vector of the same type as its two arguments, within which each element consists of all ones to denote that specified logical comparison of the corresponding elements was true. Otherwise, the element of the returned vector contains all zeros.

```
vector bool __int128 vec_cmpeq (vector signed __int128, vector signed __int128);
vector bool __int128 vec_cmpeq (vector unsigned __int128, vector unsigned __int128);
vector bool __int128 vec_cmpne (vector signed __int128, vector signed __int128);
vector bool __int128 vec_cmpne (vector unsigned __int128, vector unsigned __int128);
vector bool __int128 vec_cmpgt (vector signed __int128, vector signed __int128);
vector bool __int128 vec_cmpgt (vector unsigned __int128, vector unsigned __int128);
vector bool __int128 vec_cmplt (vector signed __int128, vector signed __int128);
vector bool __int128 vec_cmplt (vector unsigned __int128, vector unsigned __int128);
vector bool __int128 vec_cmpge (vector signed __int128, vector signed __int128);
vector bool __int128 vec_cmpge (vector unsigned __int128, vector unsigned __int128);
vector bool __int128 vec_cmple (vector signed __int128, vector signed __int128);
vector bool __int128 vec_cmple (vector unsigned __int128, vector unsigned __int128);

int vec_all_eq (vector signed __int128, vector signed __int128);
int vec_all_eq (vector unsigned __int128, vector unsigned __int128);
int vec_all_ne (vector signed __int128, vector signed __int128);
int vec_all_ne (vector unsigned __int128, vector unsigned __int128);
int vec_all_gt (vector signed __int128, vector signed __int128);
int vec_all_gt (vector unsigned __int128, vector unsigned __int128);
int vec_all_lt (vector signed __int128, vector signed __int128);
int vec_all_lt (vector unsigned __int128, vector unsigned __int128);
int vec_all_ge (vector signed __int128, vector signed __int128);
```

```

int vec_all_ge (vector unsigned __int128, vector unsigned __int128);
int vec_all_le (vector signed __int128, vector signed __int128);
int vec_all_le (vector unsigned __int128, vector unsigned __int128);

int vec_any_eq (vector signed __int128, vector signed __int128);
int vec_any_eq (vector unsigned __int128, vector unsigned __int128);
int vec_any_ne (vector signed __int128, vector signed __int128);
int vec_any_ne (vector unsigned __int128, vector unsigned __int128);
int vec_any_gt (vector signed __int128, vector signed __int128);
int vec_any_gt (vector unsigned __int128, vector unsigned __int128);
int vec_any_lt (vector signed __int128, vector signed __int128);
int vec_any_lt (vector unsigned __int128, vector unsigned __int128);
int vec_any_ge (vector signed __int128, vector signed __int128);
int vec_any_ge (vector unsigned __int128, vector unsigned __int128);
int vec_any_le (vector signed __int128, vector signed __int128);
int vec_any_le (vector unsigned __int128, vector unsigned __int128);

```

The following instances are extension of the existing overloaded built-ins `vec_sld`, `vec_sldw`, `vec_slo`, `vec_sro`, `vec_srl` that are documented in the PVIPR.

```

vector signed __int128 vec_sld (vector signed __int128,
    vector signed __int128, const unsigned int);
vector unsigned __int128 vec_sld (vector unsigned __int128,
    vector unsigned __int128, const unsigned int);
vector signed __int128 vec_sldw (vector signed __int128,
    vector signed __int128, const unsigned int);
vector unsigned __int128 vec_sldw (vector unsigned __int128,
    vector unsigned __int128, const unsigned int);
vector signed __int128 vec_slo (vector signed __int128,
    vector signed char);
vector signed __int128 vec_slo (vector signed __int128,
    vector unsigned char);
vector unsigned __int128 vec_slo (vector unsigned __int128,
    vector signed char);
vector unsigned __int128 vec_slo (vector unsigned __int128,
    vector unsigned char);
vector signed __int128 vec_sro (vector signed __int128,
    vector signed char);
vector signed __int128 vec_sro (vector signed __int128,
    vector unsigned char);
vector unsigned __int128 vec_sro (vector unsigned __int128,
    vector signed char);
vector unsigned __int128 vec_sro (vector unsigned __int128,
    vector unsigned char);
vector signed __int128 vec_srl (vector signed __int128,
    vector unsigned char);
vector unsigned __int128 vec_srl (vector unsigned __int128,
    vector unsigned char);

```

7.13.24.6 PowerPC AltiVec/VSX Built-in Functions Available on Future ISA

The following additional built-in functions are available for the PowerPC family of processors, starting with Future ISA.

Future ISA of the PowerPC may add new instructions for accelerating AES algorithm. GCC provides support for these new instructions through the following built-in functions. The third argument to `__builtin_aes_encrypt_paired` and `__builtin_aes_decrypt_paired` must be a constant integer that is 0, 1 or 2. The values correspond to 128, 192 and 256 bit

AES encryption/decryption respectively. The third argument to `--builtin-galois-field-mult` must be a constant integer that is 0 or 1. The values correspond to gcm and xts variant respectively.

```
__vector_pair __builtin_aes_encrypt_paired (__vector_pair, __vector_pair,
                                           int);

__vector_pair __builtin_aes128_encrypt_paired (__vector_pair, __vector_pair);
__vector_pair __builtin_aes192_encrypt_paired (__vector_pair, __vector_pair);
__vector_pair __builtin_aes256_encrypt_paired (__vector_pair, __vector_pair);

__vector_pair __builtin_aes_decrypt_paired (__vector_pair, __vector_pair,
                                           int);

__vector_pair __builtin_aes128_decrypt_paired (__vector_pair, __vector_pair);
__vector_pair __builtin_aes192_decrypt_paired (__vector_pair, __vector_pair);
__vector_pair __builtin_aes256_decrypt_paired (__vector_pair, __vector_pair);

__vector_pair __builtin_aes_genlastkey_paired (__vector_pair, int);
__vector_pair __builtin_aes128_genlastkey_paired (__vector_pair);
__vector_pair __builtin_aes192_genlastkey_paired (__vector_pair);
__vector_pair __builtin_aes256_genlastkey_paired (__vector_pair);

vec_t __builtin_galois_field_mult (vec_t, vec_t, int);

vec_t __builtin_galois_field_mult_gcm (vec_t, vec_t);

vec_t __builtin_galois_field_mult_xts (vec_t, vec_t);
```

7.13.25 PowerPC Hardware Transactional Memory Built-in Functions

GCC provides two interfaces for accessing the Hardware Transactional Memory (HTM) instructions available on some of the PowerPC family of processors (eg, POWER8). The two interfaces come in a low level interface, consisting of built-in functions specific to PowerPC and a higher level interface consisting of inline functions that are common between PowerPC and S/390.

7.13.25.1 PowerPC HTM Low Level Built-in Functions

The following low level built-in functions are available with `-mhtm` or `-mcpu=CPU` where CPU is 'power8' or later. They all generate the machine instruction that is part of the name.

The HTM builtins (with the exception of `--builtin_tbegin`) return the full 4-bit condition register value set by their associated hardware instruction. The header file `htmintrin.h` defines some macros that can be used to decipher the return value. The `--builtin_tbegin` builtin returns a simple `true` or `false` value depending on whether a transaction was successfully started or not. The arguments of the builtins match exactly the type and order of the associated hardware instruction's operands, except for the `--builtin_tcheck` builtin,

which does not take any input arguments. Refer to the ISA manual for a description of each instruction's operands.

```
unsigned int __builtin_tbegin (unsigned int);
unsigned int __builtin_tend (unsigned int);

unsigned int __builtin_tabort (unsigned int);
unsigned int __builtin_tabortdc (unsigned int, unsigned int, unsigned int);
unsigned int __builtin_tabortdci (unsigned int, unsigned int, int);
unsigned int __builtin_tabortwc (unsigned int, unsigned int, unsigned int);
unsigned int __builtin_tabortwci (unsigned int, unsigned int, int);

unsigned int __builtin_tcheck (void);
unsigned int __builtin_treclaim (unsigned int);
unsigned int __builtin_trechpt (void);
unsigned int __builtin_tsr (unsigned int);
```

In addition to the above HTM built-ins, we have added built-ins for some common extended mnemonics of the HTM instructions:

```
unsigned int __builtin_tendall (void);
unsigned int __builtin_tresume (void);
unsigned int __builtin_tsuspend (void);
```

Note that the semantics of the above HTM builtins are required to mimic the locking semantics used for critical sections. Builtins that are used to create a new transaction or restart a suspended transaction must have lock acquisition like semantics while those builtins that end or suspend a transaction must have lock release like semantics. Specifically, this must mimic lock semantics as specified by C++11, for example: Lock acquisition is as-if an execution of `__atomic_exchange_n(&globallock,1,__ATOMIC_ACQUIRE)` that returns 0, and lock release is as-if an execution of `__atomic_store(&globallock,0,__ATOMIC_RELEASE)`, with `globallock` being an implicit implementation-defined lock used for all transactions. The HTM instructions associated with the builtins inherently provide the correct acquisition and release hardware barriers required. However, the compiler must also be prohibited from moving loads and stores across the builtins in a way that would violate their semantics. This has been accomplished by adding memory barriers to the associated HTM instructions (which is a conservative approach to provide acquire and release semantics). Earlier versions of the compiler did not treat the HTM instructions as memory barriers. A `__TM_FENCE__` macro has been added, which can be used to determine whether the current compiler treats HTM instructions as memory barriers or not. This allows the user to explicitly add memory barriers to their code when using an older version of the compiler.

The following set of built-in functions are available to gain access to the HTM specific special purpose registers.

```
unsigned long __builtin_get_texasr (void);
unsigned long __builtin_get_texasru (void);
unsigned long __builtin_get_tfhar (void);
unsigned long __builtin_get_tfiar (void);

void __builtin_set_texasr (unsigned long);
void __builtin_set_texasru (unsigned long);
void __builtin_set_tfhar (unsigned long);
void __builtin_set_tfiar (unsigned long);
```

Example usage of these low level built-in functions may look like:

```
#include <htmintrin.h>

int num_retries = 10;

while (1)
{
    if (__builtin_tbegin (0))
    {
        /* Transaction State Initiated. */
        if (is_locked (lock))
            __builtin_tabort (0);
        ... transaction code...
        __builtin_tend (0);
        break;
    }
    else
    {
        /* Transaction State Failed. Use locks if the transaction
           failure is "persistent" or we've tried too many times. */
        if (num_retries-- <= 0
            || _TEXASRU_FAILURE_PERSISTENT (__builtin_get_texasru ()))
        {
            acquire_lock (lock);
            ... non transactional fallback path...
            release_lock (lock);
            break;
        }
    }
}
```

One final built-in function has been added that returns the value of the 2-bit Transaction State field of the Machine Status Register (MSR) as stored in CRO.

```
unsigned long __builtin_ttest (void)
```

This built-in can be used to determine the current transaction state using the following code example:

```
#include <htmintrin.h>

unsigned char tx_state = _HTM_STATE (__builtin_ttest ());

if (tx_state == _HTM_TRANSACTIONAL)
{
    /* Code to use in transactional state. */
}
else if (tx_state == _HTM_NONTRANSACTIONAL)
{
    /* Code to use in non-transactional state. */
}
else if (tx_state == _HTM_SUSPENDED)
{
    /* Code to use in transaction suspended state. */
}
```

7.13.25.2 PowerPC HTM High Level Inline Functions

The following high level HTM interface is made available by including `<htmxlintrin.h>` and using `-mhtm` or `-mcpu=CPU` where CPU is 'power8' or later. This interface is common

between PowerPC and S/390, allowing users to write one HTM source implementation that can be compiled and executed on either system.

```

long __TM_simple_begin (void);
long __TM_begin (void* const TM_buff);
long __TM_end (void);
void __TM_abort (void);
void __TM_named_abort (unsigned char const code);
void __TM_resume (void);
void __TM_suspend (void);

long __TM_is_user_abort (void* const TM_buff);
long __TM_is_named_user_abort (void* const TM_buff, unsigned char *code);
long __TM_is_illegal (void* const TM_buff);
long __TM_is_footprint_exceeded (void* const TM_buff);
long __TM_nesting_depth (void* const TM_buff);
long __TM_is_nested_too_deep(void* const TM_buff);
long __TM_is_conflict(void* const TM_buff);
long __TM_is_failure_persistent(void* const TM_buff);
long __TM_failure_address(void* const TM_buff);
long long __TM_failure_code(void* const TM_buff);

```

Using these common set of HTM inline functions, we can create a more portable version of the HTM example in the previous section that will work on either PowerPC or S/390:

```

#include <htmxlintrin.h>

int num_retries = 10;
TM_buff_type TM_buff;

while (1)
{
    if (__TM_begin (TM_buff) == _HTM_TBEGIN_STARTED)
    {
        /* Transaction State Initiated. */
        if (is_locked (lock))
            __TM_abort ();
        ... transaction code...
        __TM_end ();
        break;
    }
    else
    {
        /* Transaction State Failed. Use locks if the transaction
           failure is "persistent" or we've tried too many times. */
        if (num_retries-- <= 0
            || __TM_is_failure_persistent (TM_buff))
        {
            acquire_lock (lock);
            ... non transactional fallback path...
            release_lock (lock);
            break;
        }
    }
}

```

7.13.26 PowerPC Atomic Memory Operation Functions

ISA 3.0 of the PowerPC added new atomic memory operation (amo) instructions. GCC provides support for these instructions in 64-bit environments. All of the functions are declared in the include file `amo.h`.

The functions supported are:

```
#include <amo.h>

uint32_t amo_lwat_add (uint32_t *, uint32_t);
uint32_t amo_lwat_xor (uint32_t *, uint32_t);
uint32_t amo_lwat_ior (uint32_t *, uint32_t);
uint32_t amo_lwat_and (uint32_t *, uint32_t);
uint32_t amo_lwat_umax (uint32_t *, uint32_t);
uint32_t amo_lwat_umin (uint32_t *, uint32_t);
uint32_t amo_lwat_swap (uint32_t *, uint32_t);
uint32_t amo_lwat_cas_neq (uint32_t *, uint32_t, uint32_t);
uint32_t amo_lwat_inc_eq (uint32_t *);
uint32_t amo_lwat_inc_bounded (uint32_t *);
uint32_t amo_lwat_dec_bounded (uint32_t *);

int32_t amo_lwat_sadd (int32_t *, int32_t);
int32_t amo_lwat_smax (int32_t *, int32_t);
int32_t amo_lwat_smin (int32_t *, int32_t);
int32_t amo_lwat_sswap (int32_t *, int32_t);
int32_t amo_lwat_scas_neq (int32_t *, int32_t, int32_t);
int32_t amo_lwat_sinc_eq (int32_t *);
int32_t amo_lwat_sinc_bounded (int32_t *);
int32_t amo_lwat_sdec_bounded (int32_t *);

uint64_t amo_ldat_add (uint64_t *, uint64_t);
uint64_t amo_ldat_xor (uint64_t *, uint64_t);
uint64_t amo_ldat_ior (uint64_t *, uint64_t);
uint64_t amo_ldat_and (uint64_t *, uint64_t);
uint64_t amo_ldat_umax (uint64_t *, uint64_t);
uint64_t amo_ldat_umin (uint64_t *, uint64_t);
uint64_t amo_ldat_swap (uint64_t *, uint64_t);
uint64_t amo_ldat_cas_neq (uint64_t *, uint64_t, uint64_t);
uint64_t amo_ldat_inc_eq (uint64_t *);
uint64_t amo_ldat_inc_bounded (uint64_t *);
uint64_t amo_ldat_dec_bounded (uint64_t *);

int64_t amo_ldat_sadd (int64_t *, int64_t);
int64_t amo_ldat_smax (int64_t *, int64_t);
int64_t amo_ldat_smin (int64_t *, int64_t);
int64_t amo_ldat_sswap (int64_t *, int64_t);
int64_t amo_ldat_scas_neq (int64_t *, int64_t, int64_t);
int64_t amo_ldat_sinc_eq (int64_t *);
int64_t amo_ldat_sinc_bounded (int64_t *);
int64_t amo_ldat_sdec_bounded (int64_t *);

void amo_stwat_add (uint32_t *, uint32_t);
void amo_stwat_xor (uint32_t *, uint32_t);
void amo_stwat_ior (uint32_t *, uint32_t);
void amo_stwat_and (uint32_t *, uint32_t);
void amo_stwat_umax (uint32_t *, uint32_t);
void amo_stwat_umin (uint32_t *, uint32_t);
void amo_stwat_twin (uint32_t *, uint32_t);
```

```

void amo_stwat_sadd (int32_t *, int32_t);
void amo_stwat_smax (int32_t *, int32_t);
void amo_stwat_smin (int32_t *, int32_t);
void amo_stwat_stwin (int32_t *, int32_t);

void amo_stdatt_add (uint64_t *, uint64_t);
void amo_stdatt_xor (uint64_t *, uint64_t);
void amo_stdatt_ior (uint64_t *, uint64_t);
void amo_stdatt_and (uint64_t *, uint64_t);
void amo_stdatt_umax (uint64_t *, uint64_t);
void amo_stdatt_umin (uint64_t *, uint64_t);
void amo_stdatt_twin (uint64_t *, uint64_t);

void amo_stdatt_sadd (int64_t *, int64_t);
void amo_stdatt_smax (int64_t *, int64_t);
void amo_stdatt_smin (int64_t *, int64_t);
void amo_stdatt_stwin (int64_t *, int64_t);

```

7.13.27 PowerPC Matrix-Multiply Assist Built-in Functions

ISA 3.1 of the PowerPC added new Matrix-Multiply Assist (MMA) instructions. GCC provides support for these instructions through the following built-in functions which are enabled with the `-mhma` option. The `vec_t` type below is defined to be a normal vector unsigned char type. The `uint2`, `uint4` and `uint8` parameters are 2-bit, 4-bit and 8-bit unsigned integer constants respectively. The compiler will verify that they are constants and that their values are within range.

The built-in functions supported are:

```

void __builtin_mma_xvi4ger8 (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvi8ger4 (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvi16ger2 (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvi16ger2s (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf16ger2 (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvbf16ger2 (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf32ger (__vector_quad *, vec_t, vec_t);

void __builtin_mma_xvi4ger8pp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvi8ger4pp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvi8ger4spp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvi16ger2pp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvi16ger2spp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf16ger2pp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf16ger2pn (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf16ger2np (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf16ger2nn (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvbf16ger2pp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvbf16ger2pn (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvbf16ger2np (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvbf16ger2nn (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf32gerpp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf32gerpn (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf32gernp (__vector_quad *, vec_t, vec_t);
void __builtin_mma_xvf32gernn (__vector_quad *, vec_t, vec_t);

void __builtin_mma_pmxvi4ger8 (__vector_quad *, vec_t, vec_t, uint4, uint4, uint8);
void __builtin_mma_pmxvi4ger8pp (__vector_quad *, vec_t, vec_t, uint4, uint4, uint8);

```

```

void __builtin_mma_pmxvi8ger4 (__vector_quad *, vec_t, vec_t, uint4, uint4, uint4);
void __builtin_mma_pmxvi8ger4pp (__vector_quad *, vec_t, vec_t, uint4, uint4, uint4);
void __builtin_mma_pmxvi8ger4spp (__vector_quad *, vec_t, vec_t, uint4, uint4, uint4);

void __builtin_mma_pmxvi16ger2 (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvi16ger2s (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvf16ger2 (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvbf16ger2 (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);

void __builtin_mma_pmxvi16ger2pp (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvi16ger2spp (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvf16ger2pp (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvf16ger2pn (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvf16ger2np (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvf16ger2nn (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvbf16ger2pp (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvbf16ger2pn (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvbf16ger2np (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);
void __builtin_mma_pmxvbf16ger2nn (__vector_quad *, vec_t, vec_t, uint4, uint4, uint2);

void __builtin_mma_pmxvf32ger (__vector_quad *, vec_t, vec_t, uint4, uint4);
void __builtin_mma_pmxvf32gerpp (__vector_quad *, vec_t, vec_t, uint4, uint4);
void __builtin_mma_pmxvf32gerpn (__vector_quad *, vec_t, vec_t, uint4, uint4);
void __builtin_mma_pmxvf32gernp (__vector_quad *, vec_t, vec_t, uint4, uint4);
void __builtin_mma_pmxvf32gernn (__vector_quad *, vec_t, vec_t, uint4, uint4);

void __builtin_mma_xvf64ger (__vector_quad *, __vector_pair, vec_t);
void __builtin_mma_xvf64gerpp (__vector_quad *, __vector_pair, vec_t);
void __builtin_mma_xvf64gerpn (__vector_quad *, __vector_pair, vec_t);
void __builtin_mma_xvf64gernp (__vector_quad *, __vector_pair, vec_t);
void __builtin_mma_xvf64gernn (__vector_quad *, __vector_pair, vec_t);

void __builtin_mma_pmxvf64ger (__vector_quad *, __vector_pair, vec_t, uint4, uint2);
void __builtin_mma_pmxvf64gerpp (__vector_quad *, __vector_pair, vec_t, uint4, uint2);
void __builtin_mma_pmxvf64gerpn (__vector_quad *, __vector_pair, vec_t, uint4, uint2);
void __builtin_mma_pmxvf64gernp (__vector_quad *, __vector_pair, vec_t, uint4, uint2);
void __builtin_mma_pmxvf64gernn (__vector_quad *, __vector_pair, vec_t, uint4, uint2);

void __builtin_mma_xxmtacc (__vector_quad *);
void __builtin_mma_xxmfacc (__vector_quad *);
void __builtin_mma_xxsetaccz (__vector_quad *);

void __builtin_mma_build_acc (__vector_quad *, vec_t, vec_t, vec_t, vec_t);
void __builtin_mma_disassemble_acc (void *, __vector_quad *);

void __builtin_vsx_build_pair (__vector_pair *, vec_t, vec_t);
void __builtin_vsx_disassemble_pair (void *, __vector_pair *);

vec_t __builtin_vsx_xvcvspbf16 (vec_t);
vec_t __builtin_vsx_xvcvbf16spn (vec_t);

__vector_pair __builtin_vsx_lxvp (size_t, __vector_pair *);
void __builtin_vsx_stxvp (__vector_pair, size_t, __vector_pair *);

```

7.13.28 PRU Built-in Functions

GCC provides a couple of special builtin functions to aid in utilizing special PRU instructions.

The built-in functions supported are:

`void __delay_cycles (constant long long cycles)` [Built-in Function]

This inserts an instruction sequence that takes exactly *cycles* cycles (between 0 and 0xffffffff) to complete. The inserted sequence may use jumps, loops, or no-ops, and does not interfere with any other instructions. Note that *cycles* must be a compile-time constant integer - that is, you must pass a number, not a variable that may be optimized to a constant later. The number of cycles delayed by this builtin is exact.

`void __halt (void)` [Built-in Function]

This inserts a HALT instruction to stop processor execution.

`unsigned int __lmbd (unsigned int wordval, unsigned int bitval)` [Built-in Function]

This inserts LMBD instruction to calculate the left-most bit with value *bitval* in value *wordval*. Only the least significant bit of *bitval* is taken into account.

7.13.29 RISC-V Built-in Functions

These built-in functions are available for the RISC-V family of processors.

RISC-V intrinsic headers define macros of the form `__riscv_intrinsic_extension` with the value 1 when GCC supports intrinsics for *extension*. These macros indicate compiler support for the intrinsic API and are independent of whether the corresponding ISA extension is enabled for the current compilation unit. Include `riscv_bitmanip.h` for scalar bit-manipulation intrinsics, `riscv_crypto.h` for scalar cryptography intrinsics, `riscv_vector.h` for vector intrinsics, and vendor intrinsic headers for vendor intrinsic extensions.

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

Returns the value that is currently set in the ‘tp’ register.

`void __builtin_riscv_pause (void)` [Built-in Function]

Generates the `pause` (hint) machine instruction. If the target implements the Zihint-pause extension, it indicates that the current hart should be temporarily paused or slowed down.

7.13.30 RISC-V Vector Intrinsics

GCC supports vector intrinsics as specified in the ratified version 1.0 of the RISC-V vector intrinsic specification, which is available from the repository’s release page: <https://github.com/riscv-non-isa/rvv-intrinsic-doc/releases/tag/v1.0-ratified>. All of these functions are declared in the include file `riscv_vector.h`.

7.13.31 CORE-V Built-in Functions

For more information on all CORE-V built-ins, please see <https://github.com/openhwgroup/core-v-sw/blob/master/specifications/corev-builtin-spec.md>

These built-in functions are available for the CORE-V MAC machine architecture. For more information on CORE-V built-ins, please see <https://github.com/openhwgroup/core-v-sw/blob/master/specifications/corev-builtin-spec.md#listing-of-multiply-accumulate-builtins-xcvmac>.

<code>int32_t __builtin_riscv_cv_mac_mac (int32_t, int32_t, int32_t)</code>	[Built-in Function]
Generated assembler <code>cv.mac</code>	
<code>int32_t __builtin_riscv_cv_mac_msu (int32_t, int32_t, int32_t)</code>	[Built-in Function]
Generates the <code>cv.msu</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_muluN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.muluN</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_mulhhuN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.mulhhuN</code> machine instruction.	
<code>int32_t __builtin_riscv_cv_mac_mulsN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.mulsN</code> machine instruction.	
<code>int32_t __builtin_riscv_cv_mac_mulhhsN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.mulhhsN</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_muluRN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.muluRN</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_mulhhuRN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.mulhhuRN</code> machine instruction.	
<code>int32_t __builtin_riscv_cv_mac_mulsRN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.mulsRN</code> machine instruction.	
<code>int32_t __builtin_riscv_cv_mac_mulhhsRN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.mulhhsRN</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_macuN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.macuN</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_machhuN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.machhuN</code> machine instruction.	
<code>int32_t __builtin_riscv_cv_mac_macsN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.macsN</code> machine instruction.	

<code>int32_t __builtin_riscv_cv_mac_machhsN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.machhsN</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_macuRN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.macuRN</code> machine instruction.	
<code>uint32_t __builtin_riscv_cv_mac_machhuRN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.machhuRN</code> machine instruction.	
<code>int32_t __builtin_riscv_cv_mac_macsRN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.macsRN</code> machine instruction.	
<code>int32_t __builtin_riscv_cv_mac_machhsRN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generates the <code>cv.machhsRN</code> machine instruction.	

These built-in functions are available for the CORE-V ALU machine architecture. For more information on CORE-V built-ins, please see <https://github.com/openhwgroup/core-v-sw/blob/master/specifications/corev-builtin-spec.md#listing-of-miscellaneous-alu-builtins-xcvalu>

<code>int __builtin_riscv_cv_alu_slet (int32_t, int32_t)</code>	[Built-in Function]
Generated assembler <code>cv.slet</code>	
<code>int __builtin_riscv_cv_alu_sletu (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sletu</code>	
<code>int32_t __builtin_riscv_cv_alu_min (int32_t, int32_t)</code>	[Built-in Function]
Generated assembler <code>cv.min</code>	
<code>uint32_t __builtin_riscv_cv_alu_minu (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.minu</code>	
<code>int32_t __builtin_riscv_cv_alu_max (int32_t, int32_t)</code>	[Built-in Function]
Generated assembler <code>cv.max</code>	
<code>uint32_t __builtin_riscv_cv_alu_maxu (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.maxu</code>	
<code>int32_t __builtin_riscv_cv_alu_exths (int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.exths</code>	

<code>uint32_t __builtin_riscv_cv_alu_exthz (uint16_t)</code>	[Built-in Function]
Generated assembler <code>cv.exthz</code>	
<code>int32_t __builtin_riscv_cv_alu_extbs (int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.extbs</code>	
<code>uint32_t __builtin_riscv_cv_alu_extbz (uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.extbz</code>	
<code>int32_t __builtin_riscv_cv_alu_clip (int32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.clip</code> if the <code>uint32_t</code> operand is a constant and an exact power of 2. Generated assembler <code>cv.clipr</code> if the it is a register.	
<code>uint32_t __builtin_riscv_cv_alu_clipu (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.clipu</code> if the <code>uint32_t</code> operand is a constant and an exact power of 2. Generated assembler <code>cv.clipur</code> if the it is a register.	
<code>int32_t __builtin_riscv_cv_alu_addN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.addN</code> if the <code>uint8_t</code> operand is a constant and in the range $0 \leq \text{shift} \leq 31$. Generated assembler <code>cv.addNr</code> if the it is a register.	
<code>uint32_t __builtin_riscv_cv_alu_adduN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.adduN</code> if the <code>uint8_t</code> operand is a constant and in the range $0 \leq \text{shift} \leq 31$. Generated assembler <code>cv.adduNr</code> if the it is a register.	
<code>int32_t __builtin_riscv_cv_alu_addrN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.addrN</code> if the <code>uint8_t</code> operand is a constant and in the range $0 \leq \text{shift} \leq 31$. Generated assembler <code>cv.addrNr</code> if the it is a register.	
<code>uint32_t __builtin_riscv_cv_alu_adduRN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.adduRN</code> if the <code>uint8_t</code> operand is a constant and in the range $0 \leq \text{shift} \leq 31$. Generated assembler <code>cv.adduRNr</code> if the it is a register.	
<code>int32_t __builtin_riscv_cv_alu_subN (int32_t, int32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.subN</code> if the <code>uint8_t</code> operand is a constant and in the range $0 \leq \text{shift} \leq 31$. Generated assembler <code>cv.subNr</code> if the it is a register.	
<code>uint32_t __builtin_riscv_cv_alu_subuN (uint32_t, uint32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.subuN</code> if the <code>uint8_t</code> operand is a constant and in the range $0 \leq \text{shift} \leq 31$. Generated assembler <code>cv.subuNr</code> if the it is a register.	

`int32_t __builtin_riscv_cv_alu_subRN (int32_t, [Built-in Function]
int32_t, uint8_t)`

Generated assembler `cv.subRN` if the `uint8_t` operand is a constant and in the range `0 <= shift <= 31`. Generated assembler `cv.subRNr` if the it is a register.

`uint32_t __builtin_riscv_cv_alu_subuRN (uint32_t, [Built-in Function]
uint32_t, uint8_t)`

Generated assembler `cv.subuRN` if the `uint8_t` operand is a constant and in the range `0 <= shift <= 31`. Generated assembler `cv.subuRNr` if the it is a register.

These built-in functions are available for the CORE-V Event Load machine architecture. For more information on CORE-V ELW builtins, please see <https://github.com/openhwgroup/core-v-sw/blob/master/specifications/corev-builtin-spec.md#listing-of-event-load-word-builtins-xcvelw>

`uint32_t __builtin_riscv_cv_elw_elw (uint32_t *) [Built-in Function]`
Generated assembler `cv.elw`

These built-in functions are available for the CORE-V SIMD machine architecture. For more information on CORE-V SIMD built-ins, please see <https://github.com/openhwgroup/core-v-sw/blob/master/specifications/corev-builtin-spec.md#listing-of-pulp-816-bit-simd-builtins-xcvsimd>

`uint32_t __builtin_riscv_cv_simd_add_h (uint32_t, [Built-in Function]
uint32_t, uint4_t)`
Generated assembler `cv.add.h`

`uint32_t __builtin_riscv_cv_simd_add_b (uint32_t, [Built-in Function]
uint32_t)`
Generated assembler `cv.add.b`

`uint32_t __builtin_riscv_cv_simd_add_sc_h (uint32_t, [Built-in Function]
int16_t)`
Generated assembler `cv.add.sc.h`

`uint32_t __builtin_riscv_cv_simd_add_sc_h (uint32_t, [Built-in Function]
int6_t)`
Generated assembler `cv.add.sci.h`

`uint32_t __builtin_riscv_cv_simd_add_sc_b (uint32_t, [Built-in Function]
int8_t)`
Generated assembler `cv.add.sc.b`

`uint32_t __builtin_riscv_cv_simd_add_sc_b (uint32_t, [Built-in Function]
int6_t)`
Generated assembler `cv.add.sci.b`

`uint32_t __builtin_riscv_cv_simd_sub_h (uint32_t, [Built-in Function]
uint32_t, uint4_t)`
Generated assembler `cv.sub.h`

<code>uint32_t __builtin_riscv_cv_simd_sub_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sub_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sub_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sub_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sub_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_avg_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.avg.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_avg_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.avg.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_avg_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.avg.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_avg_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.avg.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_avg_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.avg.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_avg_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.avg.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_avgu_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.avgu.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_avgu_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.avgu.b</code>	

<code>uint32_t __builtin_riscv_cv_simd_avgu_sc_h</code> <code>(uint32_t, uint16_t)</code> Generated assembler <code>cv.avgu.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_avgu_sc_h</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.avgu.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_avgu_sc_b</code> <code>(uint32_t, uint8_t)</code> Generated assembler <code>cv.avgu.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_avgu_sc_b</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.avgu.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_min_h</code> <code>(uint32_t,</code> <code>uint32_t)</code> Generated assembler <code>cv.min.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_min_b</code> <code>(uint32_t,</code> <code>uint32_t)</code> Generated assembler <code>cv.min.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_min_sc_h</code> <code>(uint32_t,</code> <code>int16_t)</code> Generated assembler <code>cv.min.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_min_sc_h</code> <code>(uint32_t,</code> <code>int6_t)</code> Generated assembler <code>cv.min.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_min_sc_b</code> <code>(uint32_t,</code> <code>int8_t)</code> Generated assembler <code>cv.min.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_min_sc_b</code> <code>(uint32_t,</code> <code>int6_t)</code> Generated assembler <code>cv.min.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_minu_h</code> <code>(uint32_t,</code> <code>uint32_t)</code> Generated assembler <code>cv.minu.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_minu_b</code> <code>(uint32_t,</code> <code>uint32_t)</code> Generated assembler <code>cv.minu.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_minu_sc_h</code> <code>(uint32_t, uint16_t)</code> Generated assembler <code>cv.minu.sc.h</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_minu_sc_h</code> (<code>uint32_t</code> , <code>uint6_t</code>) Generated assembler <code>cv.minu.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_minu_sc_b</code> (<code>uint32_t</code> , <code>uint8_t</code>) Generated assembler <code>cv.minu.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_minu_sc_b</code> (<code>uint32_t</code> , <code>uint6_t</code>) Generated assembler <code>cv.minu.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_max_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.max.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_max_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.max.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_max_sc_h</code> (<code>uint32_t</code> , <code>int16_t</code>) Generated assembler <code>cv.max.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_max_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.max.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_max_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.max.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_max_sc_b</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.max.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_maxu_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.maxu.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_maxu_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.maxu.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_maxu_sc_h</code> (<code>uint32_t</code> , <code>uint16_t</code>) Generated assembler <code>cv.maxu.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_maxu_sc_h</code> (<code>uint32_t</code> , <code>uint6_t</code>) Generated assembler <code>cv.maxu.sci.h</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_maxu_sc_b</code> (<code>uint32_t</code> , <code>uint8_t</code>) Generated assembler <code>cv.maxu.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_maxu_sc_b</code> (<code>uint32_t</code> , <code>uint6_t</code>) Generated assembler <code>cv.maxu.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_srl_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.srl.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_srl_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.srl.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_srl_sc_h</code> (<code>uint32_t</code> , <code>int16_t</code>) Generated assembler <code>cv.srl.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_srl_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.srl.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_srl_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.srl.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_srl_sc_b</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.srl.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sra_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.sra.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sra_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.sra.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sra_sc_h</code> (<code>uint32_t</code> , <code>int16_t</code>) Generated assembler <code>cv.sra.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sra_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.sra.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sra_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.sra.sc.b</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_sra_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.sra.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sll_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sll.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sll_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sll.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sll_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.sll.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sll_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.sll.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sll_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.sll.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sll_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.sll.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_or_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.or.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_or_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.or.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_or_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.or.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_or_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.or.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_or_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.or.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_or_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.or.sci.b</code>	

<code>uint32_t __builtin_riscv_cv_simd_xor_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.xor.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_xor_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.xor.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_xor_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.xor.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_xor_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.xor.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_xor_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.xor.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_xor_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.xor.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_and_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.and.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_and_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.and.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_and_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.and.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_and_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.and.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_and_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.and.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_and_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.and.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_abs_h (uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.abs.h</code>	

<code>uint32_t __builtin_riscv_cv_simd_abs_b (uint32_t)</code> Generated assembler <code>cv.abs.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotup_h (uint32_t, uint32_t)</code> Generated assembler <code>cv.dotup.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotup_b (uint32_t, uint32_t)</code> Generated assembler <code>cv.dotup.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotup_sc_h (uint32_t, uint16_t)</code> Generated assembler <code>cv.dotup.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotup_sc_h (uint32_t, uint6_t)</code> Generated assembler <code>cv.dotup.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotup_sc_b (uint32_t, uint8_t)</code> Generated assembler <code>cv.dotup.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotup_sc_b (uint32_t, uint6_t)</code> Generated assembler <code>cv.dotup.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotusp_h (uint32_t, uint32_t)</code> Generated assembler <code>cv.dotusp.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotusp_b (uint32_t, uint32_t)</code> Generated assembler <code>cv.dotusp.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotusp_sc_h (uint32_t, int16_t)</code> Generated assembler <code>cv.dotusp.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotusp_sc_h (uint32_t, int6_t)</code> Generated assembler <code>cv.dotusp.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotusp_sc_b (uint32_t, int8_t)</code> Generated assembler <code>cv.dotusp.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_dotusp_sc_b (uint32_t, int6_t)</code> Generated assembler <code>cv.dotusp.sci.b</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_dotsp_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.dotsp.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_dotsp_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.dotsp.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_dotsp_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.dotsp.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_dotsp_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.dotsp.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_dotsp_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.dotsp.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_dotsp_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.dotsp.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sdotup_h (uint32_t, uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sdotup.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sdotup_b (uint32_t, uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sdotup.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sdotup_sc_h (uint32_t, uint16_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sdotup.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sdotup_sc_h (uint32_t, uint6_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sdotup.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_sdotup_sc_b (uint32_t, uint8_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sdotup.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sdotup_sc_b (uint32_t, uint6_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sdotup.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_sdotusp_h (uint32_t, uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.sdotusp.h</code>	

<code>uint32_t __builtin_riscv_cv_simd_sdotusp_b</code> <code>(uint32_t, uint32_t, uint32_t)</code> Generated assembler <code>cv.sdotusp.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotusp_sc_h</code> <code>(uint32_t, int16_t, uint32_t)</code> Generated assembler <code>cv.sdotusp.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotusp_sc_h</code> <code>(uint32_t, int6_t, uint32_t)</code> Generated assembler <code>cv.sdotusp.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotusp_sc_b</code> <code>(uint32_t, int8_t, uint32_t)</code> Generated assembler <code>cv.sdotusp.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotusp_sc_b</code> <code>(uint32_t, int6_t, uint32_t)</code> Generated assembler <code>cv.sdotusp.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotsp_h</code> <code>(uint32_t,</code> <code>uint32_t, uint32_t)</code> Generated assembler <code>cv.sdotsp.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotsp_b</code> <code>(uint32_t,</code> <code>uint32_t, uint32_t)</code> Generated assembler <code>cv.sdotsp.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotsp_sc_h</code> <code>(uint32_t, int16_t, uint32_t)</code> Generated assembler <code>cv.sdotsp.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotsp_sc_h</code> <code>(uint32_t, int6_t, uint32_t)</code> Generated assembler <code>cv.sdotsp.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotsp_sc_b</code> <code>(uint32_t, int8_t, uint32_t)</code> Generated assembler <code>cv.sdotsp.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_sdotsp_sc_b</code> <code>(uint32_t, int6_t, uint32_t)</code> Generated assembler <code>cv.sdotsp.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_extract_h</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.extract.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_extract_b</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.extract.b</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_extractu_h</code> (<code>uint32_t</code> , <code>uint6_t</code>) Generated assembler <code>cv.extractu.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_extractu_b</code> (<code>uint32_t</code> , <code>uint6_t</code>) Generated assembler <code>cv.extractu.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_insert_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.insert.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_insert_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.insert.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shuffle_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.shuffle.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shuffle_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.shuffle.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shuffle_sci_h</code> (<code>uint32_t</code> , <code>uint4_t</code>) Generated assembler <code>cv.shuffle.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shufflei0_sci_b</code> (<code>uint32_t</code> , <code>uint4_t</code>) Generated assembler <code>cv.shufflei0.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shufflei1_sci_b</code> (<code>uint32_t</code> , <code>uint4_t</code>) Generated assembler <code>cv.shufflei1.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shufflei2_sci_b</code> (<code>uint32_t</code> , <code>uint4_t</code>) Generated assembler <code>cv.shufflei2.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shufflei3_sci_b</code> (<code>uint32_t</code> , <code>uint4_t</code>) Generated assembler <code>cv.shufflei3.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shuffle2_h</code> (<code>uint32_t</code> , <code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.shuffle2.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_shuffle2_b</code> (<code>uint32_t</code> , <code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.shuffle2.b</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_packlo_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.pack</code>	
<code>uint32_t __builtin_riscv_cv_simd_packhi_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.pack.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_packhi_b (uint32_t, uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.packhi.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_packlo_b (uint32_t, uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.packlo.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpeq_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpeq.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpeq_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpeq.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpeq_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpeq.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpeq_sc_h (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpeq.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpeq_sc_b (uint32_t, int8_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpeq.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpeq_sc_b (uint32_t, int6_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpeq.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpne_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpne.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpne_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpne.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpne_sc_h (uint32_t, int16_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpne.sc.h</code>	

<code>uint32_t __builtin_riscv_cv_simd_cmpne_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmpne.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpne_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.cmpne.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpne_sc_b</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmpne.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgt_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmpgt.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgt_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmpgt.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgt_sc_h</code> (<code>uint32_t</code> , <code>int16_t</code>) Generated assembler <code>cv.cmpgt.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgt_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmpgt.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgt_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.cmpgt.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgt_sc_b</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmpgt.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpge_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmpge.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpge_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmpge.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpge_sc_h</code> (<code>uint32_t</code> , <code>int16_t</code>) Generated assembler <code>cv.cmpge.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpge_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmpge.sci.h</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_cmpge_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.cmpge.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpge_sc_b</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmpge.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmplt_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmplt.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmplt_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmplt.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmplt_sc_h</code> (<code>uint32_t</code> , <code>int16_t</code>) Generated assembler <code>cv.cmplt.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmplt_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmplt.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmplt_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.cmplt.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmplt_sc_b</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmplt.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmple_h</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmple.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmple_b</code> (<code>uint32_t</code> , <code>uint32_t</code>) Generated assembler <code>cv.cmple.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmple_sc_h</code> (<code>uint32_t</code> , <code>int16_t</code>) Generated assembler <code>cv.cmple.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmple_sc_h</code> (<code>uint32_t</code> , <code>int6_t</code>) Generated assembler <code>cv.cmple.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmple_sc_b</code> (<code>uint32_t</code> , <code>int8_t</code>) Generated assembler <code>cv.cmple.sc.b</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_cmple_sc_b</code> <code>(uint32_t, int6_t)</code> Generated assembler <code>cv.cmple.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgtu_h</code> (<code>uint32_t,</code> <code>uint32_t</code>) Generated assembler <code>cv.cmpgtu.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgtu_b</code> (<code>uint32_t,</code> <code>uint32_t</code>) Generated assembler <code>cv.cmpgtu.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgtu_sc_h</code> <code>(uint32_t, uint16_t)</code> Generated assembler <code>cv.cmpgtu.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgtu_sc_h</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.cmpgtu.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgtu_sc_b</code> <code>(uint32_t, uint8_t)</code> Generated assembler <code>cv.cmpgtu.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgtu_sc_b</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.cmpgtu.sci.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgeu_h</code> (<code>uint32_t,</code> <code>uint32_t</code>) Generated assembler <code>cv.cmpgeu.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgeu_b</code> (<code>uint32_t,</code> <code>uint32_t</code>) Generated assembler <code>cv.cmpgeu.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgeu_sc_h</code> <code>(uint32_t, uint16_t)</code> Generated assembler <code>cv.cmpgeu.sc.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgeu_sc_h</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.cmpgeu.sci.h</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgeu_sc_b</code> <code>(uint32_t, uint8_t)</code> Generated assembler <code>cv.cmpgeu.sc.b</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cmpgeu_sc_b</code> <code>(uint32_t, uint6_t)</code> Generated assembler <code>cv.cmpgeu.sci.b</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_cmpltu_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpltu.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpltu_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpltu.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpltu_sc_h (uint32_t, uint16_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpltu.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpltu_sc_h (uint32_t, uint6_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpltu.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpltu_sc_b (uint32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpltu.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpltu_sc_b (uint32_t, uint6_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpltu.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpleu_h (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpleu.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpleu_b (uint32_t, uint32_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpleu.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpleu_sc_h (uint32_t, uint16_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpleu.sc.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpleu_sc_h (uint32_t, uint6_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpleu.sci.h</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpleu_sc_b (uint32_t, uint8_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpleu.sc.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cmpleu_sc_b (uint32_t, uint6_t)</code>	[Built-in Function]
Generated assembler <code>cv.cmpleu.sci.b</code>	
<code>uint32_t __builtin_riscv_cv_simd_cplxmul_r (uint32_t, uint32_t, uint32_t, uint4_t)</code>	[Built-in Function]
Generated assembler <code>cv.cplxmul.r</code>	

<code>uint32_t __builtin_riscv_cv_simd_cplxmul_i</code> <code>(uint32_t, uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.cplxmul.i</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cplxmul_r</code> <code>(uint32_t, uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.cplxmul.r.div2</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cplxmul_i</code> <code>(uint32_t, uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.cplxmul.i.div2</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cplxmul_r</code> <code>(uint32_t, uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.cplxmul.r.div4</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cplxmul_i</code> <code>(uint32_t, uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.cplxmul.i.div4</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cplxmul_r</code> <code>(uint32_t, uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.cplxmul.r.div8</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cplxmul_i</code> <code>(uint32_t, uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.cplxmul.i.div8</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_cplxconj (uint32_t)</code> Generated assembler <code>cv.cplxconj</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_subrotmj (uint32_t,</code> <code>uint32_t, uint4_t)</code> Generated assembler <code>cv.subrotmj</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_subrotmj (uint32_t,</code> <code>uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.subrotmj.div2</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_subrotmj (uint32_t,</code> <code>uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.subrotmj.div4</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_subrotmj (uint32_t,</code> <code>uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.subrotmj.div8</code>	[Built-in Function]
<code>uint32_t __builtin_riscv_cv_simd_add_h (uint32_t,</code> <code>uint32_t, uint32_t, uint4_t)</code> Generated assembler <code>cv.add.div2</code>	[Built-in Function]

<code>uint32_t __builtin_riscv_cv_simd_add_h (uint32_t, uint32_t, uint32_t, uint4_t)</code>	[Built-in Function]
Generated assembler <code>cv.add.div4</code>	
<code>uint32_t __builtin_riscv_cv_simd_add_h (uint32_t, uint32_t, uint32_t, uint4_t)</code>	[Built-in Function]
Generated assembler <code>cv.add.div8</code>	
<code>uint32_t __builtin_riscv_cv_simd_sub_h (uint32_t, uint32_t, uint32_t, uint4_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.div2</code>	
<code>uint32_t __builtin_riscv_cv_simd_sub_h (uint32_t, uint32_t, uint32_t, uint4_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.div4</code>	
<code>uint32_t __builtin_riscv_cv_simd_sub_h (uint32_t, uint32_t, uint32_t, uint4_t)</code>	[Built-in Function]
Generated assembler <code>cv.sub.div8</code>	

7.13.32 RX Built-in Functions

GCC supports some of the RX instructions which cannot be expressed in the C programming language via the use of built-in functions. The following functions are supported:

<code>void __builtin_rx_brk (void)</code>	[Built-in Function]
Generates the <code>brk</code> machine instruction.	
<code>void __builtin_rx_clrpsw (int)</code>	[Built-in Function]
Generates the <code>clrpsw</code> machine instruction to clear the specified bit in the processor status word.	
<code>void __builtin_rx_int (int)</code>	[Built-in Function]
Generates the <code>int</code> machine instruction to generate an interrupt with the specified value.	
<code>void __builtin_rx_machi (int, int)</code>	[Built-in Function]
Generates the <code>machi</code> machine instruction to add the result of multiplying the top 16 bits of the two arguments into the accumulator.	
<code>void __builtin_rx_maclo (int, int)</code>	[Built-in Function]
Generates the <code>maclo</code> machine instruction to add the result of multiplying the bottom 16 bits of the two arguments into the accumulator.	
<code>void __builtin_rx_mulhi (int, int)</code>	[Built-in Function]
Generates the <code>mulhi</code> machine instruction to place the result of multiplying the top 16 bits of the two arguments into the accumulator.	
<code>void __builtin_rx_mullo (int, int)</code>	[Built-in Function]
Generates the <code>mullo</code> machine instruction to place the result of multiplying the bottom 16 bits of the two arguments into the accumulator.	

`int __builtin_rx_mvfacchi (void)` [Built-in Function]
 Generates the `mvfacchi` machine instruction to read the top 32 bits of the accumulator.

`int __builtin_rx_mvfacmi (void)` [Built-in Function]
 Generates the `mvfacmi` machine instruction to read the middle 32 bits of the accumulator.

`int __builtin_rx_mvfc (int)` [Built-in Function]
 Generates the `mvfc` machine instruction which reads the control register specified in its argument and returns its value.

`void __builtin_rx_mvtachi (int)` [Built-in Function]
 Generates the `mvtachi` machine instruction to set the top 32 bits of the accumulator.

`void __builtin_rx_mvtaclo (int)` [Built-in Function]
 Generates the `mvtaclo` machine instruction to set the bottom 32 bits of the accumulator.

`void __builtin_rx_mvtc (int reg, int val)` [Built-in Function]
 Generates the `mvtc` machine instruction which sets control register number `reg` to `val`.

`void __builtin_rx_mvtipl (int)` [Built-in Function]
 Generates the `mvtipl` machine instruction set the interrupt priority level.

`void __builtin_rx_racw (int)` [Built-in Function]
 Generates the `racw` machine instruction to round the accumulator according to the specified mode.

`int __builtin_rx_revw (int)` [Built-in Function]
 Generates the `revw` machine instruction which swaps the bytes in the argument so that bits 0–7 now occupy bits 8–15 and vice versa, and also bits 16–23 occupy bits 24–31 and vice versa.

`void __builtin_rx_rmpa (void)` [Built-in Function]
 Generates the `rmpa` machine instruction which initiates a repeated multiply and accumulate sequence.

`void __builtin_rx_round (float)` [Built-in Function]
 Generates the `round` machine instruction which returns the floating-point argument rounded according to the current rounding mode set in the floating-point status word register.

`int __builtin_rx_sat (int)` [Built-in Function]
 Generates the `sat` machine instruction which returns the saturated value of the argument.

`void __builtin_rx_setpsw (int)` [Built-in Function]
 Generates the `setpsw` machine instruction to set the specified bit in the processor status word.

`void __builtin_rx_wait (void)` [Built-in Function]
 Generates the `wait` machine instruction.

7.13.33 S/390 System z Built-in Functions

int __builtin_tbegin (void*) [Built-in Function]

Generates the `tbegin` machine instruction starting a non-constrained hardware transaction. If the parameter is non-NULL the memory area is used to store the transaction diagnostic buffer and will be passed as first operand to `tbegin`. This buffer can be defined using the `struct __htm_tdb` C struct defined in `htmintrin.h` and must reside on a double-word boundary. The second `tbegin` operand is set to `0xff0c`. This enables save/restore of all GPRs and disables aborts for FPR and AR manipulations inside the transaction body. The condition code set by the `tbegin` instruction is returned as integer value. The `tbegin` instruction by definition overwrites the content of all FPRs. The compiler will generate code which saves and restores the FPRs. For soft-float code it is recommended to use the `*_nofloat` variant. In order to prevent a TDB from being written it is required to pass a constant zero value as parameter. Passing a zero value through a variable is not sufficient. Although modifications of access registers inside the transaction will not trigger an transaction abort it is not supported to actually modify them. Access registers do not get saved when entering a transaction. They will have undefined state when reaching the abort code.

Macros for the possible return codes of `tbegin` are defined in the `htmintrin.h` header file:

_HTM_TBEGIN_STARTED [Macro]

`tbegin` has been executed as part of normal processing. The transaction body is supposed to be executed.

_HTM_TBEGIN_INDETERMINATE [Macro]

The transaction was aborted due to an indeterminate condition which might be persistent.

_HTM_TBEGIN_TRANSIENT [Macro]

The transaction aborted due to a transient failure. The transaction should be re-executed in that case.

_HTM_TBEGIN_PERSISTENT [Macro]

The transaction aborted due to a persistent failure. Re-execution under same circumstances will not be productive.

_HTM_FIRST_USER_ABORT_CODE [Macro]

The `_HTM_FIRST_USER_ABORT_CODE` defined in `htmintrin.h` specifies the first abort code which can be used for `__builtin_tabort`. Values below this threshold are reserved for machine use.

struct __htm_tdb [Data type]

The `struct __htm_tdb` defined in `htmintrin.h` describes the structure of the transaction diagnostic block as specified in the Principles of Operation manual chapter 5-91.

int __builtin_tbegin_nofloat (void*) [Built-in Function]

Same as `__builtin_tbegin` but without FPR saves and restores. Using this variant in code making use of FPRs will leave the FPRs in undefined state when entering the transaction abort handler code.

- int __builtin_tbegin_retry (void*, int)** [Built-in Function]
 In addition to `__builtin_tbegin` a loop for transient failures is generated. If `tbegin` returns a condition code of 2 the transaction will be retried as often as specified in the second argument. The `perform processor assist` instruction is used to tell the CPU about the number of fails so far.
- int __builtin_tbegin_retry_nofloat (void*, int)** [Built-in Function]
 Same as `__builtin_tbegin_retry` but without FPR saves and restores. Using this variant in code making use of FPRs will leave the FPRs in undefined state when entering the transaction abort handler code.
- void __builtin_tbegininc (void)** [Built-in Function]
 Generates the `tbegininc` machine instruction starting a constrained hardware transaction. The second operand is set to `0xff08`.
- int __builtin_tend (void)** [Built-in Function]
 Generates the `tend` machine instruction finishing a transaction and making the changes visible to other threads. The condition code generated by `tend` is returned as integer value.
- void __builtin_tabort (int)** [Built-in Function]
 Generates the `tabort` machine instruction with the specified abort code. Abort codes from 0 through 255 are reserved and will result in an error message.
- void __builtin_tx_assist (int)** [Built-in Function]
 Generates the `ppa rX,rY,1` machine instruction. Where the integer parameter is loaded into `rX` and a value of zero is loaded into `rY`. The integer parameter specifies the number of times the transaction repeatedly aborted.
- int __builtin_tx_nesting_depth (void)** [Built-in Function]
 Generates the `etnd` machine instruction. The current nesting depth is returned as integer value. For a nesting depth of 0 the code is not executed as part of an transaction.
- void __builtin_non_tx_store (uint64_t *, uint64_t)** [Built-in Function]
 Generates the `ntstg` machine instruction. The second argument is written to the first arguments location. The store operation will not be rolled-back in case of an transaction abort.

7.13.34 SH Built-in Functions

The following built-in functions are supported on the SH1, SH2, SH3 and SH4 families of processors:

- void __builtin_set_thread_pointer (void *ptr)** [Built-in Function]
 Sets the ‘GBR’ register to the specified value *ptr*. This is usually used by system code that manages threads and execution contexts. The compiler normally does not generate code that modifies the contents of ‘GBR’ and thus the value is preserved across function calls. Changing the ‘GBR’ value in user code must be done with caution, since the compiler might use ‘GBR’ in order to access thread local variables.

void * __builtin_thread_pointer (void) [Built-in Function]

Returns the value that is currently set in the ‘GBR’ register. Memory loads and stores that use the thread pointer as a base address are turned into ‘GBR’ based displacement loads and stores, if possible. For example:

```
struct my_tcb
{
    int a, b, c, d, e;
};

int get_tcb_value (void)
{
    // Generate 'mov.l @(8,gbp),r0' instruction
    return ((my_tcb *)__builtin_thread_pointer ())->c;
}
```

unsigned int __builtin_sh_get_fpscr (void) [Built-in Function]

Returns the value that is currently set in the ‘FPSCR’ register.

void __builtin_sh_set_fpscr (unsigned int val) [Built-in Function]

Sets the ‘FPSCR’ register to the specified value *val*, while preserving the current values of the FR, SZ and PR bits.

7.13.35 SPARC VIS Built-in Functions

GCC supports SIMD operations on the SPARC using both the generic vector extensions (see Section 7.8 [Vector Extensions], page 816) as well as built-in functions for the SPARC Visual Instruction Set (VIS). When you use the `-mvis` switch, the VIS extension is exposed as the following built-in functions:

```
typedef int v1si __attribute__((vector_size(4)));
typedef int v2si __attribute__((vector_size(8)));
typedef short v4hi __attribute__((vector_size(8)));
typedef short v2hi __attribute__((vector_size(4)));
typedef unsigned char v8qi __attribute__((vector_size(8)));
typedef unsigned char v4qi __attribute__((vector_size(4)));

void __builtin_vis_write_gsr (int64_t);
int64_t __builtin_vis_read_gsr (void);

void * __builtin_vis_alignaddr (void *, long);
void * __builtin_vis_alignaddr1 (void *, long);
int64_t __builtin_vis_faligndatadi (int64_t, int64_t);
v2si __builtin_vis_faligndatav2si (v2si, v2si);
v4hi __builtin_vis_faligndatav4hi (v4si, v4si);
v8qi __builtin_vis_faligndatav8qi (v8qi, v8qi);

v4hi __builtin_vis_fexpand (v4qi);

v4hi __builtin_vis_fmulsx16 (v4qi, v4hi);
v4hi __builtin_vis_fmulsx16au (v4qi, v2hi);
v4hi __builtin_vis_fmulsx16al (v4qi, v2hi);
v4hi __builtin_vis_fmulsx16 (v8qi, v4hi);
v4hi __builtin_vis_fmulsx16 (v8qi, v4hi);
v2si __builtin_vis_fmulsx16 (v4qi, v2hi);
v2si __builtin_vis_fmulsx16 (v4qi, v2hi);
```

```

v4qi __builtin_vis_fpack16 (v4hi);
v8qi __builtin_vis_fpack32 (v2si, v8qi);
v2hi __builtin_vis_fpackfix (v2si);
v8qi __builtin_vis_fmmerge (v4qi, v4qi);

int64_t __builtin_vis_pdist (v8qi, v8qi, int64_t);

long __builtin_vis_edge8 (void *, void *);
long __builtin_vis_edge8l (void *, void *);
long __builtin_vis_edge16 (void *, void *);
long __builtin_vis_edge16l (void *, void *);
long __builtin_vis_edge32 (void *, void *);
long __builtin_vis_edge32l (void *, void *);

long __builtin_vis_fcmlpe16 (v4hi, v4hi);
long __builtin_vis_fcmlpe32 (v2si, v2si);
long __builtin_vis_fcmlpne16 (v4hi, v4hi);
long __builtin_vis_fcmlpne32 (v2si, v2si);
long __builtin_vis_fcmlpgt16 (v4hi, v4hi);
long __builtin_vis_fcmlpgt32 (v2si, v2si);
long __builtin_vis_fcmlpeq16 (v4hi, v4hi);
long __builtin_vis_fcmlpeq32 (v2si, v2si);

v4hi __builtin_vis_fpadd16 (v4hi, v4hi);
v2hi __builtin_vis_fpadd16s (v2hi, v2hi);
v2si __builtin_vis_fpadd32 (v2si, v2si);
v1si __builtin_vis_fpadd32s (v1si, v1si);
v4hi __builtin_vis_fpsub16 (v4hi, v4hi);
v2hi __builtin_vis_fpsub16s (v2hi, v2hi);
v2si __builtin_vis_fpsub32 (v2si, v2si);
v1si __builtin_vis_fpsub32s (v1si, v1si);

long __builtin_vis_array8 (long, long);
long __builtin_vis_array16 (long, long);
long __builtin_vis_array32 (long, long);

```

When you use the `-mvis2` switch, the VIS version 2.0 built-in functions also become available:

```

long __builtin_vis_bmask (long, long);
int64_t __builtin_vis_bshuffledi (int64_t, int64_t);
v2si __builtin_vis_bshufflev2si (v2si, v2si);
v4hi __builtin_vis_bshufflev2si (v4hi, v4hi);
v8qi __builtin_vis_bshufflev2si (v8qi, v8qi);

long __builtin_vis_edge8n (void *, void *);
long __builtin_vis_edge8ln (void *, void *);
long __builtin_vis_edge16n (void *, void *);
long __builtin_vis_edge16ln (void *, void *);
long __builtin_vis_edge32n (void *, void *);
long __builtin_vis_edge32ln (void *, void *);

```

When you use the `-mvis3` switch, the VIS version 3.0 built-in functions also become available:

```

void __builtin_vis_cmask8 (long);
void __builtin_vis_cmask16 (long);
void __builtin_vis_cmask32 (long);

v4hi __builtin_vis_fchksm16 (v4hi, v4hi);

```

```

v4hi __builtin_vis_fsll16 (v4hi, v4hi);
v4hi __builtin_vis_fslas16 (v4hi, v4hi);
v4hi __builtin_vis_fsrl16 (v4hi, v4hi);
v4hi __builtin_vis_fsra16 (v4hi, v4hi);
v2si __builtin_vis_fsll16 (v2si, v2si);
v2si __builtin_vis_fslas16 (v2si, v2si);
v2si __builtin_vis_fsrl16 (v2si, v2si);
v2si __builtin_vis_fsra16 (v2si, v2si);

long __builtin_vis_pdistn (v8qi, v8qi);

v4hi __builtin_vis_fmean16 (v4hi, v4hi);

int64_t __builtin_vis_fpadd64 (int64_t, int64_t);
int64_t __builtin_vis_fpsub64 (int64_t, int64_t);

v4hi __builtin_vis_fpadds16 (v4hi, v4hi);
v2hi __builtin_vis_fpadds16s (v2hi, v2hi);
v4hi __builtin_vis_fpsubs16 (v4hi, v4hi);
v2hi __builtin_vis_fpsubs16s (v2hi, v2hi);
v2si __builtin_vis_fpadds32 (v2si, v2si);
v1si __builtin_vis_fpadds32s (v1si, v1si);
v2si __builtin_vis_fpsubs32 (v2si, v2si);
v1si __builtin_vis_fpsubs32s (v1si, v1si);

long __builtin_vis_fucmple8 (v8qi, v8qi);
long __builtin_vis_fucmpne8 (v8qi, v8qi);
long __builtin_vis_fucmpgt8 (v8qi, v8qi);
long __builtin_vis_fucmpeq8 (v8qi, v8qi);

float __builtin_vis_fhadds (float, float);
double __builtin_vis_fhadd (double, double);
float __builtin_vis_fhsubs (float, float);
double __builtin_vis_fhsubd (double, double);
float __builtin_vis_fnhadds (float, float);
double __builtin_vis_fnhadd (double, double);

int64_t __builtin_vis_umulxhi (int64_t, int64_t);
int64_t __builtin_vis_xmulx (int64_t, int64_t);
int64_t __builtin_vis_xmulxhi (int64_t, int64_t);

```

When you use the `-mvis4` switch, the VIS version 4.0 built-in functions also become available:

```

v8qi __builtin_vis_fpadd8 (v8qi, v8qi);
v8qi __builtin_vis_fpadds8 (v8qi, v8qi);
v8qi __builtin_vis_fpaddus8 (v8qi, v8qi);
v4hi __builtin_vis_fpaddus16 (v4hi, v4hi);

v8qi __builtin_vis_fpsub8 (v8qi, v8qi);
v8qi __builtin_vis_fpsubs8 (v8qi, v8qi);
v8qi __builtin_vis_fpsubus8 (v8qi, v8qi);
v4hi __builtin_vis_fpsubus16 (v4hi, v4hi);

long __builtin_vis_fpcmple8 (v8qi, v8qi);
long __builtin_vis_fpcmpgt8 (v8qi, v8qi);
long __builtin_vis_fpcmpule16 (v4hi, v4hi);
long __builtin_vis_fpcmpugt16 (v4hi, v4hi);

```

```

long __builtin_vis_fpcmpule32 (v2si, v2si);
long __builtin_vis_fpcmpugt32 (v2si, v2si);

v8qi __builtin_vis_fpmax8 (v8qi, v8qi);
v4hi __builtin_vis_fpmax16 (v4hi, v4hi);
v2si __builtin_vis_fpmax32 (v2si, v2si);

v8qi __builtin_vis_fpmaxu8 (v8qi, v8qi);
v4hi __builtin_vis_fpmaxu16 (v4hi, v4hi);
v2si __builtin_vis_fpmaxu32 (v2si, v2si);

v8qi __builtin_vis_fpmin8 (v8qi, v8qi);
v4hi __builtin_vis_fpmin16 (v4hi, v4hi);
v2si __builtin_vis_fpmin32 (v2si, v2si);

v8qi __builtin_vis_fpminu8 (v8qi, v8qi);
v4hi __builtin_vis_fpminu16 (v4hi, v4hi);
v2si __builtin_vis_fpminu32 (v2si, v2si);

```

When you use the `-mvis4b` switch, the VIS version 4.0B built-in functions also become available:

```

v8qi __builtin_vis_dictunpack8 (double, int);
v4hi __builtin_vis_dictunpack16 (double, int);
v2si __builtin_vis_dictunpack32 (double, int);

long __builtin_vis_fpcmp8shl (v8qi, v8qi, int);
long __builtin_vis_fpcmpgt8shl (v8qi, v8qi, int);
long __builtin_vis_fpcmp8shl (v8qi, v8qi, int);
long __builtin_vis_fpcmpne8shl (v8qi, v8qi, int);

long __builtin_vis_fpcmp16shl (v4hi, v4hi, int);
long __builtin_vis_fpcmpgt16shl (v4hi, v4hi, int);
long __builtin_vis_fpcmp16shl (v4hi, v4hi, int);
long __builtin_vis_fpcmpne16shl (v4hi, v4hi, int);

long __builtin_vis_fpcmp32shl (v2si, v2si, int);
long __builtin_vis_fpcmpgt32shl (v2si, v2si, int);
long __builtin_vis_fpcmp32shl (v2si, v2si, int);
long __builtin_vis_fpcmpne32shl (v2si, v2si, int);

long __builtin_vis_fpcmp8shl (v8qi, v8qi, int);
long __builtin_vis_fpcmpgt8shl (v8qi, v8qi, int);
long __builtin_vis_fpcmp16shl (v4hi, v4hi, int);
long __builtin_vis_fpcmpgt16shl (v4hi, v4hi, int);
long __builtin_vis_fpcmp32shl (v2si, v2si, int);
long __builtin_vis_fpcmpgt32shl (v2si, v2si, int);

long __builtin_vis_fpcmpde8shl (v8qi, v8qi, int);
long __builtin_vis_fpcmpde16shl (v4hi, v4hi, int);
long __builtin_vis_fpcmpde32shl (v2si, v2si, int);

long __builtin_vis_fpcmp8shl (v8qi, v8qi, int);
long __builtin_vis_fpcmp16shl (v4hi, v4hi, int);
long __builtin_vis_fpcmp32shl (v2si, v2si, int);

```

7.13.36 TI C6X Built-in Functions

GCC provides intrinsics to access certain instructions of the TI C6X processors. These intrinsics, listed below, are available after inclusion of the `c6x_intrinsics.h` header file. They map directly to C6X instructions.

```
int _sadd (int, int);
int _ssub (int, int);
int _sadd2 (int, int);
int _ssub2 (int, int);
long long _mpy2 (int, int);
long long _smpy2 (int, int);
int _add4 (int, int);
int _sub4 (int, int);
int _saddu4 (int, int);

int _smpy (int, int);
int _smpyh (int, int);
int _smpyhl (int, int);
int _smpylh (int, int);

int _sshl (int, int);
int _subc (int, int);

int _avg2 (int, int);
int _avgu4 (int, int);

int _clrr (int, int);
int _extr (int, int);
int _extru (int, int);
int _abs (int);
int _abs2 (int);
```

7.13.37 x86 Built-in Functions

These built-in functions are available for the x86-32 and x86-64 family of computers, depending on the command-line switches used.

If you specify command-line switches such as `-msse`, the compiler could use the extended instruction sets even if the built-ins are not used explicitly in the program. For this reason, applications that perform run-time CPU detection must compile separate files for each supported architecture, using the appropriate flags. In particular, the file containing the CPU detection code should be compiled without these options.

The following machine modes are available for use with MMX built-in functions (see Section 7.8 [Vector Extensions], page 816): `V2SI` for a vector of two 32-bit integers, `V4HI` for a vector of four 16-bit integers, and `V8QI` for a vector of eight 8-bit integers. Some of the built-in functions operate on MMX registers as a whole 64-bit entity, these use `V1DI` as their mode.

If 3DNow! extensions are enabled, `V2SF` is used as a mode for a vector of two 32-bit floating-point values.

If SSE extensions are enabled, `V4SF` is used for a vector of four 32-bit floating-point values. Some instructions use a vector of four 32-bit integers, these use `V4SI`. Finally, some instructions operate on an entire vector register, interpreting it as a 128-bit integer, these use mode `TI`.

The x86-32 and x86-64 family of processors use additional built-in functions for efficient use of TF (`__float128`) 128-bit floating point and TC 128-bit complex floating-point values.

The following floating-point built-in functions are always available:

`__float128 __builtin_fabsq (__float128 x)` [Built-in Function]
Computes the absolute value of `x`.

`__float128 __builtin_copysignq (__float128 x, __float128 y)` [Built-in Function]
Copies the sign of `y` into `x` and returns the new value of `x`.

`__float128 __builtin_infq (void)` [Built-in Function]
Similar to `__builtin_inf`, except the return type is `__float128`.

`__float128 __builtin_huge_valq (void)` [Built-in Function]
Similar to `__builtin_huge_val`, except the return type is `__float128`.

`__float128 __builtin_nanq (void)` [Built-in Function]
Similar to `__builtin_nan`, except the return type is `__float128`.

`__float128 __builtin_nansq (void)` [Built-in Function]
Similar to `__builtin_nans`, except the return type is `__float128`.

The following built-in function is always available.

`void __builtin_ia32_pause (void)` [Built-in Function]
Generates the `pause` machine instruction with a compiler memory barrier.

The following built-in functions are always available and can be used to check the target platform type.

`void __builtin_cpu_init (void)` [Built-in Function]
This function runs the CPU detection code to check the type of CPU and the features supported. This built-in function needs to be invoked along with the built-in functions to check CPU type and features, `__builtin_cpu_is` and `__builtin_cpu_supports`, only when used in a function that is executed before any constructors are called. The CPU detection code is automatically executed in a very high priority constructor.

For example, this function has to be used in `ifunc` resolvers that check for CPU type using the built-in functions `__builtin_cpu_is` and `__builtin_cpu_supports`, or in constructors on targets that don't support constructor priority.

```
static void (*resolve_memcpy (void)) (void)
{
    // ifunc resolvers fire before constructors, explicitly call the init
    // function.
    __builtin_cpu_init ();
    if (__builtin_cpu_supports ("ssse3"))
        return ssse3_memcpy; // super fast memcpy with ssse3 instructions.
    else
        return default_memcpy;
}

void *memcpy (void *, const void *, size_t)
    __attribute__((ifunc ("resolve_memcpy")));
```

`int __builtin_cpu_is (const char *cpuname)` [Built-in Function]

This function returns a positive integer if the run-time CPU is of type *cpuname* and returns 0 otherwise. The following CPU names can be detected:

<code>'amd'</code>	AMD CPU.
<code>'intel'</code>	Intel CPU.
<code>'atom'</code>	Intel Atom CPU.
<code>'slm'</code>	Intel Silvermont CPU.
<code>'core2'</code>	Intel Core 2 CPU.
<code>'corei7'</code>	Intel Core i7 CPU.
<code>'nehalem'</code>	Intel Core i7 Nehalem CPU.
<code>'westmere'</code>	Intel Core i7 Westmere CPU.
<code>'sandybridge'</code>	Intel Core i7 Sandy Bridge CPU.
<code>'ivybridge'</code>	Intel Core i7 Ivy Bridge CPU.
<code>'haswell'</code>	Intel Core i7 Haswell CPU.
<code>'broadwell'</code>	Intel Core i7 Broadwell CPU.
<code>'skylake'</code>	Intel Core i7 Skylake CPU.
<code>'skylake-avx512'</code>	Intel Core i7 Skylake AVX512 CPU.
<code>'cannonlake'</code>	Intel Core i7 Cannon Lake CPU.
<code>'icelake-client'</code>	Intel Core i7 Ice Lake Client CPU.
<code>'icelake-server'</code>	Intel Core i7 Ice Lake Server CPU.
<code>'cascadelake'</code>	Intel Core i7 Cascadelake CPU.
<code>'tigerlake'</code>	Intel Core i7 Tigerlake CPU.
<code>'cooperlake'</code>	Intel Core i7 Cooperlake CPU.
<code>'sapphirerapids'</code>	Intel Core i7 sapphirerapids CPU.
<code>'alderlake'</code>	Intel Core i7 Alderlake CPU.

`'rocketlake'`
Intel Core i7 Rocketlake CPU.

`'graniterapids'`
Intel Core i7 graniterapids CPU.

`'graniterapids-d'`
Intel Core i7 graniterapids D CPU.

`'arrowlake'`
Intel Core i7 Arrow Lake CPU.

`'arrowlake-s'`
Intel Core i7 Arrow Lake S CPU.

`'pantherlake'`
Intel Core i7 Panther Lake CPU.

`'diamondrapids'`
Intel Core i7 Diamond Rapids CPU.

`'novalake'`
Intel Core i7 Nova Lake CPU.

`'bonnell'` Intel Atom Bonnell CPU.

`'silvermont'`
Intel Atom Silvermont CPU.

`'goldmont'`
Intel Atom Goldmont CPU.

`'goldmont-plus'`
Intel Atom Goldmont Plus CPU.

`'tremont'` Intel Atom Tremont CPU.

`'sierraforest'`
Intel Atom Sierra Forest CPU.

`'grandridge'`
Intel Atom Grand Ridge CPU.

`'clearwaterforest'`
Intel Atom Clearwater Forest CPU.

`'lujiazui'`
ZHAOXIN lujiazui CPU.

`'yongfeng'`
ZHAOXIN yongfeng CPU.

`'shijidadao'`
ZHAOXIN shijidadao CPU.

`'amdfam10h'`
AMD Family 10h CPU.

```

'barcelona'
    AMD Family 10h Barcelona CPU.

'shanghai'
    AMD Family 10h Shanghai CPU.

'istanbul'
    AMD Family 10h Istanbul CPU.

'btver1'
    AMD Family 14h CPU.

'amdfam15h'
    AMD Family 15h CPU.

'bdver1'
    AMD Family 15h Bulldozer version 1.

'bdver2'
    AMD Family 15h Bulldozer version 2.

'bdver3'
    AMD Family 15h Bulldozer version 3.

'bdver4'
    AMD Family 15h Bulldozer version 4.

'btver2'
    AMD Family 16h CPU.

'amdfam17h'
    AMD Family 17h CPU.

'znver1'
    AMD Family 17h Zen version 1.

'znver2'
    AMD Family 17h Zen version 2.

'amdfam19h'
    AMD Family 19h CPU.

'znver3'
    AMD Family 19h Zen version 3.

'znver4'
    AMD Family 19h Zen version 4.

'amdfam1ah'
    AMD Family 1ah CPU.

'znver5'
    AMD Family 1ah Zen version 5.

'znver6'
    AMD Family 1ah Zen version 6.

'hygonfam18h'
    HYGON Family 18h CPU.

'c86-4g-m4'
    HYGON Family 18h model 4 dharma CPU.

'c86-4g-m6'
    HYGON Family 18h model 6 shanghai CPU.

'c86-4g-m7'
    HYGON Family 18h model 7 chengdu CPU.

```

Here is an example:

```

if (__builtin_cpu_is ("corei7"))
{

```

```

        do_corei7 (); // Core i7 specific implementation.
    }
    else
    {
        do_generic (); // Generic implementation.
    }

```

`int __builtin_cpu_supports (const char *feature)` [Built-in Function]

This function returns a positive integer if the run-time CPU supports *feature* and returns 0 otherwise. The following features can be detected:

<code>'cmov'</code>	CMOV instruction.
<code>'mmx'</code>	MMX instructions.
<code>'popcnt'</code>	POPCNT instruction.
<code>'sse'</code>	SSE instructions.
<code>'sse2'</code>	SSE2 instructions.
<code>'sse3'</code>	SSE3 instructions.
<code>'ssse3'</code>	SSSE3 instructions.
<code>'sse4.1'</code>	SSE4.1 instructions.
<code>'sse4.2'</code>	SSE4.2 instructions.
<code>'avx'</code>	AVX instructions.
<code>'avx2'</code>	AVX2 instructions.
<code>'sse4a'</code>	SSE4A instructions.
<code>'fma4'</code>	FMA4 instructions.
<code>'xop'</code>	XOP instructions.
<code>'fma'</code>	FMA instructions.
<code>'avx512f'</code>	AVX512F instructions.
<code>'bmi'</code>	BMI instructions.
<code>'bmi2'</code>	BMI2 instructions.
<code>'aes'</code>	AES instructions.
<code>'pclmul'</code>	PCLMUL instructions.
<code>'avx512vl'</code>	AVX512VL instructions.
<code>'avx512bw'</code>	AVX512BW instructions.
<code>'avx512dq'</code>	AVX512DQ instructions.
<code>'avx512cd'</code>	AVX512CD instructions.

`'avx512vbmi'`
 AVX512VBMI instructions.
`'avx512ifma'`
 AVX512IFMA instructions.
`'avx512vpopcntdq'`
 AVX512VPOPCNTDQ instructions.
`'avx512vbmi2'`
 AVX512VBMI2 instructions.
`'gfni'`
 GFNI instructions.
`'vpclmulqdq'`
 VPCLMULQDQ instructions.
`'avx512vnni'`
 AVX512VNNI instructions.
`'avx512bitalg'`
 AVX512BITALG instructions.
`'x86-64'`
 Baseline x86-64 microarchitecture level (as defined in x86-64 psABI).
`'x86-64-v2'`
 x86-64-v2 microarchitecture level.
`'x86-64-v3'`
 x86-64-v3 microarchitecture level.
`'x86-64-v4'`
 x86-64-v4 microarchitecture level.

Here is an example:

```

    if (__builtin_cpu_supports ("popcnt"))
    {
        asm("popcnt %1,%0" : "=r"(count) : "rm"(n) : "cc");
    }
    else
    {
        count = generic_countbits (n); //generic implementation.
    }
  
```

The following built-in functions are made available by `-mmmx`. All of them generate the machine instruction that is part of the name.

```

v8qi __builtin_ia32_paddb (v8qi, v8qi);
v4hi __builtin_ia32_paddw (v4hi, v4hi);
v2si __builtin_ia32_paddd (v2si, v2si);
v8qi __builtin_ia32_psubb (v8qi, v8qi);
v4hi __builtin_ia32_psubw (v4hi, v4hi);
v2si __builtin_ia32_psubd (v2si, v2si);
v8qi __builtin_ia32_paddsb (v8qi, v8qi);
v4hi __builtin_ia32_paddsw (v4hi, v4hi);
v8qi __builtin_ia32_psubsb (v8qi, v8qi);
v4hi __builtin_ia32_psubsw (v4hi, v4hi);
v8qi __builtin_ia32_paddusb (v8qi, v8qi);
  
```

```

v4hi __builtin_ia32_paddusw (v4hi, v4hi);
v8qi __builtin_ia32_psubusb (v8qi, v8qi);
v4hi __builtin_ia32_psubusw (v4hi, v4hi);
v4hi __builtin_ia32_pmullw (v4hi, v4hi);
v4hi __builtin_ia32_pmulhw (v4hi, v4hi);
di __builtin_ia32_pand (di, di);
di __builtin_ia32_pandn (di, di);
di __builtin_ia32_por (di, di);
di __builtin_ia32_pxor (di, di);
v8qi __builtin_ia32_pcmpeqb (v8qi, v8qi);
v4hi __builtin_ia32_pcmpeqw (v4hi, v4hi);
v2si __builtin_ia32_pcmpeqd (v2si, v2si);
v8qi __builtin_ia32_pcmpgtb (v8qi, v8qi);
v4hi __builtin_ia32_pcmpgtw (v4hi, v4hi);
v2si __builtin_ia32_pcmpgtd (v2si, v2si);
v8qi __builtin_ia32_punpckhbw (v8qi, v8qi);
v4hi __builtin_ia32_punpckhwd (v4hi, v4hi);
v2si __builtin_ia32_punpckhdq (v2si, v2si);
v8qi __builtin_ia32_punpcklbw (v8qi, v8qi);
v4hi __builtin_ia32_punpcklwd (v4hi, v4hi);
v2si __builtin_ia32_punpckldq (v2si, v2si);
v8qi __builtin_ia32_packsswb (v4hi, v4hi);
v4hi __builtin_ia32_packssdw (v2si, v2si);
v8qi __builtin_ia32_packuswb (v4hi, v4hi);

v4hi __builtin_ia32_psllw (v4hi, v4hi);
v2si __builtin_ia32_pslll (v2si, v2si);
v1di __builtin_ia32_psllq (v1di, v1di);
v4hi __builtin_ia32_psrlw (v4hi, v4hi);
v2si __builtin_ia32_psrl (v2si, v2si);
v1di __builtin_ia32_psrlq (v1di, v1di);
v4hi __builtin_ia32_psraw (v4hi, v4hi);
v2si __builtin_ia32_psr (v2si, v2si);
v4hi __builtin_ia32_psllwi (v4hi, int);
v2si __builtin_ia32_psllldi (v2si, int);
v1di __builtin_ia32_psllqi (v1di, int);
v4hi __builtin_ia32_psrlwi (v4hi, int);
v2si __builtin_ia32_psrlldi (v2si, int);
v1di __builtin_ia32_psrlqi (v1di, int);
v4hi __builtin_ia32_psrwi (v4hi, int);
v2si __builtin_ia32_psradi (v2si, int);

```

The following built-in functions are made available either with `-msse`, or with `-m3dnowa`. All of them generate the machine instruction that is part of the name.

```

v4hi __builtin_ia32_pmulhuw (v4hi, v4hi);
v8qi __builtin_ia32_pavgb (v8qi, v8qi);
v4hi __builtin_ia32_pavgw (v4hi, v4hi);
v1di __builtin_ia32_psadbw (v8qi, v8qi);
v8qi __builtin_ia32_pmaxub (v8qi, v8qi);
v4hi __builtin_ia32_pmaxsw (v4hi, v4hi);
v8qi __builtin_ia32_pminub (v8qi, v8qi);
v4hi __builtin_ia32_pminsw (v4hi, v4hi);
int __builtin_ia32_pmovmskb (v8qi);
void __builtin_ia32_maskmovq (v8qi, v8qi, char *);
void __builtin_ia32_movntq (di *, di);
void __builtin_ia32_sfence (void);

```

The following built-in functions are available when `-msse` is used. All of them generate the machine instruction that is part of the name.

```

int __builtin_ia32_comieq (v4sf, v4sf);
int __builtin_ia32_comineq (v4sf, v4sf);
int __builtin_ia32_comilt (v4sf, v4sf);
int __builtin_ia32_comile (v4sf, v4sf);
int __builtin_ia32_comigt (v4sf, v4sf);
int __builtin_ia32_comige (v4sf, v4sf);
int __builtin_ia32_ucomieq (v4sf, v4sf);
int __builtin_ia32_ucomineq (v4sf, v4sf);
int __builtin_ia32_ucomilt (v4sf, v4sf);
int __builtin_ia32_ucomile (v4sf, v4sf);
int __builtin_ia32_ucomigt (v4sf, v4sf);
int __builtin_ia32_ucomige (v4sf, v4sf);
v4sf __builtin_ia32_addps (v4sf, v4sf);
v4sf __builtin_ia32_subps (v4sf, v4sf);
v4sf __builtin_ia32_mulps (v4sf, v4sf);
v4sf __builtin_ia32_divps (v4sf, v4sf);
v4sf __builtin_ia32_addss (v4sf, v4sf);
v4sf __builtin_ia32_subss (v4sf, v4sf);
v4sf __builtin_ia32_mulss (v4sf, v4sf);
v4sf __builtin_ia32_divss (v4sf, v4sf);
v4sf __builtin_ia32_cmpeqps (v4sf, v4sf);
v4sf __builtin_ia32_cmpltps (v4sf, v4sf);
v4sf __builtin_ia32_cmpleps (v4sf, v4sf);
v4sf __builtin_ia32_cmpgtps (v4sf, v4sf);
v4sf __builtin_ia32_cmpgeps (v4sf, v4sf);
v4sf __builtin_ia32_cmpunordps (v4sf, v4sf);
v4sf __builtin_ia32_cmpneqps (v4sf, v4sf);
v4sf __builtin_ia32_cmpnltps (v4sf, v4sf);
v4sf __builtin_ia32_cmpnleps (v4sf, v4sf);
v4sf __builtin_ia32_cmpngtps (v4sf, v4sf);
v4sf __builtin_ia32_cmpngeps (v4sf, v4sf);
v4sf __builtin_ia32_cmpordps (v4sf, v4sf);
v4sf __builtin_ia32_cmpeqss (v4sf, v4sf);
v4sf __builtin_ia32_cmpltss (v4sf, v4sf);
v4sf __builtin_ia32_cmpless (v4sf, v4sf);
v4sf __builtin_ia32_cmpunordss (v4sf, v4sf);
v4sf __builtin_ia32_cmpneqss (v4sf, v4sf);
v4sf __builtin_ia32_cmpnltss (v4sf, v4sf);
v4sf __builtin_ia32_cmpnless (v4sf, v4sf);
v4sf __builtin_ia32_cmpordss (v4sf, v4sf);
v4sf __builtin_ia32_maxps (v4sf, v4sf);
v4sf __builtin_ia32_maxss (v4sf, v4sf);
v4sf __builtin_ia32_minps (v4sf, v4sf);
v4sf __builtin_ia32_minss (v4sf, v4sf);
v4sf __builtin_ia32_andps (v4sf, v4sf);
v4sf __builtin_ia32_andnps (v4sf, v4sf);
v4sf __builtin_ia32_orps (v4sf, v4sf);
v4sf __builtin_ia32_xorps (v4sf, v4sf);
v4sf __builtin_ia32_movss (v4sf, v4sf);
v4sf __builtin_ia32_movhlps (v4sf, v4sf);
v4sf __builtin_ia32_movlhps (v4sf, v4sf);
v4sf __builtin_ia32_unpckhps (v4sf, v4sf);
v4sf __builtin_ia32_unpcklps (v4sf, v4sf);
v4sf __builtin_ia32_cvtpi2ps (v4sf, v2si);
v4sf __builtin_ia32_cvtsi2ss (v4sf, int);
v2si __builtin_ia32_cvtps2pi (v4sf);
int __builtin_ia32_cvtss2si (v4sf);
v2si __builtin_ia32_cvttps2pi (v4sf);

```

```

int __builtin_ia32_cvttss2si (v4sf);
v4sf __builtin_ia32_rcpps (v4sf);
v4sf __builtin_ia32_rsqrtps (v4sf);
v4sf __builtin_ia32_sqrtps (v4sf);
v4sf __builtin_ia32_rcpss (v4sf);
v4sf __builtin_ia32_rsqrtss (v4sf);
v4sf __builtin_ia32_sqrtss (v4sf);
v4sf __builtin_ia32_shufps (v4sf, v4sf, int);
void __builtin_ia32_movntps (float *, v4sf);
int __builtin_ia32_movmskps (v4sf);

```

The following built-in functions are available when `-msse` is used.

<code>v4sf __builtin_ia32_loadups (float *)</code>	[Built-in Function]
Generates the <code>movups</code> machine instruction as a load from memory.	
<code>void __builtin_ia32_storeups (float *, v4sf)</code>	[Built-in Function]
Generates the <code>movups</code> machine instruction as a store to memory.	
<code>v4sf __builtin_ia32_loadss (float *)</code>	[Built-in Function]
Generates the <code>movss</code> machine instruction as a load from memory.	
<code>v4sf __builtin_ia32_loadhps (v4sf, const v2sf *)</code>	[Built-in Function]
Generates the <code>movhps</code> machine instruction as a load from memory.	
<code>v4sf __builtin_ia32_loadlps (v4sf, const v2sf *)</code>	[Built-in Function]
Generates the <code>movlps</code> machine instruction as a load from memory.	
<code>void __builtin_ia32_storehps (v2sf *, v4sf)</code>	[Built-in Function]
Generates the <code>movhps</code> machine instruction as a store to memory.	
<code>void __builtin_ia32_storelps (v2sf *, v4sf)</code>	[Built-in Function]
Generates the <code>movlps</code> machine instruction as a store to memory.	

The following built-in functions are available when `-msse2` is used. All of them generate the machine instruction that is part of the name.

```

int __builtin_ia32_comisdeq (v2df, v2df);
int __builtin_ia32_comisdlt (v2df, v2df);
int __builtin_ia32_comisdle (v2df, v2df);
int __builtin_ia32_comisdgt (v2df, v2df);
int __builtin_ia32_comisdge (v2df, v2df);
int __builtin_ia32_comisdneq (v2df, v2df);
int __builtin_ia32_ucomisdeq (v2df, v2df);
int __builtin_ia32_ucomisdlt (v2df, v2df);
int __builtin_ia32_ucomisdle (v2df, v2df);
int __builtin_ia32_ucomisdgt (v2df, v2df);
int __builtin_ia32_ucomisdge (v2df, v2df);
int __builtin_ia32_ucomisdneq (v2df, v2df);
v2df __builtin_ia32_cmpeqpd (v2df, v2df);
v2df __builtin_ia32_cmpltpd (v2df, v2df);
v2df __builtin_ia32_cmplepd (v2df, v2df);
v2df __builtin_ia32_cmpgtpd (v2df, v2df);
v2df __builtin_ia32_cmpgepd (v2df, v2df);
v2df __builtin_ia32_cmpunordpd (v2df, v2df);
v2df __builtin_ia32_cmpneqpd (v2df, v2df);
v2df __builtin_ia32_cmpnltpd (v2df, v2df);

```

```

v2df __builtin_ia32_cmpnlepd (v2df, v2df);
v2df __builtin_ia32_cmpngtpd (v2df, v2df);
v2df __builtin_ia32_cmpngepd (v2df, v2df);
v2df __builtin_ia32_cmpordpd (v2df, v2df);
v2df __builtin_ia32_cmpeqsd (v2df, v2df);
v2df __builtin_ia32_cmpltsd (v2df, v2df);
v2df __builtin_ia32_cmplepd (v2df, v2df);
v2df __builtin_ia32_cmpunordsd (v2df, v2df);
v2df __builtin_ia32_cmpneqsd (v2df, v2df);
v2df __builtin_ia32_cmpnltsd (v2df, v2df);
v2df __builtin_ia32_cmpnlesd (v2df, v2df);
v2df __builtin_ia32_cmpordsd (v2df, v2df);
v2di __builtin_ia32_paddq (v2di, v2di);
v2di __builtin_ia32_psubq (v2di, v2di);
v2df __builtin_ia32_addpd (v2df, v2df);
v2df __builtin_ia32_subpd (v2df, v2df);
v2df __builtin_ia32_mulpd (v2df, v2df);
v2df __builtin_ia32_divpd (v2df, v2df);
v2df __builtin_ia32_addsd (v2df, v2df);
v2df __builtin_ia32_subsd (v2df, v2df);
v2df __builtin_ia32_mulsd (v2df, v2df);
v2df __builtin_ia32_divsd (v2df, v2df);
v2df __builtin_ia32_minpd (v2df, v2df);
v2df __builtin_ia32_maxpd (v2df, v2df);
v2df __builtin_ia32_minsd (v2df, v2df);
v2df __builtin_ia32_maxsd (v2df, v2df);
v2df __builtin_ia32_andpd (v2df, v2df);
v2df __builtin_ia32_andnpd (v2df, v2df);
v2df __builtin_ia32_orpd (v2df, v2df);
v2df __builtin_ia32_xorpd (v2df, v2df);
v2df __builtin_ia32_movsd (v2df, v2df);
v2df __builtin_ia32_unpckhpd (v2df, v2df);
v2df __builtin_ia32_unpcklpd (v2df, v2df);
v16qi __builtin_ia32_paddb128 (v16qi, v16qi);
v8hi __builtin_ia32_paddw128 (v8hi, v8hi);
v4si __builtin_ia32_paddd128 (v4si, v4si);
v2di __builtin_ia32_paddq128 (v2di, v2di);
v16qi __builtin_ia32_psubb128 (v16qi, v16qi);
v8hi __builtin_ia32_psubw128 (v8hi, v8hi);
v4si __builtin_ia32_psubd128 (v4si, v4si);
v2di __builtin_ia32_psubq128 (v2di, v2di);
v8hi __builtin_ia32_pmullw128 (v8hi, v8hi);
v8hi __builtin_ia32_pmulhw128 (v8hi, v8hi);
v2di __builtin_ia32_pand128 (v2di, v2di);
v2di __builtin_ia32_pandn128 (v2di, v2di);
v2di __builtin_ia32_por128 (v2di, v2di);
v2di __builtin_ia32_pxor128 (v2di, v2di);
v16qi __builtin_ia32_pavgb128 (v16qi, v16qi);
v8hi __builtin_ia32_pavgw128 (v8hi, v8hi);
v16qi __builtin_ia32_pcmpeqb128 (v16qi, v16qi);
v8hi __builtin_ia32_pcmpeqw128 (v8hi, v8hi);
v4si __builtin_ia32_pcmpeqd128 (v4si, v4si);
v16qi __builtin_ia32_pcmpgtb128 (v16qi, v16qi);
v8hi __builtin_ia32_pcmpgtw128 (v8hi, v8hi);
v4si __builtin_ia32_pcmpgtd128 (v4si, v4si);
v16qi __builtin_ia32_pmaxub128 (v16qi, v16qi);
v8hi __builtin_ia32_pmaxsw128 (v8hi, v8hi);
v16qi __builtin_ia32_pminub128 (v16qi, v16qi);

```

```

v8hi __builtin_ia32_pminsw128 (v8hi, v8hi);
v16qi __builtin_ia32_punpckhbw128 (v16qi, v16qi);
v8hi __builtin_ia32_punpckhwd128 (v8hi, v8hi);
v4si __builtin_ia32_punpckhdq128 (v4si, v4si);
v2di __builtin_ia32_punpckhqdq128 (v2di, v2di);
v16qi __builtin_ia32_punpcklbw128 (v16qi, v16qi);
v8hi __builtin_ia32_punpcklwd128 (v8hi, v8hi);
v4si __builtin_ia32_punpckldq128 (v4si, v4si);
v2di __builtin_ia32_punpcklqdq128 (v2di, v2di);
v16qi __builtin_ia32_packsswb128 (v8hi, v8hi);
v8hi __builtin_ia32_packssdw128 (v4si, v4si);
v16qi __builtin_ia32_packuswb128 (v8hi, v8hi);
v8hi __builtin_ia32_pmulhuw128 (v8hi, v8hi);
void __builtin_ia32_maskmovdqu (v16qi, v16qi);
v2df __builtin_ia32_loadupd (double *);
void __builtin_ia32_storeupd (double *, v2df);
v2df __builtin_ia32_loadhpd (v2df, double const *);
v2df __builtin_ia32_loadlpd (v2df, double const *);
int __builtin_ia32_movmskpd (v2df);
int __builtin_ia32_pmovmskb128 (v16qi);
void __builtin_ia32_movnti (int *, int);
void __builtin_ia32_movnti64 (long long int *, long long int);
void __builtin_ia32_movntpd (double *, v2df);
void __builtin_ia32_movntdq (v2df *, v2df);
v4si __builtin_ia32_pshufd (v4si, int);
v8hi __builtin_ia32_pshufw (v8hi, int);
v8hi __builtin_ia32_pshufhw (v8hi, int);
v2di __builtin_ia32_psadbw128 (v16qi, v16qi);
v2df __builtin_ia32_sqrtpd (v2df);
v2df __builtin_ia32_sqrtsd (v2df);
v2df __builtin_ia32_shufpd (v2df, v2df, int);
v2df __builtin_ia32_cvtdq2pd (v4si);
v4sf __builtin_ia32_cvtdq2ps (v4si);
v4si __builtin_ia32_cvtpd2dq (v2df);
v2si __builtin_ia32_cvtpd2pi (v2df);
v4sf __builtin_ia32_cvtpd2ps (v2df);
v4si __builtin_ia32_cvttpd2dq (v2df);
v2si __builtin_ia32_cvttpd2pi (v2df);
v2df __builtin_ia32_cvtpi2pd (v2si);
int __builtin_ia32_cvtsd2si (v2df);
int __builtin_ia32_cvttss2si (v2df);
long long __builtin_ia32_cvtsd2si64 (v2df);
long long __builtin_ia32_cvttss2si64 (v2df);
v4si __builtin_ia32_cvtps2dq (v4sf);
v2df __builtin_ia32_cvtps2pd (v4sf);
v4si __builtin_ia32_cvtps2dq (v4sf);
v2df __builtin_ia32_cvtsi2sd (v2df, int);
v2df __builtin_ia32_cvtsi64sd (v2df, long long);
v4sf __builtin_ia32_cvtsd2ss (v4sf, v2df);
v2df __builtin_ia32_cvtss2sd (v2df, v4sf);
void __builtin_ia32_clflush (const void *);
void __builtin_ia32_lfence (void);
void __builtin_ia32_mfence (void);
v16qi __builtin_ia32_loadddqu (const char *);
void __builtin_ia32_storedqu (char *, v16qi);
v1di __builtin_ia32_pmuludq (v2si, v2si);
v2di __builtin_ia32_pmuludq128 (v4si, v4si);
v8hi __builtin_ia32_psllw128 (v8hi, v8hi);

```

```

v4si __builtin_ia32_psllld128 (v4si, v4si);
v2di __builtin_ia32_psllq128 (v2di, v2di);
v8hi __builtin_ia32_psrlw128 (v8hi, v8hi);
v4si __builtin_ia32_psrlld128 (v4si, v4si);
v2di __builtin_ia32_psrlq128 (v2di, v2di);
v8hi __builtin_ia32_psraw128 (v8hi, v8hi);
v4si __builtin_ia32_psrad128 (v4si, v4si);
v2di __builtin_ia32_pslldq128 (v2di, int);
v8hi __builtin_ia32_psllwi128 (v8hi, int);
v4si __builtin_ia32_psllldi128 (v4si, int);
v2di __builtin_ia32_psllqi128 (v2di, int);
v2di __builtin_ia32_psrlldi128 (v2di, int);
v8hi __builtin_ia32_psrlwi128 (v8hi, int);
v4si __builtin_ia32_psrlldi128 (v4si, int);
v2di __builtin_ia32_psrlqi128 (v2di, int);
v8hi __builtin_ia32_psrawi128 (v8hi, int);
v4si __builtin_ia32_psradi128 (v4si, int);
v4si __builtin_ia32_pmaddwd128 (v8hi, v8hi);
v2di __builtin_ia32_movq128 (v2di);

```

The following built-in functions are available when `-msse3` is used. All of them generate the machine instruction that is part of the name.

```

v2df __builtin_ia32_addsubpd (v2df, v2df);
v4sf __builtin_ia32_addsubps (v4sf, v4sf);
v2df __builtin_ia32_haddpd (v2df, v2df);
v4sf __builtin_ia32_haddps (v4sf, v4sf);
v2df __builtin_ia32_hsubpd (v2df, v2df);
v4sf __builtin_ia32_hsubps (v4sf, v4sf);
v16qi __builtin_ia32_lddqu (char const *);
void __builtin_ia32_monitor (void *, unsigned int, unsigned int);
v4sf __builtin_ia32_movshdup (v4sf);
v4sf __builtin_ia32_movsldup (v4sf);
void __builtin_ia32_mwait (unsigned int, unsigned int);

```

The following built-in functions are available when `-mssse3` is used. All of them generate the machine instruction that is part of the name.

```

v2si __builtin_ia32_phaddb (v2si, v2si);
v4hi __builtin_ia32_phaddw (v4hi, v4hi);
v4hi __builtin_ia32_phaddsw (v4hi, v4hi);
v2si __builtin_ia32_phsubd (v2si, v2si);
v4hi __builtin_ia32_phsubw (v4hi, v4hi);
v4hi __builtin_ia32_phsubsw (v4hi, v4hi);
v4hi __builtin_ia32_pmaddbsw (v8qi, v8qi);
v4hi __builtin_ia32_pmulhrsw (v4hi, v4hi);
v8qi __builtin_ia32_pshufb (v8qi, v8qi);
v8qi __builtin_ia32_psignb (v8qi, v8qi);
v2si __builtin_ia32_psignd (v2si, v2si);
v4hi __builtin_ia32_psignw (v4hi, v4hi);
v1di __builtin_ia32_palignr (v1di, v1di, int);
v8qi __builtin_ia32_pabsb (v8qi);
v2si __builtin_ia32_pabsd (v2si);
v4hi __builtin_ia32_pabsw (v4hi);

```

The following built-in functions are available when `-mssse3` is used. All of them generate the machine instruction that is part of the name.

```

v4si __builtin_ia32_phaddb128 (v4si, v4si);
v8hi __builtin_ia32_phaddw128 (v8hi, v8hi);
v8hi __builtin_ia32_phaddsw128 (v8hi, v8hi);

```

```

v4si __builtin_ia32_phsubd128 (v4si, v4si);
v8hi __builtin_ia32_phsubw128 (v8hi, v8hi);
v8hi __builtin_ia32_phsubsw128 (v8hi, v8hi);
v8hi __builtin_ia32_pmaddbsw128 (v16qi, v16qi);
v8hi __builtin_ia32_pmulhrsw128 (v8hi, v8hi);
v16qi __builtin_ia32_pshufb128 (v16qi, v16qi);
v16qi __builtin_ia32_psignb128 (v16qi, v16qi);
v4si __builtin_ia32_psignd128 (v4si, v4si);
v8hi __builtin_ia32_psignw128 (v8hi, v8hi);
v2di __builtin_ia32_palignr128 (v2di, v2di, int);
v16qi __builtin_ia32_pabsb128 (v16qi);
v4si __builtin_ia32_pabsd128 (v4si);
v8hi __builtin_ia32_pabsw128 (v8hi);

```

The following built-in functions are available when `-msse4.1` is used. All of them generate the machine instruction that is part of the name.

```

v2df __builtin_ia32_blendpd (v2df, v2df, const int);
v4sf __builtin_ia32_blendps (v4sf, v4sf, const int);
v2df __builtin_ia32_blendvpd (v2df, v2df, v2df);
v4sf __builtin_ia32_blendvps (v4sf, v4sf, v4sf);
v2df __builtin_ia32_dppd (v2df, v2df, const int);
v4sf __builtin_ia32_dpds (v4sf, v4sf, const int);
v4sf __builtin_ia32_insertps128 (v4sf, v4sf, const int);
v2di __builtin_ia32_movntdqa (v2di *);
v16qi __builtin_ia32_mpsadbw128 (v16qi, v16qi, const int);
v8hi __builtin_ia32_packusdw128 (v4si, v4si);
v16qi __builtin_ia32_pblendvb128 (v16qi, v16qi, v16qi);
v8hi __builtin_ia32_pblendw128 (v8hi, v8hi, const int);
v2di __builtin_ia32_pcmpeqq (v2di, v2di);
v8hi __builtin_ia32_phminposw128 (v8hi);
v16qi __builtin_ia32_pmaxsb128 (v16qi, v16qi);
v4si __builtin_ia32_pmaxsd128 (v4si, v4si);
v4si __builtin_ia32_pmaxud128 (v4si, v4si);
v8hi __builtin_ia32_pmaxuw128 (v8hi, v8hi);
v16qi __builtin_ia32_pminsb128 (v16qi, v16qi);
v4si __builtin_ia32_pminsd128 (v4si, v4si);
v4si __builtin_ia32_pminud128 (v4si, v4si);
v8hi __builtin_ia32_pminuw128 (v8hi, v8hi);
v4si __builtin_ia32_pmovsxbd128 (v16qi);
v2di __builtin_ia32_pmovsxbq128 (v16qi);
v8hi __builtin_ia32_pmovsxbw128 (v16qi);
v2di __builtin_ia32_pmovsxdq128 (v4si);
v4si __builtin_ia32_pmovsxdw128 (v8hi);
v2di __builtin_ia32_pmovsxwq128 (v8hi);
v4si __builtin_ia32_pmovzxbd128 (v16qi);
v2di __builtin_ia32_pmovzxbq128 (v16qi);
v8hi __builtin_ia32_pmovzxbw128 (v16qi);
v2di __builtin_ia32_pmovzxdq128 (v4si);
v4si __builtin_ia32_pmovzxdw128 (v8hi);
v2di __builtin_ia32_pmovzxwq128 (v8hi);
v2di __builtin_ia32_pmuldq128 (v4si, v4si);
v4si __builtin_ia32_pmulld128 (v4si, v4si);
int __builtin_ia32_ptestc128 (v2di, v2di);
int __builtin_ia32_ptestnzc128 (v2di, v2di);
int __builtin_ia32_ptestz128 (v2di, v2di);
v2df __builtin_ia32_roundpd (v2df, const int);
v4sf __builtin_ia32_roundps (v4sf, const int);
v2df __builtin_ia32_roundsd (v2df, v2df, const int);

```

```
v4sf __builtin_ia32_roundss (v4sf, v4sf, const int);
```

The following built-in functions are available when `-msse4.1` is used.

```
v4sf __builtin_ia32_vec_set_v4sf (v4sf, float, const [Built-in Function]
int)
```

Generates the `insertps` machine instruction.

```
int __builtin_ia32_vec_ext_v16qi (v16qi, const int) [Built-in Function]
```

Generates the `pextrb` machine instruction.

```
v16qi __builtin_ia32_vec_set_v16qi (v16qi, int, [Built-in Function]
const int)
```

Generates the `pinsrb` machine instruction.

```
v4si __builtin_ia32_vec_set_v4si (v4si, int, const [Built-in Function]
int)
```

Generates the `pinsrd` machine instruction.

```
v2di __builtin_ia32_vec_set_v2di (v2di, long long, [Built-in Function]
const int)
```

Generates the `pinsrq` machine instruction in 64bit mode.

The following built-in functions are changed to generate new SSE4.1 instructions when `-msse4.1` is used.

```
float __builtin_ia32_vec_ext_v4sf (v4sf, const int) [Built-in Function]
```

Generates the `extractps` machine instruction.

```
int __builtin_ia32_vec_ext_v4si (v4si, const int) [Built-in Function]
```

Generates the `pextrd` machine instruction.

```
long long __builtin_ia32_vec_ext_v2di (v2di, const [Built-in Function]
int)
```

Generates the `pextrq` machine instruction in 64bit mode.

The following built-in functions are available when `-msse4.2` is used. All of them generate the machine instruction that is part of the name.

```
v16qi __builtin_ia32_pcmpestrm128 (v16qi, int, v16qi, int, const int);
int __builtin_ia32_pcmpestri128 (v16qi, int, v16qi, int, const int);
int __builtin_ia32_pcmpestr128 (v16qi, int, v16qi, int, const int);
int __builtin_ia32_pcmpestric128 (v16qi, int, v16qi, int, const int);
int __builtin_ia32_pcmpestrio128 (v16qi, int, v16qi, int, const int);
int __builtin_ia32_pcmpestris128 (v16qi, int, v16qi, int, const int);
int __builtin_ia32_pcmpestriz128 (v16qi, int, v16qi, int, const int);
v16qi __builtin_ia32_pcmpistrm128 (v16qi, v16qi, const int);
int __builtin_ia32_pcmpistri128 (v16qi, v16qi, const int);
int __builtin_ia32_pcmpistria128 (v16qi, v16qi, const int);
int __builtin_ia32_pcmpistric128 (v16qi, v16qi, const int);
int __builtin_ia32_pcmpistrio128 (v16qi, v16qi, const int);
int __builtin_ia32_pcmpistris128 (v16qi, v16qi, const int);
int __builtin_ia32_pcmpistriz128 (v16qi, v16qi, const int);
v2di __builtin_ia32_pcmpgtq (v2di, v2di);
```

The following built-in functions are available when `-msse4.2` is used.

`unsigned int __builtin_ia32_crc32qi (unsigned int, unsigned char)` [Built-in Function]

Generates the `crc32b` machine instruction.

`unsigned int __builtin_ia32_crc32hi (unsigned int, unsigned short)` [Built-in Function]

Generates the `crc32w` machine instruction.

`unsigned int __builtin_ia32_crc32si (unsigned int, unsigned int)` [Built-in Function]

Generates the `crc32l` machine instruction.

`unsigned long long __builtin_ia32_crc32di (unsigned long long, unsigned long long)` [Built-in Function]

Generates the `crc32q` machine instruction.

The following built-in functions are changed to generate new SSE4.2 instructions when `-msse4.2` is used.

`int __builtin_popcount (unsigned int)` [Built-in Function]

Generates the `popcntl` machine instruction.

`int __builtin_popcountl (unsigned long)` [Built-in Function]

Generates the `popcntl` or `popcntq` machine instruction, depending on the size of unsigned long.

`int __builtin_popcountll (unsigned long long)` [Built-in Function]

Generates the `popcntq` machine instruction.

The following built-in functions are available when `-mavx` is used. All of them generate the machine instruction that is part of the name.

```
v4df __builtin_ia32_addpd256 (v4df,v4df);
v8sf __builtin_ia32_addps256 (v8sf,v8sf);
v4df __builtin_ia32_addsubpd256 (v4df,v4df);
v8sf __builtin_ia32_addsubps256 (v8sf,v8sf);
v4df __builtin_ia32_andnpd256 (v4df,v4df);
v8sf __builtin_ia32_andnps256 (v8sf,v8sf);
v4df __builtin_ia32_andpd256 (v4df,v4df);
v8sf __builtin_ia32_andps256 (v8sf,v8sf);
v4df __builtin_ia32_blendpd256 (v4df,v4df,int);
v8sf __builtin_ia32_blendps256 (v8sf,v8sf,int);
v4df __builtin_ia32_blendvpd256 (v4df,v4df,v4df);
v8sf __builtin_ia32_blendvps256 (v8sf,v8sf,v8sf);
v2df __builtin_ia32_cmppd (v2df,v2df,int);
v4df __builtin_ia32_cmppd256 (v4df,v4df,int);
v4sf __builtin_ia32_cmpps (v4sf,v4sf,int);
v8sf __builtin_ia32_cmpps256 (v8sf,v8sf,int);
v2df __builtin_ia32_cmpsd (v2df,v2df,int);
v4sf __builtin_ia32_cmpss (v4sf,v4sf,int);
v4df __builtin_ia32_cvtdq2pd256 (v4si);
v8sf __builtin_ia32_cvtdq2ps256 (v8si);
v4si __builtin_ia32_cvtpd2dq256 (v4df);
v4sf __builtin_ia32_cvtpd2ps256 (v4df);
v8si __builtin_ia32_cvtps2dq256 (v8sf);
```

```

v4df __builtin_ia32_cvttps2pd256 (v4sf);
v4si __builtin_ia32_cvttpd2dq256 (v4df);
v8si __builtin_ia32_cvttps2dq256 (v8sf);
v4df __builtin_ia32_divpd256 (v4df,v4df);
v8sf __builtin_ia32_divps256 (v8sf,v8sf);
v8sf __builtin_ia32_dpps256 (v8sf,v8sf,int);
v4df __builtin_ia32_haddpd256 (v4df,v4df);
v8sf __builtin_ia32_haddps256 (v8sf,v8sf);
v4df __builtin_ia32_hsubpd256 (v4df,v4df);
v8sf __builtin_ia32_hsubps256 (v8sf,v8sf);
v32qi __builtin_ia32_lddqu256 (pcchar);
v32qi __builtin_ia32_loaddqu256 (pcchar);
v4df __builtin_ia32_loadupd256 (pcdouble);
v8sf __builtin_ia32_loadups256 (pcfloat);
v2df __builtin_ia32_maskloadpd (pcv2df,v2df);
v4df __builtin_ia32_maskloadpd256 (pcv4df,v4df);
v4sf __builtin_ia32_maskloadps (pcv4sf,v4sf);
v8sf __builtin_ia32_maskloadps256 (pcv8sf,v8sf);
void __builtin_ia32_maskstorepd (pv2df,v2df,v2df);
void __builtin_ia32_maskstorepd256 (pv4df,v4df,v4df);
void __builtin_ia32_maskstoreps (pv4sf,v4sf,v4sf);
void __builtin_ia32_maskstoreps256 (pv8sf,v8sf,v8sf);
v4df __builtin_ia32_maxpd256 (v4df,v4df);
v8sf __builtin_ia32_maxps256 (v8sf,v8sf);
v4df __builtin_ia32_minpd256 (v4df,v4df);
v8sf __builtin_ia32_minps256 (v8sf,v8sf);
v4df __builtin_ia32_movddup256 (v4df);
int __builtin_ia32_movmskpd256 (v4df);
int __builtin_ia32_movmskps256 (v8sf);
v8sf __builtin_ia32_movshdup256 (v8sf);
v8sf __builtin_ia32_movsldup256 (v8sf);
v4df __builtin_ia32_mulpd256 (v4df,v4df);
v8sf __builtin_ia32_mulps256 (v8sf,v8sf);
v4df __builtin_ia32_orpd256 (v4df,v4df);
v8sf __builtin_ia32_orps256 (v8sf,v8sf);
v2df __builtin_ia32_pd_pd256 (v4df);
v4df __builtin_ia32_pd256_pd (v2df);
v4sf __builtin_ia32_ps_ps256 (v8sf);
v8sf __builtin_ia32_ps256_ps (v4sf);
int __builtin_ia32_ptestc256 (v4di,v4di,ptest);
int __builtin_ia32_ptestnzc256 (v4di,v4di,ptest);
int __builtin_ia32_ptestz256 (v4di,v4di,ptest);
v8sf __builtin_ia32_rcpps256 (v8sf);
v4df __builtin_ia32_roundpd256 (v4df,int);
v8sf __builtin_ia32_roundps256 (v8sf,int);
v8sf __builtin_ia32_rsqrtps_nr256 (v8sf);
v8sf __builtin_ia32_rsqrtps256 (v8sf);
v4df __builtin_ia32_shufpd256 (v4df,v4df,int);
v8sf __builtin_ia32_shufps256 (v8sf,v8sf,int);
v4si __builtin_ia32_si_si256 (v8si);
v8si __builtin_ia32_si256_si (v4si);
v4df __builtin_ia32_sqrtpd256 (v4df);
v8sf __builtin_ia32_sqrtps_nr256 (v8sf);
v8sf __builtin_ia32_sqrtps256 (v8sf);
void __builtin_ia32_storedqu256 (pchar,v32qi);
void __builtin_ia32_storeupd256 (pdouble,v4df);
void __builtin_ia32_storeups256 (pfloat,v8sf);
v4df __builtin_ia32_subpd256 (v4df,v4df);

```

```

v8sf __builtin_ia32_subps256 (v8sf,v8sf);
v4df __builtin_ia32_unpckhpd256 (v4df,v4df);
v8sf __builtin_ia32_unpckhps256 (v8sf,v8sf);
v4df __builtin_ia32_unpcklpd256 (v4df,v4df);
v8sf __builtin_ia32_unpcklps256 (v8sf,v8sf);
v4df __builtin_ia32_vbroadcastf128_pd256 (pcv2df);
v8sf __builtin_ia32_vbroadcastf128_ps256 (pcv4sf);
v4df __builtin_ia32_vbroadcastsd256 (pcdouble);
v4sf __builtin_ia32_vbroadcastss (pcfloat);
v8sf __builtin_ia32_vbroadcastss256 (pcfloat);
v2df __builtin_ia32_vextractf128_pd256 (v4df,int);
v4sf __builtin_ia32_vextractf128_ps256 (v8sf,int);
v4si __builtin_ia32_vextractf128_si256 (v8si,int);
v4df __builtin_ia32_vinsertf128_pd256 (v4df,v2df,int);
v8sf __builtin_ia32_vinsertf128_ps256 (v8sf,v4sf,int);
v8si __builtin_ia32_vinsertf128_si256 (v8si,v4si,int);
v4df __builtin_ia32_vperm2f128_pd256 (v4df,v4df,int);
v8sf __builtin_ia32_vperm2f128_ps256 (v8sf,v8sf,int);
v8si __builtin_ia32_vperm2f128_si256 (v8si,v8si,int);
v2df __builtin_ia32_vpermil2pd (v2df,v2df,v2di,int);
v4df __builtin_ia32_vpermil2pd256 (v4df,v4df,v4di,int);
v4sf __builtin_ia32_vpermil2ps (v4sf,v4sf,v4si,int);
v8sf __builtin_ia32_vpermil2ps256 (v8sf,v8sf,v8si,int);
v2df __builtin_ia32_vpermilpd (v2df,int);
v4df __builtin_ia32_vpermilpd256 (v4df,int);
v4sf __builtin_ia32_vpermilps (v4sf,int);
v8sf __builtin_ia32_vpermilps256 (v8sf,int);
v2df __builtin_ia32_vpermilvarpd (v2df,v2di);
v4df __builtin_ia32_vpermilvarpd256 (v4df,v4di);
v4sf __builtin_ia32_vpermilvarps (v4sf,v4si);
v8sf __builtin_ia32_vpermilvarps256 (v8sf,v8si);
int __builtin_ia32_vtestcpd (v2df,v2df,ptest);
int __builtin_ia32_vtestcpd256 (v4df,v4df,ptest);
int __builtin_ia32_vtestcps (v4sf,v4sf,ptest);
int __builtin_ia32_vtestcps256 (v8sf,v8sf,ptest);
int __builtin_ia32_vtestnzcpd (v2df,v2df,ptest);
int __builtin_ia32_vtestnzcpd256 (v4df,v4df,ptest);
int __builtin_ia32_vtestnzcps (v4sf,v4sf,ptest);
int __builtin_ia32_vtestnzcps256 (v8sf,v8sf,ptest);
int __builtin_ia32_vtestzpd (v2df,v2df,ptest);
int __builtin_ia32_vtestzpd256 (v4df,v4df,ptest);
int __builtin_ia32_vtestzps (v4sf,v4sf,ptest);
int __builtin_ia32_vtestzps256 (v8sf,v8sf,ptest);
void __builtin_ia32_vzeroall (void);
void __builtin_ia32_vzeroupper (void);
v4df __builtin_ia32_xorpd256 (v4df,v4df);
v8sf __builtin_ia32_xorps256 (v8sf,v8sf);

```

The following built-in functions are available when `-mavx2` is used. All of them generate the machine instruction that is part of the name.

```

v32qi __builtin_ia32_mpsadbw256 (v32qi,v32qi,int);
v32qi __builtin_ia32_pabsb256 (v32qi);
v16hi __builtin_ia32_pabsw256 (v16hi);
v8si __builtin_ia32_pabsd256 (v8si);
v16hi __builtin_ia32_packssdw256 (v8si,v8si);
v32qi __builtin_ia32_packsswb256 (v16hi,v16hi);
v16hi __builtin_ia32_packusdw256 (v8si,v8si);
v32qi __builtin_ia32_packuswb256 (v16hi,v16hi);

```

```

v32qi __builtin_ia32_paddb256 (v32qi,v32qi);
v16hi __builtin_ia32_paddw256 (v16hi,v16hi);
v8si __builtin_ia32_paddd256 (v8si,v8si);
v4di __builtin_ia32_paddq256 (v4di,v4di);
v32qi __builtin_ia32_paddsb256 (v32qi,v32qi);
v16hi __builtin_ia32_paddsw256 (v16hi,v16hi);
v32qi __builtin_ia32_paddusb256 (v32qi,v32qi);
v16hi __builtin_ia32_paddusw256 (v16hi,v16hi);
v4di __builtin_ia32_palignr256 (v4di,v4di,int);
v4di __builtin_ia32_andsi256 (v4di,v4di);
v4di __builtin_ia32_andnotsi256 (v4di,v4di);
v32qi __builtin_ia32_pavgb256 (v32qi,v32qi);
v16hi __builtin_ia32_pavgw256 (v16hi,v16hi);
v32qi __builtin_ia32_pblendvb256 (v32qi,v32qi,v32qi);
v16hi __builtin_ia32_pblendw256 (v16hi,v16hi,int);
v32qi __builtin_ia32_pcmpeqb256 (v32qi,v32qi);
v16hi __builtin_ia32_pcmpeqw256 (v16hi,v16hi);
v8si __builtin_ia32_pcmpeqd256 (v8si,v8si);
v4di __builtin_ia32_pcmpeqq256 (v4di,v4di);
v32qi __builtin_ia32_pcmpgtb256 (v32qi,v32qi);
v16hi __builtin_ia32_pcmpgtw256 (v16hi,v16hi);
v8si __builtin_ia32_pcmpgtd256 (v8si,v8si);
v4di __builtin_ia32_pcmpgtq256 (v4di,v4di);
v16hi __builtin_ia32_phaddw256 (v16hi,v16hi);
v8si __builtin_ia32_phaddd256 (v8si,v8si);
v16hi __builtin_ia32_phaddsw256 (v16hi,v16hi);
v16hi __builtin_ia32_phsubw256 (v16hi,v16hi);
v8si __builtin_ia32_phsubd256 (v8si,v8si);
v16hi __builtin_ia32_phsubsw256 (v16hi,v16hi);
v32qi __builtin_ia32_pmaddubsw256 (v32qi,v32qi);
v16hi __builtin_ia32_pmaddwd256 (v16hi,v16hi);
v32qi __builtin_ia32_pmaxsb256 (v32qi,v32qi);
v16hi __builtin_ia32_pmaxsw256 (v16hi,v16hi);
v8si __builtin_ia32_pmaxsd256 (v8si,v8si);
v32qi __builtin_ia32_pmaxub256 (v32qi,v32qi);
v16hi __builtin_ia32_pmaxuw256 (v16hi,v16hi);
v8si __builtin_ia32_pmaxud256 (v8si,v8si);
v32qi __builtin_ia32_pminsb256 (v32qi,v32qi);
v16hi __builtin_ia32_pminsw256 (v16hi,v16hi);
v8si __builtin_ia32_pminsd256 (v8si,v8si);
v32qi __builtin_ia32_pminub256 (v32qi,v32qi);
v16hi __builtin_ia32_pminuw256 (v16hi,v16hi);
v8si __builtin_ia32_pminud256 (v8si,v8si);
int __builtin_ia32_pmovmskb256 (v32qi);
v16hi __builtin_ia32_pmovsxbw256 (v16qi);
v8si __builtin_ia32_pmovsxbd256 (v16qi);
v4di __builtin_ia32_pmovsxbq256 (v16qi);
v8si __builtin_ia32_pmovsxdw256 (v8hi);
v4di __builtin_ia32_pmovsxwq256 (v8hi);
v4di __builtin_ia32_pmovsxdq256 (v4si);
v16hi __builtin_ia32_pmovzxbw256 (v16qi);
v8si __builtin_ia32_pmovzxbd256 (v16qi);
v4di __builtin_ia32_pmovzxbq256 (v16qi);
v8si __builtin_ia32_pmovzxdw256 (v8hi);
v4di __builtin_ia32_pmovzxwq256 (v8hi);
v4di __builtin_ia32_pmovzxdq256 (v4si);
v4di __builtin_ia32_pmuldq256 (v8si,v8si);
v16hi __builtin_ia32_pmulhrsw256 (v16hi, v16hi);

```

```

v16hi __builtin_ia32_pmulhuw256 (v16hi,v16hi);
v16hi __builtin_ia32_pmulhw256 (v16hi,v16hi);
v16hi __builtin_ia32_pmullw256 (v16hi,v16hi);
v8si __builtin_ia32_pmulld256 (v8si,v8si);
v4di __builtin_ia32_pmuludq256 (v8si,v8si);
v4di __builtin_ia32_por256 (v4di,v4di);
v16hi __builtin_ia32_psadbw256 (v32qi,v32qi);
v32qi __builtin_ia32_pshufb256 (v32qi,v32qi);
v8si __builtin_ia32_pshufd256 (v8si,int);
v16hi __builtin_ia32_pshufhw256 (v16hi,int);
v16hi __builtin_ia32_pshufw256 (v16hi,int);
v32qi __builtin_ia32_psignb256 (v32qi,v32qi);
v16hi __builtin_ia32_psignw256 (v16hi,v16hi);
v8si __builtin_ia32_psignd256 (v8si,v8si);
v4di __builtin_ia32_psllldq256 (v4di,int);
v16hi __builtin_ia32_psllwi256 (v16hi,int);
v16hi __builtin_ia32_psllw256 (v16hi,v8hi);
v8si __builtin_ia32_psllld256 (v8si,int);
v8si __builtin_ia32_psllld256 (v8si,v4si);
v4di __builtin_ia32_psllqi256 (v4di,int);
v4di __builtin_ia32_psllq256 (v4di,v2di);
v16hi __builtin_ia32_psrwi256 (v16hi,int);
v16hi __builtin_ia32_psrw256 (v16hi,v8hi);
v8si __builtin_ia32_psradi256 (v8si,int);
v8si __builtin_ia32_psrld256 (v8si,v4si);
v4di __builtin_ia32_psrldq256 (v4di,int);
v16hi __builtin_ia32_psrld256 (v16hi,int);
v16hi __builtin_ia32_psrldq256 (v16hi,v8hi);
v8si __builtin_ia32_psrld256 (v8si,int);
v8si __builtin_ia32_psrld256 (v8si,v4si);
v4di __builtin_ia32_psrldq256 (v4di,int);
v4di __builtin_ia32_psrldq256 (v4di,v2di);
v32qi __builtin_ia32_psubb256 (v32qi,v32qi);
v32hi __builtin_ia32_psubw256 (v16hi,v16hi);
v8si __builtin_ia32_psubd256 (v8si,v8si);
v4di __builtin_ia32_psubq256 (v4di,v4di);
v32qi __builtin_ia32_psubsb256 (v32qi,v32qi);
v16hi __builtin_ia32_psubsw256 (v16hi,v16hi);
v32qi __builtin_ia32_psubsb256 (v32qi,v32qi);
v16hi __builtin_ia32_psubsw256 (v16hi,v16hi);
v32qi __builtin_ia32_punpckhbw256 (v32qi,v32qi);
v16hi __builtin_ia32_punpckhwd256 (v16hi,v16hi);
v8si __builtin_ia32_punpckhdq256 (v8si,v8si);
v4di __builtin_ia32_punpckhdq256 (v4di,v4di);
v32qi __builtin_ia32_punpcklbw256 (v32qi,v32qi);
v16hi __builtin_ia32_punpcklwd256 (v16hi,v16hi);
v8si __builtin_ia32_punpckldq256 (v8si,v8si);
v4di __builtin_ia32_punpckldq256 (v4di,v4di);
v4di __builtin_ia32_pxor256 (v4di,v4di);
v4di __builtin_ia32_movntdqa256 (v4di);
v4sf __builtin_ia32_vbroadcastss_ps (v4sf);
v8sf __builtin_ia32_vbroadcastss_ps256 (v4sf);
v4df __builtin_ia32_vbroadcastsd_pd256 (v2df);
v4di __builtin_ia32_vbroadcastsi256 (v2di);
v4si __builtin_ia32_pblendd128 (v4si,v4si);
v8si __builtin_ia32_pblendd256 (v8si,v8si);
v32qi __builtin_ia32_pbroadcastb256 (v16qi);
v16hi __builtin_ia32_pbroadcastw256 (v8hi);

```

```

v8si __builtin_ia32_pbroadcastd256 (v4si);
v4di __builtin_ia32_pbroadcastq256 (v2di);
v16qi __builtin_ia32_pbroadcastb128 (v16qi);
v8hi __builtin_ia32_pbroadcastw128 (v8hi);
v4si __builtin_ia32_pbroadcastd128 (v4si);
v2di __builtin_ia32_pbroadcastq128 (v2di);
v8si __builtin_ia32_permvarsi256 (v8si,v8si);
v4df __builtin_ia32_permdf256 (v4df,int);
v8sf __builtin_ia32_permvarsf256 (v8sf,v8sf);
v4di __builtin_ia32_permdi256 (v4di,int);
v4di __builtin_ia32_permti256 (v4di,v4di,int);
v4di __builtin_ia32_extract128i256 (v4di,int);
v4di __builtin_ia32_insert128i256 (v4di,v2di,int);
v8si __builtin_ia32_maskloadadd256 (pcv8si,v8si);
v4di __builtin_ia32_maskloadq256 (pcv4di,v4di);
v4si __builtin_ia32_maskloadadd (pcv4si,v4si);
v2di __builtin_ia32_maskloadq (pcv2di,v2di);
void __builtin_ia32_maskstored256 (pv8si,v8si,v8si);
void __builtin_ia32_maskstoreq256 (pv4di,v4di,v4di);
void __builtin_ia32_maskstored (pv4si,v4si,v4si);
void __builtin_ia32_maskstoreq (pv2di,v2di,v2di);
v8si __builtin_ia32_psllv8si (v8si,v8si);
v4si __builtin_ia32_psllv4si (v4si,v4si);
v4di __builtin_ia32_psllv4di (v4di,v4di);
v2di __builtin_ia32_psllv2di (v2di,v2di);
v8si __builtin_ia32_psrav8si (v8si,v8si);
v4si __builtin_ia32_psrav4si (v4si,v4si);
v8si __builtin_ia32_psrlv8si (v8si,v8si);
v4si __builtin_ia32_psrlv4si (v4si,v4si);
v4di __builtin_ia32_psrlv4di (v4di,v4di);
v2di __builtin_ia32_psrlv2di (v2di,v2di);
v2df __builtin_ia32_gathersiv2df (v2df, pcdouble,v4si,v2df,int);
v4df __builtin_ia32_gathersiv4df (v4df, pcdouble,v4si,v4df,int);
v2df __builtin_ia32_gatherdiv2df (v2df, pcdouble,v2di,v2df,int);
v4df __builtin_ia32_gatherdiv4df (v4df, pcdouble,v4di,v4df,int);
v4sf __builtin_ia32_gathersiv4sf (v4sf, pcfloat,v4si,v4sf,int);
v8sf __builtin_ia32_gathersiv8sf (v8sf, pcfloat,v8si,v8sf,int);
v4sf __builtin_ia32_gatherdiv4sf (v4sf, pcfloat,v2di,v4sf,int);
v4sf __builtin_ia32_gatherdiv4sf256 (v4sf, pcfloat,v4di,v4sf,int);
v2di __builtin_ia32_gathersiv2di (v2di, pcint64,v4si,v2di,int);
v4di __builtin_ia32_gathersiv4di (v4di, pcint64,v4si,v4di,int);
v2di __builtin_ia32_gatherdiv2di (v2di, pcint64,v2di,v2di,int);
v4di __builtin_ia32_gatherdiv4di (v4di, pcint64,v4di,v4di,int);
v4si __builtin_ia32_gathersiv4si (v4si, pcint,v4si,v4si,int);
v8si __builtin_ia32_gathersiv8si (v8si, pcint,v8si,v8si,int);
v4si __builtin_ia32_gatherdiv4si (v4si, pcint,v2di,v4si,int);
v4si __builtin_ia32_gatherdiv4si256 (v4si, pcint,v4di,v4si,int);

```

The following built-in functions are available when `-maes` is used. All of them generate the machine instruction that is part of the name.

```

v2di __builtin_ia32_aesenc128 (v2di, v2di);
v2di __builtin_ia32_aesenc128 (v2di, v2di);
v2di __builtin_ia32_aesdec128 (v2di, v2di);
v2di __builtin_ia32_aesdec128 (v2di, v2di);
v2di __builtin_ia32_aeskeygenassist128 (v2di, const int);
v2di __builtin_ia32_aesimc128 (v2di);

```

The following built-in function is available when `-mpclmul` is used.

`v2di __builtin_ia32_pclmulqdq128 (v2di, v2di, const int)` [Built-in Function]

Generates the `pclmulqdq` machine instruction.

The following built-in function is available when `-mfsgsbase` is used. All of them generate the machine instruction that is part of the name.

```
unsigned int __builtin_ia32_rdfsbase32 (void);
unsigned long long __builtin_ia32_rdfsbase64 (void);
unsigned int __builtin_ia32_rdgsgbase32 (void);
unsigned long long __builtin_ia32_rdgsgbase64 (void);
void _writefsbase_u32 (unsigned int);
void _writefsbase_u64 (unsigned long long);
void _writegsbase_u32 (unsigned int);
void _writegsbase_u64 (unsigned long long);
```

The following built-in function is available when `-mrdrnd` is used. All of them generate the machine instruction that is part of the name.

```
unsigned int __builtin_ia32_rdrand16_step (unsigned short *);
unsigned int __builtin_ia32_rdrand32_step (unsigned int *);
unsigned int __builtin_ia32_rdrand64_step (unsigned long long *);
```

The following built-in function is available when `-mptwrite` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_ptwrite32 (unsigned);
void __builtin_ia32_ptwrite64 (unsigned long long);
```

The following built-in functions are available when `-msse4a` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_movntsd (double *, v2df);
void __builtin_ia32_movntss (float *, v4sf);
v2di __builtin_ia32_extrq (v2di, v16qi);
v2di __builtin_ia32_extrqi (v2di, const unsigned int, const unsigned int);
v2di __builtin_ia32_insertq (v2di, v2di);
v2di __builtin_ia32_insertqi (v2di, v2di, const unsigned int, const unsigned int);
```

The following built-in functions are available when `-mxop` is used.

```
v2df __builtin_ia32_vfrczpd (v2df);
v4sf __builtin_ia32_vfrczps (v4sf);
v2df __builtin_ia32_vfrczsd (v2df);
v4sf __builtin_ia32_vfrczss (v4sf);
v4df __builtin_ia32_vfrczpd256 (v4df);
v8sf __builtin_ia32_vfrczps256 (v8sf);
v2di __builtin_ia32_vpcmov (v2di, v2di, v2di);
v2di __builtin_ia32_vpcmov_v2di (v2di, v2di, v2di);
v4si __builtin_ia32_vpcmov_v4si (v4si, v4si, v4si);
v8hi __builtin_ia32_vpcmov_v8hi (v8hi, v8hi, v8hi);
v16qi __builtin_ia32_vpcmov_v16qi (v16qi, v16qi, v16qi);
v2df __builtin_ia32_vpcmov_v2df (v2df, v2df, v2df);
v4sf __builtin_ia32_vpcmov_v4sf (v4sf, v4sf, v4sf);
v4di __builtin_ia32_vpcmov_v4di256 (v4di, v4di, v4di);
v8si __builtin_ia32_vpcmov_v8si256 (v8si, v8si, v8si);
v16hi __builtin_ia32_vpcmov_v16hi256 (v16hi, v16hi, v16hi);
v32qi __builtin_ia32_vpcmov_v32qi256 (v32qi, v32qi, v32qi);
v4df __builtin_ia32_vpcmov_v4df256 (v4df, v4df, v4df);
v8sf __builtin_ia32_vpcmov_v8sf256 (v8sf, v8sf, v8sf);
v16qi __builtin_ia32_vpcomeqb (v16qi, v16qi);
v8hi __builtin_ia32_vpcomeqw (v8hi, v8hi);
```

```

v4si __builtin_ia32_vpcomeqd (v4si, v4si);
v2di __builtin_ia32_vpcomeqq (v2di, v2di);
v16qi __builtin_ia32_vpcomequb (v16qi, v16qi);
v4si __builtin_ia32_vpcomequd (v4si, v4si);
v2di __builtin_ia32_vpcomequq (v2di, v2di);
v8hi __builtin_ia32_vpcomequw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomeqw (v8hi, v8hi);
v16qi __builtin_ia32_vpcomfalseb (v16qi, v16qi);
v4si __builtin_ia32_vpcomfalsed (v4si, v4si);
v2di __builtin_ia32_vpcomfalseq (v2di, v2di);
v16qi __builtin_ia32_vpcomfalseub (v16qi, v16qi);
v4si __builtin_ia32_vpcomfalseud (v4si, v4si);
v2di __builtin_ia32_vpcomfalseuq (v2di, v2di);
v8hi __builtin_ia32_vpcomfalseuw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomfalsew (v8hi, v8hi);
v16qi __builtin_ia32_vpcomgeb (v16qi, v16qi);
v4si __builtin_ia32_vpcomged (v4si, v4si);
v2di __builtin_ia32_vpcomgeq (v2di, v2di);
v16qi __builtin_ia32_vpcomgeub (v16qi, v16qi);
v4si __builtin_ia32_vpcomgeud (v4si, v4si);
v2di __builtin_ia32_vpcomgeuq (v2di, v2di);
v8hi __builtin_ia32_vpcomgeuw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomgew (v8hi, v8hi);
v16qi __builtin_ia32_vpcomgtb (v16qi, v16qi);
v4si __builtin_ia32_vpcomgtd (v4si, v4si);
v2di __builtin_ia32_vpcomgtq (v2di, v2di);
v16qi __builtin_ia32_vpcomgtub (v16qi, v16qi);
v4si __builtin_ia32_vpcomgtud (v4si, v4si);
v2di __builtin_ia32_vpcomgtuq (v2di, v2di);
v8hi __builtin_ia32_vpcomgtuw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomgtw (v8hi, v8hi);
v16qi __builtin_ia32_vpcomleb (v16qi, v16qi);
v4si __builtin_ia32_vpcomled (v4si, v4si);
v2di __builtin_ia32_vpcomleq (v2di, v2di);
v16qi __builtin_ia32_vpcomleub (v16qi, v16qi);
v4si __builtin_ia32_vpcomleud (v4si, v4si);
v2di __builtin_ia32_vpcomleuq (v2di, v2di);
v8hi __builtin_ia32_vpcomleuw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomlew (v8hi, v8hi);
v16qi __builtin_ia32_vpcomltb (v16qi, v16qi);
v4si __builtin_ia32_vpcomltd (v4si, v4si);
v2di __builtin_ia32_vpcomltq (v2di, v2di);
v16qi __builtin_ia32_vpcomltub (v16qi, v16qi);
v4si __builtin_ia32_vpcomltud (v4si, v4si);
v2di __builtin_ia32_vpcomltuq (v2di, v2di);
v8hi __builtin_ia32_vpcomltuw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomltw (v8hi, v8hi);
v16qi __builtin_ia32_vpcomneb (v16qi, v16qi);
v4si __builtin_ia32_vpcomned (v4si, v4si);
v2di __builtin_ia32_vpcomneq (v2di, v2di);
v16qi __builtin_ia32_vpcomneub (v16qi, v16qi);
v4si __builtin_ia32_vpcomneud (v4si, v4si);
v2di __builtin_ia32_vpcomneuq (v2di, v2di);
v8hi __builtin_ia32_vpcomneuw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomnew (v8hi, v8hi);
v16qi __builtin_ia32_vpcomtrueb (v16qi, v16qi);
v4si __builtin_ia32_vpcomtrued (v4si, v4si);
v2di __builtin_ia32_vpcomtrueq (v2di, v2di);

```

```

v16qi __builtin_ia32_vpcomtrueub (v16qi, v16qi);
v4si __builtin_ia32_vpcomtrueud (v4si, v4si);
v2di __builtin_ia32_vpcomtrueuq (v2di, v2di);
v8hi __builtin_ia32_vpcomtrueuw (v8hi, v8hi);
v8hi __builtin_ia32_vpcomtruew (v8hi, v8hi);
v4si __builtin_ia32_vphaddbd (v16qi);
v2di __builtin_ia32_vphaddbq (v16qi);
v8hi __builtin_ia32_vphaddbw (v16qi);
v2di __builtin_ia32_vphadddq (v4si);
v4si __builtin_ia32_vphaddbub (v16qi);
v2di __builtin_ia32_vphaddubq (v16qi);
v8hi __builtin_ia32_vphaddubw (v16qi);
v2di __builtin_ia32_vphaddudq (v4si);
v4si __builtin_ia32_vphaddudw (v8hi);
v2di __builtin_ia32_vphadduwq (v8hi);
v4si __builtin_ia32_vphaddwd (v8hi);
v2di __builtin_ia32_vphaddwq (v8hi);
v8hi __builtin_ia32_vphsubbw (v16qi);
v2di __builtin_ia32_vphsubdq (v4si);
v4si __builtin_ia32_vphsubwd (v8hi);
v4si __builtin_ia32_vpmacsd (v4si, v4si, v4si);
v2di __builtin_ia32_vpmacsdq (v4si, v4si, v2di);
v2di __builtin_ia32_vpmacsdql (v4si, v4si, v2di);
v4si __builtin_ia32_vpmacssd (v4si, v4si, v4si);
v2di __builtin_ia32_vpmacssdq (v4si, v4si, v2di);
v2di __builtin_ia32_vpmacssdql (v4si, v4si, v2di);
v4si __builtin_ia32_vpmacssw (v8hi, v8hi, v4si);
v8hi __builtin_ia32_vpmacssw (v8hi, v8hi, v8hi);
v4si __builtin_ia32_vpmacsw (v8hi, v8hi, v4si);
v8hi __builtin_ia32_vpmacsw (v8hi, v8hi, v8hi);
v4si __builtin_ia32_vpmadcsw (v8hi, v8hi, v4si);
v4si __builtin_ia32_vpmadcsw (v8hi, v8hi, v4si);
v16qi __builtin_ia32_vpper (v16qi, v16qi, v16qi);
v16qi __builtin_ia32_vprotb (v16qi, v16qi);
v4si __builtin_ia32_vprot (v4si, v4si);
v2di __builtin_ia32_vprotq (v2di, v2di);
v8hi __builtin_ia32_vprotw (v8hi, v8hi);
v16qi __builtin_ia32_vpshab (v16qi, v16qi);
v4si __builtin_ia32_vpshad (v4si, v4si);
v2di __builtin_ia32_vpshaq (v2di, v2di);
v8hi __builtin_ia32_vps (v8hi, v8hi);
v16qi __builtin_ia32_vps (v16qi, v16qi);
v4si __builtin_ia32_vps (v4si, v4si);
v2di __builtin_ia32_vps (v2di, v2di);
v8hi __builtin_ia32_vps (v8hi, v8hi);

```

The following built-in functions are available when `-mfma4` is used. All of them generate the machine instruction that is part of the name.

```

v2df __builtin_ia32_vfmaddpd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfmaddps (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfmaddsd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfmaddss (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfmsubpd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfmsubps (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfmsubsd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfmsubss (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfnmaddpd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfnmaddps (v4sf, v4sf, v4sf);

```

```

v2df __builtin_ia32_vfnmaddsd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfnmaddss (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfnmsubpd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfnmsubps (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfnmsubsd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfnmsubss (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfmaddsubpd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfmaddsubps (v4sf, v4sf, v4sf);
v2df __builtin_ia32_vfmsubaddpd (v2df, v2df, v2df);
v4sf __builtin_ia32_vfmsubaddps (v4sf, v4sf, v4sf);
v4df __builtin_ia32_vfmaddpd256 (v4df, v4df, v4df);
v8sf __builtin_ia32_vfmaddps256 (v8sf, v8sf, v8sf);
v4df __builtin_ia32_vfmsubpd256 (v4df, v4df, v4df);
v8sf __builtin_ia32_vfmsubps256 (v8sf, v8sf, v8sf);
v4df __builtin_ia32_vfnmaddpd256 (v4df, v4df, v4df);
v8sf __builtin_ia32_vfnmaddps256 (v8sf, v8sf, v8sf);
v4df __builtin_ia32_vfnmsubpd256 (v4df, v4df, v4df);
v8sf __builtin_ia32_vfnmsubps256 (v8sf, v8sf, v8sf);
v4df __builtin_ia32_vfmaddsubpd256 (v4df, v4df, v4df);
v8sf __builtin_ia32_vfmaddsubps256 (v8sf, v8sf, v8sf);
v4df __builtin_ia32_vfmsubaddpd256 (v4df, v4df, v4df);
v8sf __builtin_ia32_vfmsubaddps256 (v8sf, v8sf, v8sf);

```

The following built-in functions are available when `-mlwp` is used.

```

void __builtin_ia32_llwpcb16 (void *);
void __builtin_ia32_llwpcb32 (void *);
void __builtin_ia32_llwpcb64 (void *);
void * __builtin_ia32_llwpcb16 (void);
void * __builtin_ia32_llwpcb32 (void);
void * __builtin_ia32_llwpcb64 (void);
void __builtin_ia32_lwpval16 (unsigned short, unsigned int, unsigned short);
void __builtin_ia32_lwpval32 (unsigned int, unsigned int, unsigned int);
void __builtin_ia32_lwpval64 (unsigned __int64, unsigned int, unsigned int);
unsigned char __builtin_ia32_lwpins16 (unsigned short, unsigned int, unsigned short);
unsigned char __builtin_ia32_lwpins32 (unsigned int, unsigned int, unsigned int);
unsigned char __builtin_ia32_lwpins64 (unsigned __int64, unsigned int, unsigned int);

```

The following built-in functions are available when `-mbmi` is used. All of them generate the machine instruction that is part of the name.

```

unsigned int __builtin_ia32_bextr_u32(unsigned int, unsigned int);
unsigned long long __builtin_ia32_bextr_u64 (unsigned long long, unsigned long long);

```

The following built-in functions are available when `-mbmi2` is used. All of them generate the machine instruction that is part of the name.

```

unsigned int _bzhi_u32 (unsigned int, unsigned int);
unsigned int _pdep_u32 (unsigned int, unsigned int);
unsigned int _pext_u32 (unsigned int, unsigned int);
unsigned long long _bzhi_u64 (unsigned long long, unsigned long long);
unsigned long long _pdep_u64 (unsigned long long, unsigned long long);
unsigned long long _pext_u64 (unsigned long long, unsigned long long);

```

The following built-in functions are available when `-mlzcnt` is used. All of them generate the machine instruction that is part of the name.

```

unsigned short __builtin_ia32_lzcnt_u16(unsigned short);
unsigned int __builtin_ia32_lzcnt_u32(unsigned int);
unsigned long long __builtin_ia32_lzcnt_u64 (unsigned long long);

```

The following built-in functions are available when `-mfxsr` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_fxsave (void *);
void __builtin_ia32_fxrstor (void *);
void __builtin_ia32_fxsave64 (void *);
void __builtin_ia32_fxrstor64 (void *);
```

The following built-in functions are available when `-mxsave` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_xsave (void *, long long);
void __builtin_ia32_xrstor (void *, long long);
void __builtin_ia32_xsave64 (void *, long long);
void __builtin_ia32_xrstor64 (void *, long long);
```

The following built-in functions are available when `-mxsaveopt` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_xsaveopt (void *, long long);
void __builtin_ia32_xsaveopt64 (void *, long long);
```

The following built-in functions are available when `-mtbm` is used. Both of them generate the immediate form of the `bextr` machine instruction.

```
unsigned int __builtin_ia32_bextri_u32 (unsigned int,
                                         const unsigned int);
unsigned long long __builtin_ia32_bextri_u64 (unsigned long long,
                                              const unsigned long long);
```

The following built-in functions are available when `-m3dnow` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_femms (void);
v8qi __builtin_ia32_pavgusb (v8qi, v8qi);
v2si __builtin_ia32_pf2id (v2sf);
v2sf __builtin_ia32_pfac (v2sf, v2sf);
v2sf __builtin_ia32_pfadd (v2sf, v2sf);
v2si __builtin_ia32_pfcmeq (v2sf, v2sf);
v2si __builtin_ia32_pfcmpge (v2sf, v2sf);
v2si __builtin_ia32_pfcmpgt (v2sf, v2sf);
v2sf __builtin_ia32_pfmmax (v2sf, v2sf);
v2sf __builtin_ia32_pfmmin (v2sf, v2sf);
v2sf __builtin_ia32_pfmul (v2sf, v2sf);
v2sf __builtin_ia32_pfrcp (v2sf);
v2sf __builtin_ia32_pfrcpit1 (v2sf, v2sf);
v2sf __builtin_ia32_pfrcpit2 (v2sf, v2sf);
v2sf __builtin_ia32_pfrsqrt (v2sf);
v2sf __builtin_ia32_pfsb (v2sf, v2sf);
v2sf __builtin_ia32_pfsbr (v2sf, v2sf);
v2sf __builtin_ia32_pi2fd (v2si);
v4hi __builtin_ia32_pmulhrw (v4hi, v4hi);
```

The following built-in functions are available when `-m3dnowa` is used. All of them generate the machine instruction that is part of the name.

```
v2si __builtin_ia32_pf2iw (v2sf);
v2sf __builtin_ia32_pfnacc (v2sf, v2sf);
v2sf __builtin_ia32_pfpnacc (v2sf, v2sf);
v2sf __builtin_ia32_pi2fw (v2si);
v2sf __builtin_ia32_pswapdsf (v2sf);
v2si __builtin_ia32_pswapdsi (v2si);
```

The following built-in functions are available when `-mrtm` is used. They are used for restricted transactional memory. These are the internal low level functions. Normally the

functions in Section 7.13.38 [x86 transactional memory intrinsics], page 1025, should be used instead.

```
int __builtin_ia32_xbegin ();
void __builtin_ia32_xend ();
void __builtin_ia32_xabort (status);
int __builtin_ia32_xtest ();
```

The following built-in functions are available when `-mmwaitx` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_monitorx (void *, unsigned int, unsigned int);
void __builtin_ia32_mwaitx (unsigned int, unsigned int, unsigned int);
```

The following built-in functions are available when `-mclzero` is used. All of them generate the machine instruction that is part of the name.

```
void __builtin_ia32_clzero (void *);
```

The following built-in functions are available when `-mpku` is used. They generate reads and writes to PKRU.

```
void __builtin_ia32_wrpkr (unsigned int);
unsigned int __builtin_ia32_rdpkr ();
```

The following built-in functions are available when `-mshstk` option is used. They support shadow stack machine instructions from Intel Control-flow Enforcement Technology (CET). Each built-in function generates the machine instruction that is part of the function's name. These are the internal low-level functions. Normally the functions in Section 7.13.39 [x86 control-flow protection intrinsics], page 1027, should be used instead.

```
unsigned int __builtin_ia32_rdsspd (void);
unsigned long long __builtin_ia32_rdsppq (void);
void __builtin_ia32_incspdp (unsigned int);
void __builtin_ia32_incspdpq (unsigned long long);
void __builtin_ia32_saveprevssp (void);
void __builtin_ia32_rstorssp (void *);
void __builtin_ia32_wrssdp (unsigned int, void *);
void __builtin_ia32_wrssdpq (unsigned long long, void *);
void __builtin_ia32_wrssdp (unsigned int, void *);
void __builtin_ia32_wrssdpq (unsigned long long, void *);
void __builtin_ia32_setssbsy (void);
void __builtin_ia32_clrssbsy (void *);
```

7.13.38 x86 Transactional Memory Intrinsics

These hardware transactional memory intrinsics for x86 allow you to use memory transactions with RTM (Restricted Transactional Memory). This support is enabled with the `-mrtm` option. For using HLE (Hardware Lock Elision) see [x86 specific memory model extensions for transactional memory], page 822, instead.

A memory transaction commits all changes to memory in an atomic way, as visible to other threads. If the transaction fails it is rolled back and all side effects discarded.

Generally there is no guarantee that a memory transaction ever succeeds and suitable fallback code always needs to be supplied.

unsigned _xbegin () [RTM Function]
 Start a RTM (Restricted Transactional Memory) transaction. Returns `_XBEGIN_STARTED` when the transaction started successfully (note this is not 0, so the constant has to be explicitly tested).

If the transaction aborts, all side effects are undone and an abort code encoded as a bit mask is returned. The following macros are defined:

<code>_XABORT_EXPLICIT</code>	[Macro]
Transaction was explicitly aborted with <code>_xabort</code> . The parameter passed to <code>_xabort</code> is available with <code>_XABORT_CODE(status)</code> .	
<code>_XABORT_RETRY</code>	[Macro]
Transaction retry is possible.	
<code>_XABORT_CONFLICT</code>	[Macro]
Transaction abort due to a memory conflict with another thread.	
<code>_XABORT_CAPACITY</code>	[Macro]
Transaction abort due to the transaction using too much memory.	
<code>_XABORT_DEBUG</code>	[Macro]
Transaction abort due to a debug trap.	
<code>_XABORT_NESTED</code>	[Macro]
Transaction abort in an inner nested transaction.	

There is no guarantee any transaction ever succeeds, so there always needs to be a valid fallback path.

<code>void _xend ()</code>	[RTM Function]
Commit the current transaction. When no transaction is active this faults. All memory side effects of the transaction become visible to other threads in an atomic manner.	
<code>int _xtest ()</code>	[RTM Function]
Return a nonzero value if a transaction is currently active, otherwise 0.	
<code>void _xabort (status)</code>	[RTM Function]
Abort the current transaction. When no transaction is active this is a no-op. The <i>status</i> is an 8-bit constant; its value is encoded in the return value from <code>_xbegin</code> .	

Here is an example showing handling for `_XABORT_RETRY` and a fallback path for other failures:

```
#include <immintrin.h>

int n_tries, max_tries;
unsigned status = _XABORT_EXPLICIT;
...

for (n_tries = 0; n_tries < max_tries; n_tries++)
{
    status = _xbegin ();
    if (status == _XBEGIN_STARTED || !(status & _XABORT_RETRY))
        break;
}
if (status == _XBEGIN_STARTED)
{
```

```

    ... transaction code...
    _xend ();
}
else
{
    ... non-transactional fallback path...
}

```

Note that, in most cases, the transactional and non-transactional code must synchronize together to ensure consistency.

7.13.39 x86 Control-Flow Protection Intrinsics

ret_type _get_ssp (void) [CET Function]

Get the current value of shadow stack pointer if shadow stack support from Intel CET is enabled in the hardware or 0 otherwise. The **ret_type** is **unsigned long long** for 64-bit targets and **unsigned int** for 32-bit targets.

void _inc_ssp (unsigned int) [CET Function]

Increment the current shadow stack pointer by the size specified by the function argument. The argument is masked to a byte value for security reasons, so to increment by more than 255 bytes you must call the function multiple times.

The shadow stack unwind code looks like:

```

#include <immintrin.h>

/* Unwind the shadow stack for EH. */
#define _Unwind_Frames_Extra(x) \
do \
{ \
    _Unwind_Word ssp = _get_ssp (); \
    if (ssp != 0) \
    { \
        _Unwind_Word tmp = (x); \
        while (tmp > 255) \
        { \
            _inc_ssp (tmp); \
            tmp -= 255; \
        } \
        _inc_ssp (tmp); \
    } \
} \
while (0)

```

This code runs unconditionally on all 64-bit processors. For 32-bit processors the code runs on those that support multi-byte NOP instructions.

8 Extensions to the C++ Language

The GNU compiler provides these extensions to the C++ language (and you can also use most of the C language extensions in your C++ programs). If you want to write code that checks whether these features are available, you can test for the GNU compiler the same way as for C programs: check for a predefined macro `__GNUC__`. You can also use `__GNUG__` to test specifically for GNU C++ (see Section “Predefined Macros” in *The GNU C Preprocessor*).

8.1 When is a Volatile C++ Object Accessed?

The C++ standard differs from the C standard in its treatment of volatile objects. It fails to specify what constitutes a volatile access, except to say that C++ should behave in a similar manner to C with respect to volatiles, where possible. However, the different lvalueness of expressions between C and C++ complicate the behavior. G++ behaves the same as GCC for volatile access, See Chapter 6 [Volatiles], page 575, for a description of GCC’s behavior.

The C and C++ language specifications differ when an object is accessed in a void context:

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

The C++ standard specifies that such expressions do not undergo lvalue to rvalue conversion, and that the type of the dereferenced object may be incomplete. The C++ standard does not specify explicitly that it is lvalue to rvalue conversion that is responsible for causing an access. There is reason to believe that it is, because otherwise certain simple expressions become undefined. However, because it would surprise most programmers, G++ treats dereferencing a pointer to volatile object of complete type as GCC would do for an equivalent type in C. When the object has incomplete type, G++ issues a warning; if you wish to force an error, you must force a conversion to rvalue with, for instance, a static cast.

When using a reference to volatile, G++ does not treat equivalent expressions as accesses to volatiles, but instead issues a warning that no volatile is accessed. The rationale for this is that otherwise it becomes difficult to determine where volatile access occur, and not possible to ignore the return value from functions returning volatile references. Again, if you wish to force a read, cast the reference to an rvalue.

G++ implements the same behavior as GCC does when assigning to a volatile object—there is no reread of the assigned-to object, the assigned rvalue is reused. Note that in C++ assignment expressions are lvalues, and if used as an lvalue, the volatile object is referred to. For instance, `vref` refers to `vobj`, as expected, in the following example:

```
volatile int vobj;
volatile int &vref = vobj = something;
```

8.2 Restricting Pointer Aliasing

As with the C front end, G++ understands the C99 feature of restricted pointers, specified with the `__restrict__`, or `__restrict` type qualifier. Because you cannot compile C++ by specifying the `-std=c99` language flag, `restrict` is not a keyword in C++.

In addition to allowing restricted pointers, you can specify restricted references, which indicate that the reference is not aliased in the local context.

```
void fn (int *__restrict__ rptr, int &__restrict__ rref)
```

```

{
    /* ... */
}

```

In the body of `fn`, `rptr` points to an unaliased integer and `rref` refers to a (different) unaliased integer.

You may also specify whether a member function's *this* pointer is unaliased by using `__restrict__` as a member function qualifier.

```

void T::fn () __restrict__
{
    /* ... */
}

```

Within the body of `T::fn`, *this* has the effective definition `T *__restrict__ const this`. Notice that the interpretation of a `__restrict__` member function qualifier is different to that of `const` or `volatile` qualifier, in that it is applied to the pointer rather than the object. This is consistent with other compilers that implement restricted pointers.

As with all outermost parameter qualifiers, `__restrict__` is ignored in function definition matching. This means you only need to specify `__restrict__` in a function definition, rather than in a function prototype as well.

8.3 Vague Linkage

There are several constructs in C++ that require space in the object file but are not clearly tied to a single translation unit. We say that these constructs have “vague linkage”. Typically such constructs are emitted wherever they are needed, though sometimes we can be more clever.

Inline Functions

Inline functions are typically defined in a header file which can be included in many different compilations. Hopefully they can usually be inlined, but sometimes an out-of-line copy is necessary, if the address of the function is taken or if inlining fails. In general, we emit an out-of-line copy in all translation units where one is needed. As an exception, we only emit inline virtual functions with the vtable, since it always requires a copy.

Local static variables and string constants used in an inline function are also considered to have vague linkage, since they must be shared between all inlined and out-of-line instances of the function.

VTables

C++ virtual functions are implemented in most compilers using a lookup table, known as a vtable. The vtable contains pointers to the virtual functions provided by a class, and each object of the class contains a pointer to its vtable (or vtables, in some multiple-inheritance situations). If the class declares any non-inline, non-pure virtual functions, the first one is chosen as the “key method” for the class, and the vtable is only emitted in the translation unit where the key method is defined.

Note: If the chosen key method is later defined as inline, the vtable is still emitted in every translation unit that defines it. Make sure that any inline virtuals are declared inline in the class body, even if they are not defined there.

type_info objects

C++ requires information about types to be written out in order to implement ‘dynamic_cast’, ‘typeid’ and exception handling. For polymorphic classes (classes with virtual functions), the ‘type_info’ object is written out along with the vtable so that ‘dynamic_cast’ can determine the dynamic type of a class object at run time. For all other types, we write out the ‘type_info’ object when it is used: when applying ‘typeid’ to an expression, throwing an object, or referring to a type in a catch clause or exception specification.

Template Instantiations

Most everything in this section also applies to template instantiations, but there are other options as well. See Section 8.5 [Where’s the Template?], page 1032.

When used with GNU ld version 2.8 or later on an ELF system such as GNU/Linux or Solaris 2, or on Microsoft Windows, duplicate copies of these constructs will be discarded at link time. This is known as COMDAT support.

On targets that don’t support COMDAT, but do support weak symbols, GCC uses them. This way one copy overrides all the others, but the unused copies still take up space in the executable.

For targets that do not support either COMDAT or weak symbols, most entities with vague linkage are emitted as local symbols to avoid duplicate definition errors from the linker. This does not happen for local statics in inlines, however, as having multiple copies almost certainly breaks things.

See Section 8.4 [Declarations and Definitions in One Header], page 1031, for another way to control placement of these constructs.

8.4 C++ Interface and Implementation Pragas

#pragma interface and **#pragma implementation** provide the user with a way of explicitly directing the compiler to emit entities with vague linkage (and debugging information) in a particular translation unit.

Note: These **#pragmas** have been superseded as of GCC 2.7.2 by COMDAT support and the “key method” heuristic mentioned in Section 8.3 [Vague Linkage], page 1030. Using them can actually cause your program to grow due to unnecessary out-of-line copies of inline functions.

#pragma interface

#pragma interface "subdir/objects.h"

Use this directive in *header files* that define object classes, to save space in most of the object files that use those classes. Normally, local copies of certain information (backup copies of inline member functions, debugging information, and the internal tables that implement virtual functions) must be kept in each object file that includes class definitions. You can use this pragma to avoid such duplication. When a header file containing ‘**#pragma interface**’ is included in a compilation, this auxiliary information is not generated (unless the main input source file itself uses ‘**#pragma implementation**’). Instead, the object files contain references to be resolved at link time.

The second form of this directive is useful for the case where you have multiple headers with the same name in different directories. If you use this form, you must specify the same string to ‘`#pragma implementation`’.

```
#pragma implementation
```

```
#pragma implementation "objects.h"
```

Use this pragma in a *main input file*, when you want full output from included header files to be generated (and made globally visible). The included header file, in turn, should use ‘`#pragma interface`’. Backup copies of inline member functions, debugging information, and the internal tables used to implement virtual functions are all generated in implementation files.

If you use ‘`#pragma implementation`’ with no argument, it applies to an include file with the same basename¹ as your source file. For example, in `allclass.cc`, giving just ‘`#pragma implementation`’ by itself is equivalent to ‘`#pragma implementation "allclass.h"`’.

Use the string argument if you want a single implementation file to include code from multiple header files. (You must also use ‘`#include`’ to include the header file; ‘`#pragma implementation`’ only specifies how to use the file—it doesn’t actually include it.)

There is no way to split up the contents of a single header file into multiple implementation files.

‘`#pragma implementation`’ and ‘`#pragma interface`’ also have an effect on function inlining.

If you define a class in a header file marked with ‘`#pragma interface`’, the effect on an inline function defined in that class is similar to an explicit `extern` declaration—the compiler emits no code at all to define an independent version of the function. Its definition is used only for inlining with its callers.

Conversely, when you include the same header file in a main source file that declares it as ‘`#pragma implementation`’, the compiler emits code for the function itself; this defines a version of the function that can be found via pointers (or by callers compiled without inlining). If all calls to the function can be inlined, you can avoid emitting the function by compiling with `-fno-implement-inlines`. If any calls are not inlined, you will get linker errors.

8.5 Where’s the Template?

C++ templates were the first language feature to require more intelligence from the environment than was traditionally found on a UNIX system. Somehow the compiler and linker have to make sure that each template instance occurs exactly once in the executable if it is needed, and not at all otherwise. There are two basic approaches to this problem, which are referred to as the Borland model and the Cfront model.

Borland model

Borland C++ solved the template instantiation problem by adding the code equivalent of common blocks to their linker; the compiler emits template in-

¹ A file’s *basename* is the name stripped of all leading path information and of trailing suffixes, such as ‘.h’ or ‘.C’ or ‘.cc’.

stances in each translation unit that uses them, and the linker collapses them together. The advantage of this model is that the linker only has to consider the object files themselves; there is no external complexity to worry about. The disadvantage is that compilation time is increased because the template code is being compiled repeatedly. Code written for this model tends to include definitions of all templates in the header file, since they must be seen to be instantiated.

Cfront model

The AT&T C++ translator, Cfront, solved the template instantiation problem by creating the notion of a template repository, an automatically maintained place where template instances are stored. A more modern version of the repository works as follows: As individual object files are built, the compiler places any template definitions and instantiations encountered in the repository. At link time, the link wrapper adds in the objects in the repository and compiles any needed instances that were not previously emitted. The advantages of this model are more optimal compilation speed and the ability to use the system linker; to implement the Borland model a compiler vendor also needs to replace the linker. The disadvantages are vastly increased complexity, and thus potential for error; for some code this can be just as transparent, but in practice it can be very difficult to build multiple programs in one directory and one program in multiple directories. Code written for this model tends to separate definitions of non-inline member templates into a separate file, which should be compiled separately.

G++ implements the Borland model on targets where the linker supports it, including ELF targets (such as GNU/Linux), macOS and Microsoft Windows. Otherwise G++ implements neither automatic model.

You have the following options for dealing with template instantiations:

1. Do nothing. Code written for the Borland model works fine, but each translation unit contains instances of each of the templates it uses. The duplicate instances will be discarded by the linker, but in a large program, this can lead to an unacceptable amount of code duplication in object files or shared libraries.

Duplicate instances of a template can be avoided by defining an explicit instantiation in one object file, and preventing the compiler from doing implicit instantiations in any other object files by using an explicit instantiation declaration, using the **extern template** syntax:

```
extern template int max (int, int);
```

This syntax is defined in the C++ 2011 standard, but has been supported by G++ and other compilers since well before 2011.

Explicit instantiations can be used for the largest or most frequently duplicated instances, without having to know exactly which other instances are used in the rest of the program. You can scatter the explicit instantiations throughout your program, perhaps putting them in the translation units where the instances are used or the translation units that define the templates themselves; you can put all of the explicit instantiations you need into one big file; or you can create small files like

```
#include "Foo.h"
```

```
#include "Foo.cc"

template class Foo<int>;
template ostream& operator <<
    (ostream&, const Foo<int>&);
```

for each of the instances you need, and create a template instantiation library from those.

This is the simplest option, but also offers flexibility and fine-grained control when necessary. It is also the most portable alternative and programs using this approach will work with most modern compilers.

2. Compile your code with **-fno-implicit-templates** to disable the implicit generation of template instances, and explicitly instantiate all the ones you use. This approach requires more knowledge of exactly which instances you need than do the others, but it's less mysterious and allows greater control if you want to ensure that only the intended instances are used.

If you are using Cfront-model code, you can probably get away with not using **-fno-implicit-templates** when compiling files that don't **#include** the member template definitions.

If you use one big file to do the instantiations, you may want to compile it without **-fno-implicit-templates** so you get all of the instances required by your explicit instantiations (but not by any other files) without having to specify them as well.

In addition to forward declaration of explicit instantiations (with **extern**), G++ has extended the template instantiation syntax to support instantiation of the compiler support data for a template class (i.e. the vtable) without instantiating any of its members (with **inline**), and instantiation of only the static data members of a template class, without the support data or member functions (with **static**):

```
inline template class Foo<int>;
static template class Foo<int>;
```

8.6 Extracting the Function Pointer from a Bound Pointer to Member Function

In C++, pointer to member functions (PMFs) are implemented using a wide pointer of sorts to handle all the possible call mechanisms; the PMF needs to store information about how to adjust the **this** pointer, and if the function pointed to is virtual, where to find the vtable, and where in the vtable to look for the member function. If you are using PMFs in an inner loop, you should really reconsider that decision. If that is not an option, you can extract the pointer to the function that would be called for a given object/PMF pair and call it directly inside the inner loop, to save a bit of time.

Note that you still pay the penalty for the call through a function pointer; on most modern architectures, such a call defeats the branch prediction features of the CPU. This is also true of normal virtual function calls.

The syntax for this extension is

```
extern A a;
extern int (A::*fp)();
typedef int (*fp_ptr)(A *);
```

```
fp_ptr p = (fp_ptr)(a.*fp);
```

For PMF constants (i.e. expressions of the form ‘&Klasse::Member’), no object is needed to obtain the address of the function. They can be converted to function pointers directly:

```
fp_ptr p1 = (fp_ptr)&A::foo;
```

You must specify `-Wno-pmf-conversions` to use this extension.

8.7 C++-Specific Variable, Function, and Type Attributes

Some attributes only make sense for C++ programs.

abi_tag ("tag", ...)

The **abi_tag** attribute can be applied to a function, variable, class declaration, or inline namespace.

When applied to a function, variable, or class declaration, it modifies the mangled name of the entity to incorporate the tag name, in order to distinguish the function or class from an earlier version with a different ABI; perhaps the class has changed size, or the function has a different return type that is not encoded in the mangled name.

When applied to an inline namespace, it does not affect the mangled name of the namespace; in this case it is only used for `-Wabi-tag` warnings and automatic tagging of functions and variables. Tagging inline namespaces is generally preferable to tagging individual declarations, but the latter is sometimes necessary, such as when only certain members of a class need to be tagged.

The argument can be a list of strings of arbitrary length. The strings are sorted on output, so the order of the list is unimportant.

A redeclaration of an entity must not add new ABI tags, since doing so would change the mangled name.

The ABI tags apply to a name, so all instantiations and specializations of a template have the same tags. The attribute is ignored if applied to an explicit specialization or instantiation.

The `-Wabi-tag` flag enables a warning about a class which does not have all the ABI tags used by its subobjects and virtual functions; if your code needs to coexist with an earlier ABI, using this option can help to find all affected types that need to be tagged.

When a type involving an ABI tag is used as the type of a variable or return type of a function where that tag is not already present in the signature of the function, the tag is automatically applied to the variable or function. `-Wabi-tag` also warns about this situation; you can avoid this warning by explicitly tagging the variable or function or moving it into a tagged inline namespace.

init_priority (priority)

This attribute applies to namespace-scope variables.

In Standard C++, objects defined at namespace scope are guaranteed to be initialized in an order in strict accordance with that of their definitions *in a given translation unit*. No guarantee is made for initializations across translation units. However, GNU C++ allows you to control the order of initialization

of objects defined at namespace scope with the `init_priority` attribute by specifying a relative *priority*, with the same meaning as for the `constructor` attribute (see Section 6.4.1 [Common Attributes], page 595).

In the following example, A is normally created before B, but the `init_priority` attribute reverses that order:

```
Some_Class A __attribute__((init_priority (2000)));
Some_Class B __attribute__((init_priority (543)));
```

Note that the particular values of *priority* do not matter; only their relative ordering.

As with the *priority* argument to the `constructor` and `destructor` attributes, a few targets do not support the `init_priority` attribute. In that case the attribute is rejected with an error rather than ignored.

no_dangling

This attribute can be applied to a class type, function, or member function.

Dangling references to classes marked with this attribute have the `-Wdangling-reference` diagnostic suppressed; so do references returned from the `gnu::no_dangling`-marked functions. For example:

```
class [[gnu::no_dangling]] S { ... };
```

Or:

```
class A {
    int *p;
    [[gnu::no_dangling]] int &foo() { return *p; }
};

[[gnu::no_dangling]] const int &
foo(const int &i)
{
    ...
}
```

This attribute takes an optional argument, which must be an expression that evaluates to true or false:

```
template <typename T>
struct [[gnu::no_dangling(std::is_reference_v<T>)]] S {
    ...
};
```

Or:

```
template <typename T>
[[gnu::no_dangling(std::is_lvalue_reference_v<T>)]]
decltype(auto) foo(T&& t) {
    ...
};
```

warn_unused

This attribute applies to types.

For C++ types with non-trivial constructors and/or destructors, it is impossible for the compiler to determine whether a variable of this type is truly unused if it is not referenced. This type attribute informs the compiler that variables of this type should be warned about if they appear to be unused, just like variables of fundamental types.

This attribute is appropriate for types which just represent a value, such as `std::string`; it is not appropriate for types which control a resource, such as `std::lock_guard`.

This attribute is also accepted in C, but it is unnecessary because C does not have constructors or destructors.

cold

hot

In addition to functions and labels, GNU C++ allows the `cold` and `hot` attributes to be used on C++ classes, structs, or unions.

Applying these attributes on a type has the effect of propagating it to every member function of the type, including implicit special member functions. See Section 6.4.1 [Common Attributes], page 595.

trivial_abi

The `trivial_abi` attribute can be applied to a C++ class, struct, or union. It instructs the compiler to pass and return the type using the C ABI for the underlying type when the type would otherwise be considered non-trivial for the purposes of calls.

The attribute can be specified either as `__attribute__((trivial_abi))` or `[[clang::trivial_abi]]` for compatibility with the original implementation in Clang. To avoid portability complications, `[[gnu::trivial_abi]]` is ignored with a warning.

A class annotated with `trivial_abi` can have non-trivial destructors or copy/move constructors without automatically becoming non-trivial for the purposes of calls. For example:

```
// A is trivial for the purposes of calls despite the user-provided
// special member functions.
struct __attribute__((trivial_abi)) A {
    ~A();
    A(const A &);
    A(A &&);
    int x;
};

// B is trivial for the purposes of calls because A is.
struct B {
    A a;
};
```

If a type is trivial for the purposes of calls, has a non-trivial destructor, and is passed as an argument by value, the convention is that the callee will destroy the object before returning. The lifetime of the copy of the parameter in the caller ends without a destructor call when the call begins.

When a type marked with `trivial_abi` is used as a function argument, the compiler may omit the call to the copy constructor. Thus, side effects of the copy constructor are potentially not performed. For example, objects that contain pointers to themselves or otherwise depend on their address (or the address of their subobjects) should not be declared with `trivial_abi`.

Attribute `trivial_abi` has no effect in the following cases:

- The class has a virtual base or virtual member function.

- Copy constructors and move constructors of the class are all deleted.
- The class has a base class that is non-trivial for the purposes of calls.
- The class has a non-static data member whose type is non-trivial for the purposes of calls, which includes:
 - classes that are non-trivial for the purposes of calls
 - arrays of any of the above

8.8 Function Multiversioning

Function multiversioning is a mechanism that enables compiling multiple versions of a function, each specialized for different combinations of architecture extensions. Additionally, the compiler generates a resolver that the dynamic linker uses to detect architecture support and choose the appropriate version at runtime.

Function multiversioning relies on the indirect function extension to the ELF standard, and therefore Binutils version 2.20.1 or higher and GNU C Library version 2.11.1 are required to use this feature.

There are two versions of function multiversioning supported by GCC.

For targets supporting the `target_version` attribute (AArch64, LoongArch, and RISC-V), when compiling for C or C++, a function version set can be defined by a combination of function definitions with `target_version` and `target_clones` attributes, across translation units.

For example:

```
// fmv.h:
int foo ();
int foo [[gnu::target_clones("sve", "sve2")]] ();
int foo [[gnu::target_version("dotprod;priority=1")]] ();

// fmv1.cc
#include "fmv.h"

int foo ()
{
    // The default version of foo.
    return 0;
}

// fmv2.cc:
#include "fmv.h"

int foo [[gnu::target_clones("sve", "sve2")]] ()
{
    // foo versions for sve and sve2
    return 1;
}

int foo [[gnu::target_version("dotprod")]] ()
{
    // foo version for dotprod extension
    return 2;
}
```

```
// main.cc
#include "fmv.h"

int main ()
{
    int (*p)() = &foo;
    assert ((*p) () == foo ());
    return 0;
}
```

This example results in 4 versions of the foo function being generated, and a resolver which is used by the dynamic linker to choose the correct version.

For the AArch64 target GCC implements function multiversioning, with the semantics and version strings as specified in the Section 7.13.5 [Arm C Language Extensions (ACLE)], page 849.

For targets that support multiversioning with the `target` attribute (x86) a multiversed function can be defined with either multiple function definitions with the `target` attribute (in C++) within a translation unit, or a single definition with the `target_clones` attribute.

Here is an example.

```
__attribute__((target ("default")))
int foo ()
{
    // The default version of foo.
    return 0;
}

__attribute__((target ("sse4.2")))
int foo ()
{
    // foo version for SSE4.2
    return 1;
}

__attribute__((target ("arch=atom")))
int foo ()
{
    // foo version for the Intel ATOM processor
    return 2;
}

__attribute__((target ("arch=amdfam10")))
int foo ()
{
    // foo version for the AMD Family 0x10 processors.
    return 3;
}

int main ()
{
    int (*p)() = &foo;
    assert ((*p) () == foo ());
    return 0;
}
```

In the above example, four versions of function foo are created. The first version of foo with the target attribute "default" is the default version. This version gets executed when

no other target specific version qualifies for execution on a particular platform. A new version of `foo` is created by using the same function signature but with a different target string. Function `foo` is called or a pointer to it is taken just like a regular function. GCC takes care of doing the dispatching to call the right version at runtime. Refer to the GCC wiki on Function Multiversioning (<https://gcc.gnu.org/wiki/FunctionMultiVersioning>) for more details.

8.9 Type Traits

The C++ front end implements syntactic extensions that allow compile-time determination of various characteristics of a type (or of a pair of types).

bool `__has_nothrow_assign` (*type*) [Built-in Function]

If *type* is `const`-qualified or is a reference type then the trait is `false`. Otherwise if `__has_trivial_assign` (*type*) is `true` then the trait is `true`, else if *type* is a cv-qualified class or union type with copy assignment operators that are known not to throw an exception then the trait is `true`, else it is `false`. Requires: *type* shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

bool `__has_nothrow_copy` (*type*) [Built-in Function]

If `__has_trivial_copy` (*type*) is `true` then the trait is `true`, else if *type* is a cv-qualified class or union type with copy constructors that are known not to throw an exception then the trait is `true`, else it is `false`. Requires: *type* shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

bool `__has_nothrow_constructor` (*type*) [Built-in Function]

If `__has_trivial_constructor` (*type*) is `true` then the trait is `true`, else if *type* is a cv class or union type (or array thereof) with a default constructor that is known not to throw an exception then the trait is `true`, else it is `false`. Requires: *type* shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

bool `__has_trivial_assign` (*type*) [Built-in Function]

If *type* is `const`-qualified or is a reference type then the trait is `false`. Otherwise if `__is_trivial` (*type*) is `true` then the trait is `true`, else if *type* is a cv-qualified class or union type with a trivial copy assignment (`[class.copy]`) then the trait is `true`, else it is `false`. Requires: *type* shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

bool `__has_trivial_copy` (*type*) [Built-in Function]

If `__is_trivial` (*type*) is `true` or *type* is a reference type then the trait is `true`, else if *type* is a cv class or union type with a trivial copy constructor (`[class.copy]`) then the trait is `true`, else it is `false`. Requires: *type* shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

bool `__has_trivial_constructor` (*type*) [Built-in Function]

If `__is_trivial` (*type*) is `true` then the trait is `true`, else if *type* is a cv-qualified class or union type (or array thereof) with a trivial default constructor (`[class.ctor]`) then the trait is `true`, else it is `false`. Requires: *type* shall be a complete type, (possibly cv-qualified) `void`, or an array of unknown bound.

- bool __has_trivial_destructor (type)** [Built-in Function]
 If `__is_trivial (type)` is **true** or `type` is a reference type then the trait is **true**, else if `type` is a cv class or union type (or array thereof) with a trivial destructor (`[class.dtor]`) then the trait is **true**, else it is **false**. Requires: `type` shall be a complete type, (possibly cv-qualified) void, or an array of unknown bound.
- bool __has_virtual_destructor (type)** [Built-in Function]
 If `type` is a class type with a virtual destructor (`[class.dtor]`) then the trait is **true**, else it is **false**. Requires: If `type` is a non-union class type, it shall be a complete type.
- bool __is_abstract (type)** [Built-in Function]
 If `type` is an abstract class (`[class.abstract]`) then the trait is **true**, else it is **false**. Requires: If `type` is a non-union class type, it shall be a complete type.
- bool __is_aggregate (type)** [Built-in Function]
 If `type` is an aggregate type (`[dcl.init.aggr]`) the trait is **true**, else it is **false**. Requires: If `type` is a class type, it shall be a complete type.
- bool __is_base_of (base_type, derived_type)** [Built-in Function]
 If `base_type` is a base class of `derived_type` (`[class.derived]`) then the trait is **true**, otherwise it is **false**. Top-level cv-qualifications of `base_type` and `derived_type` are ignored. For the purposes of this trait, a class type is considered its own base. The trait is **true** even if `base_type` is an ambiguous or inaccessible base class of `derived_type`. Requires: if `__is_class (base_type)` and `__is_class (derived_type)` are **true** and `base_type` and `derived_type` are not the same type (disregarding cv-qualifiers), `derived_type` shall be a complete type. A diagnostic is produced if this requirement is not met.
- bool __builtin_is_virtual_base_of (base_type, derived_type)** [Built-in Function]
 If `base_type` is a virtual base class of `derived_type` (`[class.derived]`, `[class.mi]`) then the trait is **true**, otherwise it is **false**. Top-level cv-qualifications of `base_type` and `derived_type` are ignored. The trait is **true** even if `base_type` is an ambiguous or inaccessible virtual base class of `derived_type`. Requires: if `__is_class (base_type)` and `__is_class (derived_type)` are **true**, `derived_type` shall be a complete type. A diagnostic is produced if this requirement is not met.
- bool __is_class (type)** [Built-in Function]
 If `type` is a cv-qualified class type, and not a union type (`[basic.compound]`) the trait is **true**, else it is **false**.
- bool __is_empty (type)** [Built-in Function]
 If `__is_class (type)` is **false** then the trait is **false**. Otherwise `type` is considered empty if and only if: `type` has no non-static data members, or all non-static data members, if any, are bit-fields of length 0, and `type` has no virtual members, and `type` has no virtual base classes, and `type` has no base classes `base_type` for which `__is_empty (base_type)` is **false**. Requires: If `type` is a non-union class type, it shall be a complete type.

- bool** `__is_enum (type)` [Built-in Function]
 If *type* is a cv enumeration type ([basic.compound]) the trait is **true**, else it is **false**.
- bool** `__is_final (type)` [Built-in Function]
 If *type* is a class or union type marked **final**, then the trait is **true**, else it is **false**.
 Requires: If *type* is a class type, it shall be a complete type.
- bool** `__is_literal_type (type)` [Built-in Function]
 If *type* is a literal type ([basic.types]) the trait is **true**, else it is **false**. Requires:
type shall be a complete type, (possibly cv-qualified) **void**, or an array of unknown
 bound.
- bool** `__is_pod (type)` [Built-in Function]
 If *type* is a cv POD type ([basic.types]) then the trait is **true**, else it is **false**.
 Requires: *type* shall be a complete type, (possibly cv-qualified) **void**, or an array of
 unknown bound.
- bool** `__is_polymorphic (type)` [Built-in Function]
 If *type* is a polymorphic class ([class.virtual]) then the trait is **true**, else it is **false**.
 Requires: If *type* is a non-union class type, it shall be a complete type.
- bool** `__is_standard_layout (type)` [Built-in Function]
 If *type* is a standard-layout type ([basic.types]) the trait is **true**, else it is **false**.
 Requires: *type* shall be a complete type, an array of complete types, or (possibly
 cv-qualified) **void**.
- bool** `__is_trivial (type)` [Built-in Function]
 If *type* is a trivial type ([basic.types]) the trait is **true**, else it is **false**. Requires:
type shall be a complete type, an array of complete types, or (possibly cv-qualified)
void.
- bool** `__is_union (type)` [Built-in Function]
 If *type* is a cv union type ([basic.compound]) the trait is **true**, else it is **false**.
- bool** `__underlying_type (type)` [Built-in Function]
 The underlying type of *type*. Requires: *type* shall be an enumeration type
 ([dcl.enum]).
- bool** `__integer_pack (length)` [Built-in Function]
 When used as the pattern of a pack expansion within a template definition, expands
 to a template argument pack containing integers from 0 to *length*-1. This is provided
 for efficient implementation of `std::make_integer_sequence`.
- bool** `__is_same (type1, type2)` [Built-in Function]
 A binary type trait: **true** whenever the *type1* and *type2* refer to the same type.
- size_t** `__builtin_structured_binding_size (type)` [Built-in Function]
 This trait returns the structured binding size ([dcl.struct.bind]) of *type*. If a type does
 not have a structured binding size, an error is diagnosed unless it is used in SFINAE
 contexts.

8.10 Deprecated Features

In the past, the GNU C++ compiler was extended to experiment with new features, at a time when the C++ language was still evolving. Now that the C++ standard is complete, some of those features are superseded by superior alternatives. Using the old features might cause a warning in some cases that the feature will be dropped in the future. In other cases, the feature might be gone already.

G++ allows a virtual function returning ‘void *’ to be overridden by one returning a different pointer type. This extension to the covariant return type rules is now deprecated and will be removed from a future version.

The use of default arguments in function pointers, function typedefs and other places where they are not permitted by the standard is deprecated and will be removed from a future version of G++.

G++ allows attributes to follow a parenthesized direct initializer, e.g. ‘`int f(0) __attribute__((something));`’. This extension has been ignored since G++ 3.3 and is deprecated.

G++ allows anonymous structs and unions to have members that are not public non-static data members (i.e. fields). These extensions are deprecated.

8.11 Backwards Compatibility

Now that there is a definitive ISO standard C++, G++ has a specification to adhere to. The C++ language evolved over time, and features that used to be acceptable in previous drafts of the standard, such as the ARM [Annotated C++ Reference Manual], are no longer accepted. In order to allow compilation of C++ written to such drafts, G++ contains some backwards compatibilities. *All such backwards compatibility features are liable to disappear in future versions of G++.* They should be considered deprecated. See Section 8.10 [Deprecated Features], page 1043.

Implicit C language

Old C system header files did not contain an `extern "C" {...}` scope to set the language. On such systems, all system header files are implicitly scoped inside a C language scope. Such headers must correctly prototype function argument types, there is no leeway for `()` to indicate an unspecified set of arguments.

9 GNU Objective-C Features

This document is meant to describe some of the GNU Objective-C features. It is not intended to teach you Objective-C. There are several resources on the Internet that present the language.

9.1 GNU Objective-C Runtime API

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

The GNU Objective-C runtime provides an API that allows you to interact with the Objective-C runtime system, querying the live runtime structures and even manipulating them. This allows you for example to inspect and navigate classes, methods and protocols; to define new classes or new methods, and even to modify existing classes or protocols.

If you are using a “Foundation” library such as GNUstep-Base, this library will provide you with a rich set of functionality to do most of the inspection tasks, and you probably will only need direct access to the GNU Objective-C runtime API to define new classes or methods.

9.1.1 Modern GNU Objective-C Runtime API

The GNU Objective-C runtime provides an API which is similar to the one provided by the “Objective-C 2.0” Apple/NeXT Objective-C runtime. The API is documented in the public header files of the GNU Objective-C runtime:

- `objc/objc.h`: this is the basic Objective-C header file, defining the basic Objective-C types such as `id`, `Class` and `BOOL`. You have to include this header to do almost anything with Objective-C.
- `objc/runtime.h`: this header declares most of the public runtime API functions allowing you to inspect and manipulate the Objective-C runtime data structures. These functions are fairly standardized across Objective-C runtimes and are almost identical to the Apple/NeXT Objective-C runtime ones. It does not declare functions in some specialized areas (constructing and forwarding message invocations, threading) which are in the other headers below. You have to include `objc/objc.h` and `objc/runtime.h` to use any of the functions, such as `class_getName()`, declared in `objc/runtime.h`.
- `objc/message.h`: this header declares public functions used to construct, deconstruct and forward message invocations. Because messaging is done in quite a different way on different runtimes, functions in this header are specific to the GNU Objective-C runtime implementation.
- `objc/objc-exception.h`: this header declares some public functions related to Objective-C exceptions. For example functions in this header allow you to throw an Objective-C exception from plain C/C++ code.
- `objc/objc-sync.h`: this header declares some public functions related to the Objective-C `@synchronized()` syntax, allowing you to emulate an Objective-C `@synchronized()` block in plain C/C++ code.
- `objc/thr.h`: this header declares a public runtime API threading layer that is only provided by the GNU Objective-C runtime. It declares functions such as `objc_mutex_lock()`, which provide a platform-independent set of threading functions.

The header files contain detailed documentation for each function in the GNU Objective-C runtime API.

9.1.2 Traditional GNU Objective-C Runtime API

The GNU Objective-C runtime used to provide a different API, which we call the “traditional” GNU Objective-C runtime API. Functions belonging to this API are easy to recognize because they use a different naming convention, such as `class_get_super_class()` (traditional API) instead of `class_getSuperclass()` (modern API). Software using this API includes the file `objc/objc-api.h` where it is declared.

Starting with GCC 4.7.0, the traditional GNU runtime API is no longer available.

9.2 +load: Executing Code before main

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

The GNU Objective-C runtime provides a way that allows you to execute code before the execution of the program enters the `main` function. The code is executed on a per-class and a per-category basis, through a special class method `+load`.

This facility is very useful if you want to initialize global variables which can be accessed by the program directly, without sending a message to the class first. The usual way to initialize global variables, in the `+initialize` method, might not be useful because `+initialize` is only called when the first message is sent to a class object, which in some cases could be too late.

Suppose for example you have a `FileStream` class that declares `Stdin`, `Stdout` and `Stderr` as global variables, like below:

```
FileStream *Stdin = nil;
FileStream *Stdout = nil;
FileStream *Stderr = nil;

@implementation FileStream

+ (void)initialize
{
    Stdin = [[FileStream new] initWithFd:0];
    Stdout = [[FileStream new] initWithFd:1];
    Stderr = [[FileStream new] initWithFd:2];
}

/* Other methods here */
@end
```

In this example, the initialization of `Stdin`, `Stdout` and `Stderr` in `+initialize` occurs too late. The programmer can send a message to one of these objects before the variables are actually initialized, thus sending messages to the `nil` object. The `+initialize` method which actually initializes the global variables is not invoked until the first message is sent to the class object. The solution would require these variables to be initialized just before entering `main`.

The correct solution of the above problem is to use the `+load` method instead of `+initialize`:

```
@implementation FileStream

+ (void)load
{
    Stdin = [[FileStream new] initWithFd:0];
    Stdout = [[FileStream new] initWithFd:1];
    Stderr = [[FileStream new] initWithFd:2];
}

/* Other methods here */
@end
```

The `+load` is a method that is not overridden by categories. If a class and a category of it both implement `+load`, both methods are invoked. This allows some additional initializations to be performed in a category.

This mechanism is not intended to be a replacement for `+initialize`. You should be aware of its limitations when you decide to use it instead of `+initialize`.

9.2.1 What You Can and Cannot Do in `+load`

`+load` is to be used only as a last resort. Because it is executed very early, most of the Objective-C runtime machinery will not be ready when `+load` is executed; hence `+load` works best for executing C code that is independent on the Objective-C runtime.

The `+load` implementation in the GNU runtime guarantees you the following things:

- you can write whatever C code you like;
- you can allocate and send messages to objects whose class is implemented in the same file;
- the `+load` implementation of all super classes of a class are executed before the `+load` of that class is executed;
- the `+load` implementation of a class is executed before the `+load` implementation of any category.

In particular, the following things, even if they can work in a particular case, are not guaranteed:

- allocation of or sending messages to arbitrary objects;
- allocation of or sending messages to objects whose classes have a category implemented in the same file;
- sending messages to Objective-C constant strings (`@"this is a constant string"`);

You should make no assumptions about receiving `+load` in sibling classes when you write `+load` of a class. The order in which sibling classes receive `+load` is not guaranteed.

The order in which `+load` and `+initialize` are called could be problematic if this matters. If you don't allocate objects inside `+load`, it is guaranteed that `+load` is called before `+initialize`. If you create an object inside `+load` the `+initialize` method of object's class is invoked even if `+load` was not invoked. Note if you explicitly call `+load` on a class,

`+initialize` will be called first. To avoid possible problems try to implement only one of these methods.

The `+load` method is also invoked when a bundle is dynamically loaded into your running program. This happens automatically without any intervening operation from you. When you write bundles and you need to write `+load` you can safely create and send messages to objects whose classes already exist in the running program. The same restrictions as above apply to classes defined in bundle.

9.3 Type Encoding

This is an advanced section. Type encodings are used extensively by the compiler and by the runtime, but you generally do not need to know about them to use Objective-C.

The Objective-C compiler generates type encodings for all the types. These type encodings are used at runtime to find out information about selectors and methods and about objects and classes.

The types are encoded in the following way:

<code>_Bool</code>	B
<code>char</code>	c
<code>unsigned char</code>	C
<code>short</code>	s
<code>unsigned short</code>	S
<code>int</code>	i
<code>unsigned int</code>	I
<code>long</code>	l
<code>unsigned long</code>	L
<code>long long</code>	q
<code>unsigned long long</code>	Q
<code>float</code>	f
<code>double</code>	d
<code>long double</code>	D
<code>void</code>	v
<code>id</code>	@
<code>Class</code>	#
<code>SEL</code>	:
<code>char*</code>	*
<code>enum</code>	an <code>enum</code> is encoded exactly as the integer type that the compiler uses for it, which depends on the enumeration values. Often the compiler uses <code>unsigned int</code> , which is then encoded as I.
unknown type	?
Complex types	j followed by the inner type. For example <code>_Complex double</code> is encoded as "jd".
bit-fields	b followed by the starting position of the bit-field, the type of the bit-field and the size of the bit-field (the bit-fields encoding was changed from the NeXT's compiler encoding, see below)

The encoding of bit-fields has changed to allow bit-fields to be properly handled by the runtime functions that compute sizes and alignments of types that contain bit-fields. The

previous encoding contained only the size of the bit-field. Using only this information it is not possible to reliably compute the size occupied by the bit-field. This is very important in the presence of the Boehm's garbage collector because the objects are allocated using the typed memory facility available in this collector. The typed memory allocation requires information about where the pointers are located inside the object.

The position in the bit-field is the position, counting in bits, of the bit closest to the beginning of the structure.

The non-atomic types are encoded as follows:

pointers	'^' followed by the pointed type.
arrays	'[' followed by the number of elements in the array followed by the type of the elements followed by ']'
structures	'{' followed by the name of the structure (or '?' if the structure is unnamed), the '=' sign, the type of the members and by '}'
unions	'(' followed by the name of the structure (or '?' if the union is unnamed), the '=' sign, the type of the members followed by ')'
vectors	'![' followed by the vector_size (the number of bytes composing the vector) followed by a comma, followed by the alignment (in bytes) of the vector, followed by the type of the elements followed by ']'

Here are some types and their encodings, as they are generated by the compiler on an i386 machine:

Objective-C type	Compiler encoding
<code>int a[10];</code>	<code>[10i]</code>
<code>struct { int i; float f[3]; int a:3; int b:2; char c; }</code>	<code>{?=i[3f]b128i3b131i2c}</code>
<code>int a __attribute__((vector_size (16)));</code>	<code>![16,16i]</code> (alignment depends on the machine)

In addition to the types the compiler also encodes the type specifiers. The table below describes the encoding of the current Objective-C type specifiers:

Specifier	Encoding
<code>const</code>	<code>r</code>
<code>in</code>	<code>n</code>
<code>inout</code>	<code>N</code>
<code>out</code>	<code>o</code>
<code>bycopy</code>	<code>O</code>
<code>byref</code>	<code>R</code>
<code>oneway</code>	<code>V</code>

The type specifiers are encoded just before the type. Unlike types however, the type specifiers are only encoded when they appear in method argument types.

Note how `const` interacts with pointers:

Objective-C type	Compiler encoding
<code>const int</code>	<code>ri</code>
<code>const int*</code>	<code>^ri</code>
<code>int *const</code>	<code>r^i</code>

`const int*` is a pointer to a `const int`, and so is encoded as `^ri`. `int* const`, instead, is a `const` pointer to an `int`, and so is encoded as `r^i`.

Finally, there is a complication when encoding `const char *` versus `char * const`. Because `char *` is encoded as `*` and not as `^c`, there is no way to express the fact that `r` applies to the pointer or to the pointee.

Hence, it is assumed as a convention that `r*` means `const char *` (since it is what is most often meant), and there is no way to encode `char *const`. `char *const` would simply be encoded as `*`, and the `const` is lost.

9.3.1 Legacy Type Encoding

Unfortunately, historically GCC used to have a number of bugs in its encoding code. The NeXT runtime expects GCC to emit type encodings in this historical format (compatible with GCC-3.3), so when using the NeXT runtime, GCC will introduce on purpose a number of incorrect encodings:

- the read-only qualifier of the pointee gets emitted before the `^`. The read-only qualifier of the pointer itself gets ignored, unless it is a typedef. Also, the `r` is only emitted for the outermost type.
- 32-bit longs are encoded as `'l'` or `'L'`, but not always. For typedefs, the compiler uses `'i'` or `'I'` instead if encoding a struct field or a pointer.
- `enums` are always encoded as `'i'` (`int`) even if they are actually unsigned or long.

In addition to that, the NeXT runtime uses a different encoding for bitfields. It encodes them as `b` followed by the size, without a bit offset or the underlying field type.

9.3.2 @encode

GNU Objective-C supports the `@encode` syntax that allows you to create a type encoding from a C/Objective-C type. For example, `@encode(int)` is compiled by the compiler into `"i"`.

`@encode` does not support type qualifiers other than `const`. For example, `@encode(const char*)` is valid and is compiled into `"r*"`, while `@encode(bycopy char *)` is invalid and will cause a compilation error.

9.3.3 Method Signatures

This section documents the encoding of method types, which is rarely needed to use Objective-C. You should skip it at a first reading; the runtime provides functions that will work on methods and can walk through the list of parameters and interpret them for you. These functions are part of the public “API” and are the preferred way to interact with method signatures from user code.

But if you need to debug a problem with method signatures and need to know how they are implemented (i.e., the “ABI”), read on.

Methods have their “signature” encoded and made available to the runtime. The “signature” encodes all the information required to dynamically build invocations of the method at runtime: return type and arguments.

The “signature” is a null-terminated string, composed of the following:

- The return type, including type qualifiers. For example, a method returning `int` would have `i` here.
- The total size (in bytes) required to pass all the parameters. This includes the two hidden parameters (the object `self` and the method selector `_cmd`).
- Each argument, with the type encoding, followed by the offset (in bytes) of the argument in the list of parameters.

For example, a method with no arguments and returning `int` would have the signature `i8@0:4` if the size of a pointer is 4. The signature is interpreted as follows: the `i` is the return type (an `int`), the `8` is the total size of the parameters in bytes (two pointers each of size 4), the `@0` is the first parameter (an object at byte offset 0) and `:4` is the second parameter (a SEL at byte offset 4).

You can easily find more examples by running the “strings” program on an Objective-C object file compiled by GCC. You’ll see a lot of strings that look very much like `i8@0:4`. They are signatures of Objective-C methods.

9.4 Garbage Collection

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

Support for garbage collection with the GNU runtime has been added by using a powerful conservative garbage collector, known as the Boehm-Demers-Weiser conservative garbage collector.

To enable the support for it you have to configure the compiler using an additional argument, `--enable-objc-gc`. This will build the `boehm-gc` library, and build an additional runtime library which has several enhancements to support the garbage collector. The new library has a new name, `libobjc_gc.a` to not conflict with the non-garbage-collected library.

When the garbage collector is used, the objects are allocated using the so-called typed memory allocation mechanism available in the Boehm-Demers-Weiser collector. This mode requires precise information on where pointers are located inside objects. This information is computed once per class, immediately after the class has been initialized.

There is a new runtime function `class_ivar_set_gcinvisible()` which can be used to declare a so-called *weak pointer* reference. Such a pointer is basically hidden for the garbage

collector; this can be useful in certain situations, especially when you want to keep track of the allocated objects, yet allow them to be collected. This kind of pointers can only be members of objects, you cannot declare a global pointer as a weak reference. Every type which is a pointer type can be declared a weak pointer, including `id`, `Class` and `SEL`.

Here is an example of how to use this feature. Suppose you want to implement a class whose instances hold a weak pointer reference; the following class does this:

```
@interface WeakPointer : Object
{
    const void* weakPointer;
}

- initWithPointer:(const void*)p;
- (const void*)weakPointer;
@end

@implementation WeakPointer

+ (void)initialize
{
    if (self == objc_lookUpClass ("WeakPointer"))
        class_ivar_set_gcinvisible (self, "weakPointer", YES);
}

- initWithPointer:(const void*)p
{
    weakPointer = p;
    return self;
}

- (const void*)weakPointer
{
    return weakPointer;
}

@end
```

Weak pointers are supported through a new type character specifier represented by the ‘!’ character. The `class_ivar_set_gcinvisible()` function adds or removes this specifier to the string type description of the instance variable named as argument.

9.5 Constant String Objects

GNU Objective-C provides constant string objects that are generated directly by the compiler. You declare a constant string object by prefixing a C constant string with the character ‘@’:

```
id myString = @"this is a constant string object";
```

The constant string objects are by default instances of the `NXConstantString` class which is provided by the GNU Objective-C runtime. To get the definition of this class you must include the `objc/NXConstStr.h` header file.

User defined libraries may want to implement their own constant string class. To be able to support them, the GNU Objective-C compiler provides a new command line op-

tions `-fconstant-string-class=class-name`. The provided class should adhere to a strict structure, the same as `NXConstantString`'s structure:

```
@interface MyConstantStringClass
{
    Class isa;
    char *c_string;
    unsigned int len;
}
@end
```

`NXConstantString` inherits from `Object`; user class libraries may choose to inherit the customized constant string class from a different class than `Object`. There is no requirement in the methods the constant string class has to implement, but the final ivar layout of the class must be compatible with the given structure.

When the compiler creates the statically allocated constant string object, the `c_string` field will be filled by the compiler with the string; the `length` field will be filled by the compiler with the string length; the `isa` pointer will be filled with `NULL` by the compiler, and it will later be fixed up automatically at runtime by the GNU Objective-C runtime library to point to the class which was set by the `-fconstant-string-class` option when the object file is loaded (if you wonder how it works behind the scenes, the name of the class to use, and the list of static objects to fixup, are stored by the compiler in the object file in a place where the GNU runtime library will find them at runtime).

As a result, when a file is compiled with the `-fconstant-string-class` option, all the constant string objects will be instances of the class specified as argument to this option. It is possible to have multiple compilation units referring to different constant string classes, neither the compiler nor the linker impose any restrictions in doing this.

9.6 compatibility_alias

The keyword `@compatibility_alias` allows you to define a class name as equivalent to another class name. For example:

```
@compatibility_alias WOAApplication GSWApplication;
```

tells the compiler that each time it encounters `WOApplication` as a class name, it should replace it with `GSWApplication` (that is, `WOApplication` is just an alias for `GSWApplication`).

There are some constraints on how this can be used—

- `WOApplication` (the alias) must not be an existing class;
- `GSWApplication` (the real class) must be an existing class.

9.7 Exceptions

GNU Objective-C provides exception support built into the language, as in the following example:

```
@try {
    ...
    @throw expr;
    ...
}
```

```

}
@catch (AnObjCClass *exc) {
    ...
    @throw expr;
    ...
    @throw;
    ...
}
@catch (AnotherClass *exc) {
    ...
}
@catch (id allOthers) {
    ...
}
@finally {
    ...
    @throw expr;
    ...
}

```

The `@throw` statement may appear anywhere in an Objective-C or Objective-C++ program; when used inside of a `@catch` block, the `@throw` may appear without an argument (as shown above), in which case the object caught by the `@catch` will be rethrown.

Note that only (pointers to) Objective-C objects may be thrown and caught using this scheme. When an object is thrown, it will be caught by the nearest `@catch` clause capable of handling objects of that type, analogously to how `catch` blocks work in C++ and Java. A `@catch(id ...)` clause (as shown above) may also be provided to catch any and all Objective-C exceptions not caught by previous `@catch` clauses (if any).

The `@finally` clause, if present, will be executed upon exit from the immediately preceding `@try ... @catch` section. This will happen regardless of whether any exceptions are thrown, caught or rethrown inside the `@try ... @catch` section, analogously to the behavior of the `finally` clause in Java.

There are several caveats to using the new exception mechanism:

- The `-fobjc-exceptions` command line option must be used when compiling Objective-C files that use exceptions.
- With the GNU runtime, exceptions are always implemented as “native” exceptions and it is recommended that the `-fexceptions` and `-shared-libgcc` options are used when linking.
- With the NeXT runtime, although currently designed to be binary compatible with `NS_HANDLER`-style idioms provided by the `NSException` class, the new exceptions can only be used on Mac OS X 10.3 (Panther) and later systems, due to additional functionality needed in the NeXT Objective-C runtime.
- As mentioned above, the new exceptions do not support handling types other than Objective-C objects. Furthermore, when used from Objective-C++, the Objective-C exception model does not interoperate with C++ exceptions at this time. This means you cannot `@throw` an exception from Objective-C and `catch` it in C++, or vice versa (i.e., `throw ... @catch`).

9.8 Synchronization

GNU Objective-C provides support for synchronized blocks:

```
@synchronized (ObjCClass *guard) {
    ...
}
```

Upon entering the `@synchronized` block, a thread of execution shall first check whether a lock has been placed on the corresponding `guard` object by another thread. If it has, the current thread shall wait until the other thread relinquishes its lock. Once `guard` becomes available, the current thread will place its own lock on it, execute the code contained in the `@synchronized` block, and finally relinquish the lock (thereby making `guard` available to other threads).

Unlike Java, Objective-C does not allow for entire methods to be marked `@synchronized`. Note that throwing exceptions out of `@synchronized` blocks is allowed, and will cause the guarding object to be unlocked properly.

Because of the interactions between synchronization and exception handling, you can only use `@synchronized` when compiling with exceptions enabled, that is with the command line option `-fobjc-exceptions`.

9.9 Fast Enumeration

9.9.1 Using Fast Enumeration

GNU Objective-C provides support for the fast enumeration syntax:

```
id array = ...;
id object;

for (object in array)
{
    /* Do something with 'object' */
}
```

`array` needs to be an Objective-C object (usually a collection object, for example an array, a dictionary or a set) which implements the “Fast Enumeration Protocol” (see below). If you are using a Foundation library such as GNUstep Base or Apple Cocoa Foundation, all collection objects in the library implement this protocol and can be used in this way.

The code above would iterate over all objects in `array`. For each of them, it assigns it to `object`, then executes the `Do something with 'object'` statements.

Here is a fully worked-out example using a Foundation library (which provides the implementation of `NSArray`, `NSString` and `NSLog`):

```
NSArray *array = [NSArray arrayWithObjects: @"1", @"2", @"3", nil];
NSString *object;

for (object in array)
    NSLog (@"Iterating over %@", object);
```

9.9.2 C99-Like Fast Enumeration Syntax

A c99-like declaration syntax is also allowed:

```
id array = ...;
```

```

for (id object in array)
{
    /* Do something with 'object' */
}

```

this is completely equivalent to:

```

id array = ...;

{
    id object;
    for (object in array)
    {
        /* Do something with 'object' */
    }
}

```

but can save some typing.

Note that the option `-std=c99` is not required to allow this syntax in Objective-C.

9.9.3 Fast Enumeration Details

Here is a more technical description with the gory details. Consider the code

```

for (object expression in collection expression)
{
    statements
}

```

here is what happens when you run it:

- *collection expression* is evaluated exactly once and the result is used as the collection object to iterate over. This means it is safe to write code such as `for (object in [NSDictionary keyEnumerator])`
- the iteration is implemented by the compiler by repeatedly getting batches of objects from the collection object using the fast enumeration protocol (see below), then iterating over all objects in the batch. This is faster than a normal enumeration where objects are retrieved one by one (hence the name “fast enumeration”).
- if there are no objects in the collection, then *object expression* is set to `nil` and the loop immediately terminates.
- if there are objects in the collection, then for each object in the collection (in the order they are returned) *object expression* is set to the object, then *statements* are executed.
- *statements* can contain `break` and `continue` commands, which will abort the iteration or skip to the next loop iteration as expected.
- when the iteration ends because there are no more objects to iterate over, *object expression* is set to `nil`. This allows you to determine whether the iteration finished because a `break` command was used (in which case *object expression* will remain set to the last object that was iterated over) or because it iterated over all the objects (in which case *object expression* will be set to `nil`).
- *statements* must not make any changes to the collection object; if they do, it is a hard error and the fast enumeration terminates by invoking `objc_enumerationMutation`, a runtime function that normally aborts the program but which can be customized by Foundation libraries via `objc_set_mutation_handler` to do something different, such as raising an exception.

9.9.4 Fast Enumeration Protocol

If you want your own collection object to be usable with fast enumeration, you need to have it implement the method

```
- (unsigned long) countByEnumeratingWithState: (NSFastEnumerationState *)state
                                objects: (id *)objects
                                count: (unsigned long)len;
```

where `NSFastEnumerationState` must be defined in your code as follows:

```
typedef struct
{
    unsigned long state;
    id            *itemsPtr;
    unsigned long *mutationsPtr;
    unsigned long extra[5];
} NSFastEnumerationState;
```

If no `NSFastEnumerationState` is defined in your code, the compiler will automatically replace `NSFastEnumerationState *` with `struct __objcFastEnumerationState *`, where that type is silently defined by the compiler in an identical way. This can be confusing and we recommend that you define `NSFastEnumerationState` (as shown above) instead.

The method is called repeatedly during a fast enumeration to retrieve batches of objects. Each invocation of the method should retrieve the next batch of objects.

The return value of the method is the number of objects in the current batch; this should not exceed `len`, which is the maximum size of a batch as requested by the caller. The batch itself is returned in the `itemsPtr` field of the `NSFastEnumerationState` struct.

To help with returning the objects, the `objects` array is a C array preallocated by the caller (on the stack) of size `len`. In many cases you can put the objects you want to return in that `objects` array, then do `itemsPtr = objects`. But you don't have to; if your collection already has the objects to return in some form of C array, it could return them from there instead.

The `state` and `extra` fields of the `NSFastEnumerationState` structure allows your collection object to keep track of the state of the enumeration. In a simple array implementation, `state` may keep track of the index of the last object that was returned, and `extra` may be unused.

The `mutationsPtr` field of the `NSFastEnumerationState` is used to keep track of mutations. It should point to a number; before working on each object, the fast enumeration loop will check that this number has not changed. If it has, a mutation has happened and the fast enumeration will abort. So, `mutationsPtr` could be set to point to some sort of version number of your collection, which is increased by one every time there is a change (for example when an object is added or removed). Or, if you are content with less strict mutation checks, it could point to the number of objects in your collection or some other value that can be checked to perform an approximate check that the collection has not been mutated.

Finally, note how we declared the `len` argument and the return value to be of type `unsigned long`. They could also be declared to be of type `unsigned int` and everything would still work.

9.10 Messaging with the GNU Objective-C Runtime

This section is specific for the GNU Objective-C runtime. If you are using a different runtime, you can skip it.

The implementation of messaging in the GNU Objective-C runtime is designed to be portable, and so is based on standard C.

Sending a message in the GNU Objective-C runtime is composed of two separate steps. First, there is a call to the lookup function, `objc_msg_lookup()` (or, in the case of messages to super, `objc_msg_lookup_super()`). This runtime function takes as argument the receiver and the selector of the method to be called; it returns the `IMP`, that is a pointer to the function implementing the method. The second step of method invocation consists of casting this pointer function to the appropriate function pointer type, and calling the function pointed to it with the right arguments.

For example, when the compiler encounters a method invocation such as `[object init]`, it compiles it into a call to `objc_msg_lookup(object, @selector(init))` followed by a cast of the returned value to the appropriate function pointer type, and then it calls it.

9.10.1 Dynamically Registering Methods

If `objc_msg_lookup()` does not find a suitable method implementation, because the receiver does not implement the required method, it tries to see if the class can dynamically register the method.

To do so, the runtime checks if the class of the receiver implements the method

```
+ (BOOL) resolveInstanceMethod: (SEL)selector;
```

in the case of an instance method, or

```
+ (BOOL) resolveClassMethod: (SEL)selector;
```

in the case of a class method. If the class implements it, the runtime invokes it, passing as argument the selector of the original method, and if it returns **YES**, the runtime tries the lookup again, which could now succeed if a matching method was added dynamically by `+resolveInstanceMethod:` or `+resolveClassMethod:`.

This allows classes to dynamically register methods (by adding them to the class using `class_addMethod`) when they are first called. To do so, a class should implement `+resolveInstanceMethod:` (or, depending on the case, `+resolveClassMethod:`) and have it recognize the selectors of methods that can be registered dynamically at runtime, register them, and return **YES**. It should return **NO** for methods that it does not dynamically register at runtime.

If `+resolveInstanceMethod:` (or `+resolveClassMethod:`) is not implemented or returns **NO**, the runtime then tries the forwarding hook.

Support for `+resolveInstanceMethod:` and `resolveClassMethod:` was added to the GNU Objective-C runtime in GCC version 4.6.

9.10.2 Forwarding Hook

The GNU Objective-C runtime provides a hook, called `__objc_msg_forward2`, which is called by `objc_msg_lookup()` when it cannot find a method implementation in the runtime tables and after calling `+resolveInstanceMethod:` and `+resolveClassMethod:` has been attempted and did not succeed in dynamically registering the method.

To configure the hook, you set the global variable `__objc_msg_forward2` to a function with the same argument and return types of `objc_msg_lookup()`. When `objc_msg_lookup()` cannot find a method implementation, it invokes the hook function you provided to get a method implementation to return. So, in practice `__objc_msg_forward2` allows you to extend `objc_msg_lookup()` by adding some custom code that is called to do a further lookup when no standard method implementation can be found using the normal lookup.

This hook is generally reserved for “Foundation” libraries such as GNUstep Base, which use it to implement their high-level method forwarding API, typically based around the `forwardInvocation:` method. So, unless you are implementing your own “Foundation” library, you should not set this hook.

In a typical forwarding implementation, the `__objc_msg_forward2` hook function determines the argument and return type of the method that is being looked up, and then creates a function that takes these arguments and has that return type, and returns it to the caller. Creating this function is non-trivial and is typically performed using a dedicated library such as `libffi`.

The forwarding method implementation thus created is returned by `objc_msg_lookup()` and is executed as if it was a normal method implementation. When the forwarding method implementation is called, it is usually expected to pack all arguments into some sort of object (typically, an `NSInvocation` in a “Foundation” library), and hand it over to the programmer (`forwardInvocation:`) who is then allowed to manipulate the method invocation using a high-level API provided by the “Foundation” library. For example, the programmer may want to examine the method invocation arguments and name and potentially change them before forwarding the method invocation to one or more local objects (`performInvocation:`) or even to remote objects (by using Distributed Objects or some other mechanism). When all this completes, the return value is passed back and must be returned correctly to the original caller.

Note that the GNU Objective-C runtime currently provides no support for method forwarding or method invocations other than the `__objc_msg_forward2` hook.

If the forwarding hook does not exist or returns `NULL`, the runtime currently attempts forwarding using an older, deprecated API, and if that fails, it aborts the program. In future versions of the GNU Objective-C runtime, the runtime will immediately abort.

10 Binary Compatibility

Binary compatibility encompasses several related concepts:

application binary interface (ABI)

The set of runtime conventions followed by all of the tools that deal with binary representations of a program, including compilers, assemblers, linkers, and language runtime support. Some ABIs are formal with a written specification, possibly designed by multiple interested parties. Others are simply the way things are actually done by a particular set of tools.

ABI conformance

A compiler conforms to an ABI if it generates code that follows all of the specifications enumerated by that ABI. A library conforms to an ABI if it is implemented according to that ABI. An application conforms to an ABI if it is built using tools that conform to that ABI and does not contain source code that specifically changes behavior specified by the ABI.

calling conventions

Calling conventions are a subset of an ABI that specify of how arguments are passed and function results are returned.

interoperability

Different sets of tools are interoperable if they generate files that can be used in the same program. The set of tools includes compilers, assemblers, linkers, libraries, header files, startup files, and debuggers. Binaries produced by different sets of tools are not interoperable unless they implement the same ABI. This applies to different versions of the same tools as well as tools from different vendors.

intercallability

Whether a function in a binary built by one set of tools can call a function in a binary built by a different set of tools is a subset of interoperability.

implementation-defined features

Language standards include lists of implementation-defined features whose behavior can vary from one implementation to another. Some of these features are normally covered by a platform's ABI and others are not. The features that are not covered by an ABI generally affect how a program behaves, but not intercallability.

compatibility

Conformance to the same ABI and the same behavior of implementation-defined features are both relevant for compatibility.

The application binary interface implemented by a C or C++ compiler affects code generation and runtime support for:

- size and alignment of data types
- layout of structured types
- calling conventions

- register usage conventions
- interfaces for runtime arithmetic support
- object file formats

In addition, the application binary interface implemented by a C++ compiler affects code generation and runtime support for:

- name mangling
- exception handling
- invoking constructors and destructors
- layout, alignment, and padding of classes
- layout and alignment of virtual tables

Some GCC compilation options cause the compiler to generate code that does not conform to the platform's default ABI. Other options cause different program behavior for implementation-defined features that are not covered by an ABI. These options are provided for consistency with other compilers that do not follow the platform's default ABI or the usual behavior of implementation-defined features for the platform. Be very careful about using such options.

Most platforms have a well-defined ABI that covers C code, but ABIs that cover C++ functionality are not yet common.

Starting with GCC 3.2, GCC binary conventions for C++ are based on a written, vendor-neutral C++ ABI that was designed to be specific to 64-bit Itanium but also includes generic specifications that apply to any platform. This C++ ABI is also implemented by other compiler vendors on some platforms, notably GNU/Linux and BSD systems. We have tried hard to provide a stable ABI that will be compatible with future GCC releases, but it is possible that we will encounter problems that make this difficult. Such problems could include different interpretations of the C++ ABI by different vendors, bugs in the ABI, or bugs in the implementation of the ABI in different compilers. GCC's `-Wabi` switch warns when G++ generates code that is probably not compatible with the C++ ABI.

The C++ library used with a C++ compiler includes the Standard C++ Library, with functionality defined in the C++ Standard, plus language runtime support. The runtime support is included in a C++ ABI, but there is no formal ABI for the Standard C++ Library. Two implementations of that library are interoperable if one follows the de-facto ABI of the other and if they are both built with the same compiler, or with compilers that conform to the same ABI for C++ compiler and runtime support.

When G++ and another C++ compiler conform to the same C++ ABI, but the implementations of the Standard C++ Library that they normally use do not follow the same ABI for the Standard C++ Library, object files built with those compilers can be used in the same program only if they use the same C++ library. This requires specifying the location of the C++ library header files when invoking the compiler whose usual library is not being used. The location of GCC's C++ header files depends on how the GCC build was configured, but can be seen by using the G++ `-v` option. With default configuration options for G++ 3.3 the compile line for a different C++ compiler needs to include

```
-Igcc_install_directory/include/c++/3.3
```

Similarly, compiling code with G++ that must use a C++ library other than the GNU C++ library requires specifying the location of the header files for that other library.

The most straightforward way to link a program to use a particular C++ library is to use a C++ driver that specifies that C++ library by default. The `g++` driver, for example, tells the linker where to find GCC's C++ library (`libstdc++`) plus the other libraries and startup files it needs, in the proper order.

If a program must use a different C++ library and it's not possible to do the final link using a C++ driver that uses that library by default, it is necessary to tell `g++` the location and name of that library. It might also be necessary to specify different startup files and other runtime support libraries, and to suppress the use of GCC's support libraries with one or more of the options `-nostdlib`, `-nostartfiles`, and `-nodefaultlibs`.

11 gcov—a Test Coverage Program

`gcov` is a tool you can use in conjunction with GCC to test code coverage in your programs.

11.1 Introduction to gcov

`gcov` is a test coverage program. Use it in concert with GCC to analyze your programs to help create more efficient, faster running code and to discover untested parts of your program. You can use `gcov` as a profiling tool to help discover where your optimization efforts will best affect your code. You can also use `gcov` along with the other profiling tool, `gprof`, to assess which parts of your code use the greatest amount of computing time.

Profiling tools help you analyze your code's performance. Using a profiler such as `gcov` or `gprof`, you can find out some basic performance statistics, such as:

- how often each line of code executes
- what lines of code are actually executed
- how much computing time each section of code uses

Once you know these things about how your code works when compiled, you can look at each module to see which modules should be optimized. `gcov` helps you determine where to work on optimization.

Software developers also use coverage testing in concert with testsuites, to make sure software is actually good enough for a release. Testsuites can verify that a program works as expected; a coverage program tests to see how much of the program is exercised by the testsuite. Developers can then determine what kinds of test cases need to be added to the testsuites to create both better testing and a better final product.

You should compile your code without optimization if you plan to use `gcov` because the optimization, by combining some lines of code into one function, may not give you as much information as you need to look for 'hot spots' where the code is using a great deal of computer time. Likewise, because `gcov` accumulates statistics by line (at the lowest resolution), it works best with a programming style that places only one statement on each line. If you use complicated macros that expand to loops or to other control structures, the statistics are less helpful—they only report on the line where the macro call appears. If your complex macros behave like functions, you can replace them with inline functions to solve this problem.

`gcov` creates a logfile called `sourcefile.gcov` which indicates how many times each line of a source file `sourcefile.c` has executed. You can use these logfiles along with `gprof` to aid in fine-tuning the performance of your programs. `gprof` gives timing information you can use along with the information you get from `gcov`.

`gcov` works only on code compiled with GCC. It is not compatible with any other profiling or test coverage mechanism.

11.2 Invoking gcov

```
gcov [options] files
```

`gcov` accepts the following options:

-a

--all-blocks

Write individual execution counts for every basic block. Normally gcov outputs execution counts only for the main blocks of a line. With this option you can determine if blocks within a single line are not being executed.

-b

--branch-probabilities

Write branch frequencies to the output file, and write branch summary info to the standard output. This option allows you to see how often each branch in your program was taken. Unconditional branches will not be shown, unless the **-u** option is given.

-c

--branch-counts

Write branch frequencies as the number of branches taken, rather than the percentage of branches taken.

-g

--conditions

Write condition coverage to the output file, and write condition summary info to the standard output. This option allows you to see if the conditions in your program at least once had an independent effect on the outcome of the boolean expression (modified condition/decision coverage). This requires you to compile the source with **-fcondition-coverage**.

-e

--prime-paths

Write path coverage to the output file, and write path summary info to the standard output. This option allows you to see how many prime paths were taken at least once. A path is a sequence of basic blocks. A path is simple if it has no repeated blocks (no loops) except maybe the first and last block, and prime if it is a simple path of maximal length. For the regular output this option only includes the number of paths covered. For more fine grained information on paths you can use **--prime-paths-lines** or **--prime-paths-source**. With **--json-format** all path details are included in the output. This requires you to compile the source with **-fpath-coverage**.

--prime-paths-lines [=type]

Write path coverage to the output file, and write path summary info to the standard output. This option allows you to see how many prime paths were taken at least once, and dense report on the covered or uncovered paths and how to cover them. This mode is useful for automated reporting and progress tracking. *type* may be omitted, or one of:

- *uncovered* - Include the uncovered (not taken) paths. This is the default.
- *covered* - Include the covered (taken) paths.
- *both* - Include all paths. This is equivalent to using both *covered* and *uncovered*.

This is an example of `--prime-paths-lines` output:

```
paths covered 12 of 15
path  2 not covered: lines 8 8(false) 11(true) 11 13(true) 13(true) 14 17
path  3 not covered: lines 8 8(false) 11(true) 11 13(true) 13(false) 16 17
path  4 not covered: lines 8 8(false) 11(true) 11 13(false) 16 17
```

This means to cover path 2 you must run lines 8, 11, 13, 14, and 17, evaluating the decision at 8 false and the decisions at 11 and 13 to false.

`--prime-paths-source [=type]`

Write path coverage to the output file, and write path summary info to the standard output. This option allows you to see how many prime paths were taken at least once, and detailed report on the uncovered paths and how to cover them. This mode is useful for understanding paths and interactions between sections of your program. *type* may be omitted, or one of:

- *uncovered* - Include the uncovered (not taken) paths. This is the default.
- *covered* - Include the covered (taken) paths.
- *both* - Include all paths. This is equivalent to using both *covered* and *uncovered*.

This is an example of `--prime-paths-source` output:

```
path 10 not covered:
BB  3:      8:  for (i = 0; i < 10; i++)
BB  3:      9:    total += i;
BB  4: (false) 8:  for (i = 0; i < 10; i++)
BB  5: (true) 11:  int v = total > 100 ? 1 : 2;
BB  6:      11:  int v = total > 100 ? 1 : 2;
BB  8: (false) 13:  if (total != 45 && v == 1)
BB 11:      16:    printf ("Success\n");
BB 12:      17:  return 0;
```

The first (BB) column is the sequence of basic blocks (see `-w`). The middle column (true/false) is the decision for that line. The third column is the line number. The fourth column is the line itself. These lines must be run in this order to cover path 10.

`-d`

`--display-progress`

Display the progress on the standard output.

`-f`

`--function-summaries`

Output summaries for each function in addition to the file level summary.

`--include regex`

Include functions matching *regex*. This option makes `gcov` only report on functions that match the extended regular expression *regex*. This flag can be combined with `--exclude`. If a function matches both includes and excludes, the last include/exclude applies. By default `gcov` reports on all functions, but if a `--include` is used then only functions matching the include will be reported.

`--exclude regex`

Exclude functions matching *regex*. This option makes `gcov` not report on functions that match the extended regular expression *regex*. This flag can be com-

lined with `--include`. If a function matches both includes and excludes, the last include/exclude applies. By default `gcov` reports on all functions, and if `--exclude` is used then functions matching it will be omitted.

`-h`

`--help` Display help about using `gcov` (on the standard output), and exit without doing any further processing.

`-j`

`--json-format`

Output `gcov` file in an easy-to-parse JSON intermediate format which does not require source code for generation. The JSON file is compressed with `gzip` compression algorithm and the files have `.gcov.json.gz` extension.

Structure of the JSON is following:

```
{
  "current_working_directory": "foo/bar",
  "data_file": "a.out",
  "format_version": "2",
  "gcc_version": "11.1.1 20210510"
  "files": ["$file"]
}
```

Fields of the root element have following semantics:

- *current_working_directory*: working directory where a compilation unit was compiled
- *data_file*: name of the data file (GCDA)
- *format_version*: semantic version of the format

Changes in version 2:

- *calls*: information about function calls is added
- *gcc_version*: version of the GCC compiler

Each *file* has the following form:

```
{
  "file": "a.c",
  "functions": ["$function"],
  "lines": ["$line"]
}
```

Fields of the *file* element have following semantics:

- *file_name*: name of the source file

Each *function* has the following form:

```
{
  "blocks": 2,
  "blocks_executed": 2,
  "demangled_name": "foo",
  "end_column": 1,
  "end_line": 4,
  "execution_count": 1,
  "name": "foo",
  "start_column": 5,
  "start_line": 1
}
```

Fields of the *function* element have following semantics:

- *blocks*: number of blocks that are in the function
- *blocks_executed*: number of executed blocks of the function
- *demangled_name*: demangled name of the function
- *end_column*: column in the source file where the function ends
- *end_line*: line in the source file where the function ends
- *execution_count*: number of executions of the function
- *name*: name of the function
- *start_column*: column in the source file where the function begins
- *start_line*: line in the source file where the function begins

Note that line numbers and column numbers number from 1. In the current implementation, *start_line* and *start_column* do not include any template parameters and the leading return type but that this is likely to be fixed in the future.

Each *line* has the following form:

```
{
  "block_ids": ["$block_id"],
  "branches": ["$branch"],
  "calls": ["$call"],
  "count": 2,
  "conditions": ["$condition"],
  "line_number": 15,
  "unexecuted_block": false,
  "function_name": "foo",
}
```

Branches and calls are present only with *-b* option. Fields of the *line* element have following semantics:

- *block_ids*: IDs of basic blocks that belong to the line
- *count*: number of executions of the line
- *line_number*: line number
- *unexecuted_block*: flag whether the line contains an unexecuted block (not all statements on the line are executed)
- *function_name*: a name of a function this *line* belongs to (for a line with an inlined statements can be not set)

Each *branch* has the following form:

```
{
  "count": 11,
  "destination_block_id": 17,
  "fallthrough": true,
  "source_block_id": 13,
  "throw": false
}
```

Fields of the *branch* element have following semantics:

- *count*: number of executions of the branch
- *fallthrough*: true when the branch is a fall through branch

- *throw*: true when the branch is an exceptional branch
- *isource_block_id*: ID of the basic block where this branch happens
- *destination_block_id*: ID of the basic block this branch jumps to

Each *call* has the following form:

```
{
  "destination_block_id": 1,
  "returned": 11,
  "source_block_id": 13
}
```

Fields of the *call* element have following semantics:

- *returned*: number of times a function call returned (call count is equal to *line::count*)
- *isource_block_id*: ID of the basic block where this call happens
- *destination_block_id*: ID of the basic block this calls continues after return

Each *condition* has the following form:

```
{
  "count": 4,
  "covered": 2,
  "not_covered_false": [],
  "not_covered_true": [0, 1],
}
```

Fields of the *condition* element have following semantics:

- *count*: number of condition outcomes in this expression
- *covered*: number of covered condition outcomes in this expression
- *not_covered_true*: terms, by index, not seen as true in this expression
- *not_covered_false*: terms, by index, not seen as false in this expression

-H

--human-readable

Write counts in human readable format (like 24.6k).

-k

--use-colors

Use colors for lines of code that have zero coverage. We use red color for non-exceptional lines and cyan for exceptional. Same colors are used for basic blocks with **-a** option.

-l

--long-file-names

Create long file names for included source files. For example, if the header file *x.h* contains code, and was included in the file *a.c*, then running *gcov* on the file *a.c* will produce an output file called *a.c##x.h.gcov* instead of *x.h.gcov*. This can be useful if *x.h* is included in multiple source files and you want to see the individual contributions. If you use the **-p** option, both the including and included file names will be complete path names.

-m
--demangled-names
Display demangled function names in output. The default is to show mangled function names.

-M
--filter-on-demangled
Make **--include** and **--exclude** match demangled names. This does only affects the matching and does not imply **--demangled-names**, but it can safely be combined with it.

-n
--no-output
Do not create the gcov output file.

-o *directory|file*
--object-directory *directory*
--object-file *file*
Specify either the directory containing the gcov data files, or the object path name. The **.gcno**, and **.gcda** data files are searched for using this option. If a directory is specified, the data files are in that directory and named after the input file name, without its extension. If a file is specified here, the data files are named after that file, without its extension.

-p
--preserve-paths
Preserve complete path information in the names of generated **.gcov** files. Without this option, just the filename component is used. With this option, all directories are used, with **/** characters translated to **#** characters, **.** directory components removed and unremoveable **..** components renamed to **^**. This is useful if sourcefiles are in several different directories.

-q
--use-hotness-colors
Emit perf-like colored output for hot lines. Legend of the color scale is printed at the very beginning of the output file.

-r
--relative-only
Only output information about source files with a relative pathname (after source prefix elision). Absolute paths are usually system header files and coverage of any inline functions therein is normally uninteresting.

-s *directory*
--source-prefix *directory*
A prefix for source file names to remove when generating the output coverage files. This option is useful when building in a separate directory, and the pathname to the source directory is not wanted when determining the output file names. Note that this prefix detection is applied before determining whether the source file is absolute.

```

-t
--stdout  Output to standard output instead of output files.

-u
--unconditional-branches
           When branch probabilities are given, include those of unconditional branches.
           Unconditional branches are normally not interesting.

-v
--version
           Display the gcov version number (on the standard output), and exit without
           doing any further processing.

-w
--verbose
           Print verbose informations related to basic blocks and arcs.

-x
--hash-filenames
           When using -preserve-paths, gcov uses the full pathname of the source
           files to create an output filename. This can lead to long filenames that
           can overflow filesystem limits. This option creates names of the form
           source-file###md5.gcov, where the source-file component is the final filename
           part and the md5 component is calculated from the full mangled name
           that would have been used otherwise. The option is an alternative to the
           -preserve-paths on systems which have a filesystem limit.

```

gcov should be run with the current directory the same as that when you invoked the compiler. Otherwise it will not be able to locate the source files. gcov produces files called *mangledname.gcov* in the current directory. These contain the coverage information of the source file they correspond to. One .gcov file is produced for each source (or header) file containing code, which was compiled to produce the data files. The *mangledname* part of the output file name is usually simply the source file name, but can be something more complicated if the *-l* or *-p* options are given. Refer to those options for details.

If you invoke gcov with multiple input files, the contributions from each input file are summed. Typically you would invoke it with the same list of files as the final link of your executable.

The .gcov files contain the ':' separated fields along with program source code. The format is

```
execution_count:line_number:source line text
```

Additional block information may succeed each line, when requested by command line option. The *execution_count* is '-' for lines containing no code. Unexecuted lines are marked '####' or '====', depending on whether they are reachable by non-exceptional paths or only exceptional paths such as C++ exception handlers, respectively. Given the *-a* option, unexecuted blocks are marked '\$\$\$\$' or '%%%', depending on whether a basic block is reachable via non-exceptional or exceptional paths. Executed basic blocks having a statement with zero *execution_count* end with '*' character and are colored with magenta color with the *-k* option. This functionality is not supported in Ada.

Note that GCC can completely remove the bodies of functions that are not needed – for instance if they are inlined everywhere. Such functions are marked with ‘-’, which can be confusing. Use the `-fkeep-inline-functions` and `-fkeep-static-functions` options to retain these functions and allow gcov to properly show their *execution_count*.

Some lines of information at the start have *line_number* of zero. These preamble lines are of the form

```
-:0:tag:value
```

The ordering and number of these preamble lines will be augmented as gcov development progresses — do not rely on them remaining unchanged. Use *tag* to locate a particular preamble line.

The additional block information is of the form

```
tag information
```

The *information* is human readable, but designed to be simple enough for machine parsing too.

When printing percentages, 0% and 100% are only printed when the values are *exactly* 0% and 100% respectively. Other values which would conventionally be rounded to 0% or 100% are instead printed as the nearest non-boundary value.

When using gcov, you must first compile your program with a special GCC option ‘`--coverage`’. This tells the compiler to generate additional information needed by gcov (basically a flow graph of the program) and also includes additional code in the object files for generating the extra profiling information needed by gcov. These additional files are placed in the directory where the object file is located.

Running the program will cause profile output to be generated. For each source file compiled with `-fprofile-arcs`, an accompanying `.gcda` file will be placed in the object file directory.

Running gcov with your program’s source file names as arguments will now produce a listing of the code along with frequency of execution for each line. For example, if your program is called `tmp.cpp`, this is what you see when you use the basic gcov facility:

```
$ g++ --coverage tmp.cpp -c
$ g++ --coverage tmp.o
$ a.out
$ gcov tmp.cpp -m
File 'tmp.cpp'
Lines executed:92.86% of 14
Creating 'tmp.cpp.gcov'
```

The file `tmp.cpp.gcov` contains output from gcov. Here is a sample:

```
-: 0:Source:tmp.cpp
-: 0:Working directory:/home/gcc/testcase
-: 0:Graph:tmp.gcno
-: 0:Data:tmp.gcda
-: 0:Runs:1
-: 0:Programs:1
-: 1:#include <stdio.h>
-: 2:
-: 3:template<class T>
-: 4:class Foo
-: 5:{
-: 6: public:
```

```

1*:    7:  Foo(): b (1000) {}
-----
Foo<char>::Foo():
#####:    7:  Foo(): b (1000) {}
-----
Foo<int>::Foo():
1:      7:  Foo(): b (1000) {}
-----
2*:    8:  void inc () { b++; }
-----
Foo<char>::inc():
#####:    8:  void inc () { b++; }
-----
Foo<int>::inc():
2:      8:  void inc () { b++; }
-----
-:     9:
-:    10: private:
-:    11: int b;
-:    12:};
-:    13:
-:   14:template class Foo<int>;
-:   15:template class Foo<char>;
-:    16:
-:    17:int
1:    18:main (void)
-:    19:{
-:    20: int i, total;
1:    21: Foo<int> counter;
-:    22:
1:    23: counter.inc();
1:    24: counter.inc();
1:    25: total = 0;
-:    26:
11:   27: for (i = 0; i < 10; i++)
10:   28:     total += i;
-:    29:
1*:   30: int v = total > 100 ? 1 : 2;
-:    31:
1:    32: if (total != 45)
#####:   33:     printf ("Failure\n");
-:    34: else
1:    35:     printf ("Success\n");
1:    36: return 0;
-:    37:}

```

Note that line 7 is shown in the report multiple times. First occurrence presents total number of execution of the line and the next two belong to instances of class Foo constructors. As you can also see, line 30 contains some unexecuted basic blocks and thus execution count has asterisk symbol.

When you use the `-a` option, you will get individual block counts, and the output looks like this:

```

-:    0:Source:tmp.cpp
-:    0:Working directory:/home/gcc/testcase
-:    0:Graph:tmp.gcno
-:    0:Data:tmp.gcda
-:    0:Runs:1

```

```

-:      0:Programs:1
-:      1:#include <stdio.h>
-:      2:
-:      3:template<class T>
-:      4:class Foo
-:      5:{
-:      6: public:
1*:      7: Foo(): b (1000) {}
-----
Foo<char>::Foo():
#####:      7: Foo(): b (1000) {}
-----
Foo<int>::Foo():
1:      7: Foo(): b (1000) {}
-----
2*:      8: void inc () { b++; }
-----
Foo<char>::inc():
#####:      8: void inc () { b++; }
-----
Foo<int>::inc():
2:      8: void inc () { b++; }
-----
-:      9:
-:     10: private:
-:     11: int b;
-:     12:};
-:     13:
-:     14:template class Foo<int>;
-:     15:template class Foo<char>;
-:     16:
-:     17:int
1:     18:main (void)
-:     19:{
-:     20: int i, total;
1:     21: Foo<int> counter;
1:     21-block 0
-:     22:
1:     23: counter.inc();
1:     23-block 0
1:     24: counter.inc();
1:     24-block 0
1:     25: total = 0;
-:     26:
11:    27: for (i = 0; i < 10; i++)
1:     27-block 0
11:    27-block 1
10:    28:     total += i;
10:    28-block 0
-:     29:
1*:    30: int v = total > 100 ? 1 : 2;
1:     30-block 0
%%%%:    30-block 1
1:     30-block 2
-:     31:
1:     32: if (total != 45)
1:     32-block 0
#####:    33:     printf ("Failure\n");

```

```

%%%: 33-block 0
-: 34: else
1: 35: printf ("Success\n");
1: 35-block 0
1: 36: return 0;
1: 36-block 0
-: 37:}

```

In this mode, each basic block is only shown on one line – the last line of the block. A multi-line block will only contribute to the execution count of that last line, and other lines will not be shown to contain code, unless previous blocks end on those lines. The total execution count of a line is shown and subsequent lines show the execution counts for individual blocks that end on that line. After each block, the branch and call counts of the block will be shown, if the `-b` option is given.

Because of the way GCC instruments calls, a call count can be shown after a line with no individual blocks. As you can see, line 33 contains a basic block that was not executed.

When you use the `-b` option, your output looks like this:

```

-: 0:Source:tmp.cpp
-: 0:Working directory:/home/gcc/testcase
-: 0:Graph:tmp.gcno
-: 0:Data:tmp.gcda
-: 0:Runs:1
-: 0:Programs:1
-: 1:#include <stdio.h>
-: 2:
-: 3:template<class T>
-: 4:class Foo
-: 5:{
-: 6: public:
1*: 7: Foo(): b (1000) {}
-----
Foo<char>::Foo():
function Foo<char>::Foo() called 0 returned 0% blocks executed 0%
#####: 7: Foo(): b (1000) {}
-----
Foo<int>::Foo():
function Foo<int>::Foo() called 1 returned 100% blocks executed 100%
1: 7: Foo(): b (1000) {}
-----
2*: 8: void inc () { b++; }
-----
Foo<char>::inc():
function Foo<char>::inc() called 0 returned 0% blocks executed 0%
#####: 8: void inc () { b++; }
-----
Foo<int>::inc():
function Foo<int>::inc() called 2 returned 100% blocks executed 100%
2: 8: void inc () { b++; }
-----
-: 9:
-: 10: private:
-: 11: int b;
-: 12:};
-: 13:
-: 14:template class Foo<int>;
-: 15:template class Foo<char>;

```

```

-: 16:
-: 17:int
function main called 1 returned 100% blocks executed 81%
1: 18:main (void)
-: 19:{
-: 20: int i, total;
1: 21: Foo<int> counter;
call 0 returned 100%
branch 1 taken 100% (fallthrough)
branch 2 taken 0% (throw)
-: 22:
1: 23: counter.inc();
call 0 returned 100%
branch 1 taken 100% (fallthrough)
branch 2 taken 0% (throw)
1: 24: counter.inc();
call 0 returned 100%
branch 1 taken 100% (fallthrough)
branch 2 taken 0% (throw)
1: 25: total = 0;
-: 26:
11: 27: for (i = 0; i < 10; i++)
branch 0 taken 91% (fallthrough)
branch 1 taken 9%
10: 28: total += i;
-: 29:
1*: 30: int v = total > 100 ? 1 : 2;
branch 0 taken 0% (fallthrough)
branch 1 taken 100%
-: 31:
1: 32: if (total != 45)
branch 0 taken 0% (fallthrough)
branch 1 taken 100%
##### 33: printf ("Failure\n");
call 0 never executed
branch 1 never executed
branch 2 never executed
-: 34: else
1: 35: printf ("Success\n");
call 0 returned 100%
branch 1 taken 100% (fallthrough)
branch 2 taken 0% (throw)
1: 36: return 0;
-: 37:}

```

For each function, a line is printed showing how many times the function is called, how many times it returns and what percentage of the function's blocks were executed.

For each basic block, a line is printed after the last line of the basic block describing the branch or call that ends the basic block. There can be multiple branches and calls listed for a single source line if there are multiple basic blocks that end on that line. In this case, the branches and calls are each given a number. There is no simple way to map these branches and calls back to source constructs. In general, though, the lowest numbered branch or call will correspond to the leftmost construct on the source line.

For a branch, if it was executed at least once, then a percentage indicating the number of times the branch was taken divided by the number of times the branch was executed will be printed. Otherwise, the message “never executed” is printed.

For a call, if it was executed at least once, then a percentage indicating the number of times the call returned divided by the number of times the call was executed will be printed. This will usually be 100%, but may be less for functions that call `exit` or `longjmp`, and thus may not return every time they are called.

When you use the `-g` option, your output looks like this:

```
$ gcov -t -m -g tmp
-: 0:Source:tmp.cpp
-: 0:Graph:tmp.gcno
-: 0:Data:tmp.gcda
-: 0:Runs:1
-: 1:#include <stdio.h>
-: 2:
-: 3:int
1: 4:main (void)
-: 5:{
-: 6:  int i, total;
1: 7:  total = 0;
-: 8:
11: 9:  for (i = 0; i < 10; i++)
condition outcomes covered 2/2
10: 10:    total += i;
-: 11:
1*: 12:  int v = total > 100 ? 1 : 2;
condition outcomes covered 1/2
condition 0 not covered (true)
-: 13:
1*: 14:  if (total != 45 && v == 1)
condition outcomes covered 1/4
condition 0 not covered (true)
condition 1 not covered (true false)
#####: 15:    printf ("Failure\n");
-: 16:  else
1: 17:    printf ("Success\n");
1: 18:  return 0;
-: 19:}
```

For every condition the number of taken and total outcomes are printed, and if there are uncovered outcomes a line will be printed for each condition showing the uncovered outcome in parentheses. Conditions are identified by their index – index 0 is the left-most condition. In `a || (b && c)`, `a` is condition 0, `b` condition 1, and `c` condition 2.

An outcome is considered covered if it has an independent effect on the decision, also known as masking MC/DC (Modified Condition/Decision Coverage). In this example the decision evaluates to true and `a` is evaluated, but not covered. This is because `a` cannot affect the decision independently – both `a` and `b` must change value for the decision to change.

```
$ gcov -t -m -g tmp
-: 0:Source:tmp.c
-: 0:Graph:tmp.gcno
-: 0:Data:tmp.gcda
-: 0:Runs:1
-: 1:#include <stdio.h>
-: 2:
1: 3:int main()
-: 4:{
1: 5:  int a = 1;
```

```

1:      6:  int b = 0;
-:      7:
1:      8:  if (a && b)
condition outcomes covered 1/4
condition 0 not covered (true false)
condition 1 not covered (true)
#####:  9:      printf ("Success!\n");
-:      10:  else
1:      11:      printf ("Failure!\n");
-:      12:}

```

When you compile with `--coverage -fpath-coverage` and use the option `-e` your output looks like this:

```

$ gcov -t -e tmp
-:      0:Source:tmp.cpp
-:      0:Graph:tmp.gcno
-:      0:Data:tmp.gcda
-:      0:Runs:1
-:      1:#include <stdio.h>
-:      2:
paths covered 4 of 15
1:      3:int main ()
-:      4:{
-:      5:  int i, total;
1:      6:  total = 0;
-:      7:
11:     8:  for (i = 0; i < 10; i++)
10:     9:      total += i;
-:     10:
1*:    11:  int v = total > 100 ? 1 : 2;
-:     12:
1*:    13:  if (total != 45 && v == 1)
#####:  14:      printf ("Failure\n");
-:     15:  else
1:     16:      printf ("Success\n");
1:     17:  return 0;
-:     18:}

```

This output is useful to figure out roughly where coverage is missing and testing how different inputs change the coverage. The `--prime-paths-source` is a useful tool for understanding paths.

```

$ gcov -t --prime-paths-source tmp
-:      0:Source:tmp.cpp
-:      0:Graph:tmp.gcno
-:      0:Data:tmp.gcda
-:      0:Runs:1
-:      1:#include <stdio.h>
-:      2:
paths covered 4 of 15
path 1:
BB 2:      3:int main ()
BB 2:      6:  total = 0;
BB 2:      8:  for (i = 0; i < 10; i++)
BB 4: (false)  8:  for (i = 0; i < 10; i++)
BB 5: (true)   11:  int v = total > 100 ? 1 : 2;
BB 6:      11:  int v = total > 100 ? 1 : 2;
BB 8: (true)   13:  if (total != 45 && v == 1)
BB 9: (true)   13:  if (total != 45 && v == 1)

```

```
BB 10:      14:    printf ("Failure\n");
BB 12:      17:    return 0;
```

In this mode, gcov will print details on the missing paths. The first column lists the sequence of basic blocks (BB). The second column is the decision to take at that line if there is one. The final columns are the line number and the line itself. This is useful for understanding the paths, in particular those that are hard to cover or even unreachable. Lines may be repeated, for example the `for` loop, if the same line is a part of multiple basic blocks. This mode is intended for humans and good at understanding what code is exercised under testing or for given inputs. This output is quite verbose, and for focusing on specific functions it can be combined with the filters `--include` and `--exclude`.

A denser output is available with `--prime-paths-lines`, which looks like this:

```
-:      0:Source:tmp.cpp
-:      0:Graph:tmp.gcno
-:      0:Data:tmp.gcda
-:      0:Runs:1
-:      1:#include <stdio.h>
-:      2:

paths covered 4 of 15
path 1 not covered: lines 8 8(false) 11(true) 11 13(true) 13(true) 14 17
path 2 not covered: lines 8 8(false) 11(true) 11 13(true) 13(false) 16 17
path 3 not covered: lines 8 8(false) 11(true) 11 13(false) 16 17
path 4 not covered: lines 8 8(false) 11(false) 11 13(true) 13(true) 14 17
path 5 not covered: lines 8 8(false) 11(false) 11 13(true) 13(false) 16 17
path 6 not covered: lines 8 8(false) 11(false) 11 13(false) 16 17
path 8 not covered: lines 9 8(false) 11(true) 11 13(true) 13(true) 14 17
path 9 not covered: lines 9 8(false) 11(true) 11 13(true) 13(false) 16 17
path 10 not covered: lines 9 8(false) 11(true) 11 13(false) 16 17
path 11 not covered: lines 9 8(false) 11(false) 11 13(true) 13(true) 14 17
path 12 not covered: lines 9 8(false) 11(false) 11 13(true) 13(false) 16 17
1:      3:int main ()
-:      4:{
```

In this mode, every missing path is expanded using the lines and decisions like `--prime-paths-source` but printed on a single line. This mode provides a good overview over the paths and for tracking how different tests and inputs exercises the code.

The execution counts are cumulative. If the example program were executed again without removing the `.gcda` file, the count for the number of times each line in the source was executed would be added to the results of the previous run(s). This is potentially useful in several ways. For example, it could be used to accumulate data over a number of program runs as part of a test verification suite, or to provide more accurate long-term information over a large number of program runs.

The data in the `.gcda` files is saved immediately before the program exits. For each source file compiled with `-fprofile-arcs`, the profiling code first attempts to read in an existing `.gcda` file; if the file doesn't match the executable (differing number of basic block counts) it will ignore the contents of the file. It then adds in the new execution counts and finally writes the data to the file.

You can report on a subset of functions by using `--include` and `--exclude`. This is very useful when combined with `--stdout` trying to understand behavior and coverage for a particular function by running a test, looking at gcov output, testing another input, and running gcov again.

```
$ gcov -m --stdout --include inc tmp
```

```

-:      0:Source:tmp.cpp
-:      0:Graph:tmp.gcno
-:      0:Data:tmp.gcda
-:      0:Runs:1
2*:      8: void inc () { b++; }
-----
Foo<char>::inc():
#####:      8: void inc () { b++; }
-----
Foo<int>::inc():
2:      8: void inc () { b++; }
-----

```

gcov will match on mangled names by default, which you can control with the `-M` flag. Note that matching and reporting are independent, so you can match on mangled names while printing demangled names, and vice versa. To report on the `int` instantiation of `Foo` matching on mangled and demangled names:

```

$ gcov -t -m -M tmp --include 'Foo<int>'
-:      0:Source:tmp.cpp
-:      0:Graph:tmp.gcno
-:      0:Data:tmp.gcda
-:      0:Runs:1
1:      7: Foo(): b (1000) {}
2:      8: void inc () { b++; }

$ gcov -t -m tmp --include 'FooIi'
-:      0:Source:tmp.cpp
-:      0:Graph:tmp.gcno
-:      0:Data:tmp.gcda
-:      0:Runs:1
1:      7: Foo(): b (1000) {}
2:      8: void inc () { b++; }

```

The arguments to `--include` and `--exclude` are extended regular expressions (like `grep -E`), so the pattern `in.?` matches both `inc` and `main`. If used with `-M` then all `int` instantiations of `Foo` would match too. `--include` and `--exclude` can be used multiple times, and if a name matches multiple filters it is the last one to match which takes preference. For example, to match `main` and the `int` instantiation of `inc`, while omitting the `Foo` constructor:

```

$ gcov -t -m -M --include in --exclude Foo --include '<int>::inc' tmp
-:      0:Source:tmp.cpp
-:      0:Graph:tmp.gcno
-:      0:Data:tmp.gcda
-:      0:Runs:1
2:      8: void inc () { b++; }
1:     18:main (void)
-:     19:{
-:     20: int i, total;
1:     21: Foo<int> counter;
-:     22:
1:     23: counter.inc();
1:     24: counter.inc();
1:     25: total = 0;
-:     26:
11:    27: for (i = 0; i < 10; i++)
10:    28:     total += i;
-:     29:
1*:    30: int v = total > 100 ? 1 : 2;
-:    31:

```

```

1: 32: if (total != 45)
####: 33:     printf ("Failure\n");
-: 34: else
1: 35:     printf ("Success\n");
1: 36: return 0;

```

11.3 Using gcov with GCC Optimization

If you plan to use `gcov` to help optimize your code, you must first compile your program with a special GCC option ‘`--coverage`’. Aside from that, you can use any other GCC options; but if you want to prove that every single line in your program was executed, you should not compile with optimization at the same time. On some machines the optimizer can eliminate some simple code lines by combining them with other lines. For example, code like this:

```

if (a != b)
    c = 1;
else
    c = 0;

```

can be compiled into one instruction on some machines. In this case, there is no way for `gcov` to calculate separate execution counts for each line because there isn’t separate code for each line. Hence the `gcov` output looks like this if you compiled the program with optimization:

```

100: 12:if (a != b)
100: 13:  c = 1;
100: 14:else
100: 15:  c = 0;

```

The output shows that this block of code, combined by optimization, executed 100 times. In one sense this result is correct, because there was only one instruction representing all four of these lines. However, the output does not indicate how many times the result was 0 and how many times the result was 1.

Inlineable functions can create unexpected line counts. Line counts are shown for the source code of the inlineable function, but what is shown depends on where the function is inlined, or if it is not inlined at all.

If the function is not inlined, the compiler must emit an out of line copy of the function, in any object file that needs it. If `fileA.o` and `fileB.o` both contain out of line bodies of a particular inlineable function, they will also both contain coverage counts for that function. When `fileA.o` and `fileB.o` are linked together, the linker will, on many systems, select one of those out of line bodies for all calls to that function, and remove or ignore the other. Unfortunately, it will not remove the coverage counters for the unused function body. Hence when instrumented, all but one use of that function will show zero counts.

If the function is inlined in several places, the block structure in each location might not be the same. For instance, a condition might now be calculable at compile time in some instances. Because the coverage of all the uses of the inline function will be shown for the same source lines, the line counts themselves might seem inconsistent.

Long-running applications can use the `__gcov_reset` and `__gcov_dump` facilities to restrict profile collection to the program region of interest. Calling `__gcov_reset(void)` will clear all run-time profile counters to zero, and calling `__gcov_dump(void)` will cause the profile information collected at that point to be dumped to `.gcda` output files. Instrumented

applications use a static destructor with priority 99 to invoke the `__gcov_dump` function. Thus `__gcov_dump` is executed after all user defined static destructors, as well as handlers registered with `atexit`.

If an executable loads a dynamic shared object via `dlopen` functionality, `-Wl,--dynamic-list-data` is needed to dump all profile data.

Profiling run-time library reports various errors related to profile manipulation and profile saving. Errors are printed into standard error output or ‘`GCOV_ERROR_FILE`’ file, if environment variable is used. In order to terminate immediately after an errors occurs set ‘`GCOV_EXIT_AT_ERROR`’ environment variable. That can help users to find profile clashing which leads to a misleading profile.

11.4 Brief Description of gcov Data Files

`gcov` uses two files for profiling. The names of these files are derived from the original *object* file by substituting the file suffix with either `.gcno`, or `.gcda`. The files contain coverage and profile data stored in a platform-independent format. The `.gcno` files are placed in the same directory as the object file. By default, the `.gcda` files are also stored in the same directory as the object file, but the GCC `-fprofile-dir` option may be used to store the `.gcda` files in a separate directory.

The `.gcno` notes file is generated when the source file is compiled with the GCC `-ftest-coverage` option. It contains information to reconstruct the basic block graphs and assign source line numbers to blocks.

The `.gcda` count data file is generated when a program containing object files built with the GCC `-fprofile-arcs` option is executed. A separate `.gcda` file is created for each object file compiled with this option. It contains arc transition counts, value profile counts, and some summary information.

It is not recommended to access the coverage files directly. Consumers should use the intermediate format that is provided by `gcov` tool via `--json-format` option.

11.5 Data File Relocation to Support Cross-Profiling

Running the program will cause profile output to be generated. For each source file compiled with `-fprofile-arcs`, an accompanying `.gcda` file will be placed in the object file directory. That implicitly requires running the program on the same system as it was built or having the same absolute directory structure on the target system. The program will try to create the needed directory structure, if it is not already present.

To support cross-profiling, a program compiled with `-fprofile-arcs` can relocate the data files based on two environment variables:

- `GCOV_PREFIX` contains the prefix to add to the absolute paths in the object file. Prefix can be absolute, or relative. The default is no prefix.
- `GCOV_PREFIX_STRIP` indicates the how many initial directory names to strip off the hardwired absolute paths. Default value is 0.

Note: If `GCOV_PREFIX_STRIP` is set without `GCOV_PREFIX` is undefined, then a relative path is made out of the hardwired absolute paths.

For example, if the object file `/user/build/foo.o` was built with `-fprofile-arcs`, the final executable will try to create the data file `/user/build/foo.gcda` when running on

the target system. This will fail if the corresponding directory does not exist and it is unable to create it. This can be overcome by, for example, setting the environment as 'GCOV_PREFIX=/target/run' and 'GCOV_PREFIX_STRIP=1'. Such a setting will name the data file /target/run/build/foo.gcda.

You must move the data files to the expected directory tree in order to use them for profile directed optimizations (`-fprofile-use`), or to use the `gcov` tool.

11.6 Profiling and Test Coverage in Freestanding Environments

In case your application runs in a hosted environment such as GNU/Linux, then this section is likely not relevant to you. This section is intended for application developers targeting freestanding environments (for example embedded systems) with limited resources. In particular, systems or test cases which do not support constructors/destructors or the C library file I/O. In this section, the *target system* runs your application instrumented for profiling or test coverage. You develop and analyze your application on the *host system*. We now provide an overview how profiling and test coverage can be obtained in this scenario followed by a tutorial which can be exercised on the host system. Finally, some system initialization caveats are listed.

11.6.1 Overview

For an application instrumented for profiling or test coverage, the compiler generates some global data structures which are updated by instrumentation code while the application runs. These data structures are called the *gcov information*. Normally, when the application exits, the gcov information is stored to `.gcda` files. There is one file per translation unit instrumented for profiling or test coverage. The function `__gcov_exit()`, which stores the gcov information to a file, is called by a global destructor function for each translation unit instrumented for profiling or test coverage. It runs at process exit. In a global constructor function, the `__gcov_init()` function is called to register the gcov information of a translation unit in a global list. In some situations, this procedure does not work. Firstly, if you want to profile the global constructor or exit processing of an operating system, the compiler generated functions may conflict with the test objectives. Secondly, you may want to test early parts of the system initialization or abnormal program behaviour which do not allow a global constructor or exit processing. Thirdly, you need a filesystem to store the files.

The `-fprofile-info-section` GCC option enables you to use profiling and test coverage in freestanding environments. This option disables the use of global constructors and destructors for the gcov information. Instead, a pointer to the gcov information is stored in a special linker input section for each translation unit which is compiled with this option. By default, the section name is `.gcov_info`. The gcov information is statically initialized. The pointers to the gcov information from all translation units of an executable can be collected by the linker in a contiguous memory block. For the GNU linker, the below linker script output section definition can be used to achieve this:

```
.gcov_info      :
{
    PROVIDE (__gcov_info_start = .);
    KEEP (*( .gcov_info))
}
```

```
    PROVIDE (__gcov_info_end = .);
}
```

The linker will provide two global symbols, `__gcov_info_start` and `__gcov_info_end`, which define the start and end of the array of pointers to gcov information blocks, respectively. The `KEEP ()` directive is required to prevent a garbage collection of the pointers. They are not directly referenced by anything in the executable. The section may be placed in a read-only memory area.

In order to transfer the profiling and test coverage data from the target to the host system, the application has to provide a function to produce a reliable in order byte stream from the target to the host. The byte stream may be compressed and encoded using error detection and correction codes to meet application-specific requirements. The GCC provided `libgcov` target library provides two functions, `__gcov_info_to_gcda()` and `__gcov_filename_to_gcfn()`, to generate a byte stream from a gcov information block. The functions are declared in `#include <gcov.h>`. The byte stream can be deserialized by the `merge-stream` subcommand of the `gcov-tool` to create or update `.gcda` files in the host filesystem for the instrumented application.

11.6.2 Tutorial

This tutorial should be exercised on the host system. We will build a program instrumented for test coverage. The program runs an application and dumps the gcov information to `stderr` encoded as a printable character stream. The application simply decodes such character streams from `stdin` and writes the decoded character stream to `stdout` (warning: this is binary data). The decoded character stream is consumed by the `merge-stream` subcommand of the `gcov-tool` to create or update the `.gcda` files.

To get started, create an empty directory. Change into the new directory. Then you will create the following three files in this directory

1. `app.h` - a header file included by `app.c` and `main.c`,
2. `app.c` - a source file which contains an example application, and
3. `main.c` - a source file which contains the program main function and code to dump the gcov information.

Firstly, create the header file `app.h` with the following content:

```
static const unsigned char a = 'a';

static inline unsigned char *
encode (unsigned char c, unsigned char buf[2])
{
    buf[0] = c % 16 + a;
    buf[1] = (c / 16) % 16 + a;
    return buf;
}

extern void application (void);
```

Secondly, create the source file `app.c` with the following content:

```
#include "app.h"

#include <stdio.h>

/* The application reads a character stream encoded by encode() from stdin,
```

```

    decodes it, and writes the decoded characters to stdout.  Characters other
    than the 16 characters 'a' to 'p' are ignored.  */

static int can_decode (unsigned char c)
{
    return (unsigned char)(c - a) < 16;
}

void
application (void)
{
    int first = 1;
    int i;
    unsigned char c;

    while ((i = fgetc (stdin)) != EOF)
    {
        unsigned char x = (unsigned char)i;

        if (can_decode (x))
        {
            if (first)
                c = x - a;
            else
                fputc (c + 16 * (x - a), stdout);
            first = !first;
        }
        else
            first = 1;
    }
}

```

Thirdly, create the source file `main.c` with the following content:

```

#include "app.h"

#include <gcov.h>
#include <stdio.h>
#include <stdlib.h>

/* The start and end symbols are provided by the linker script.  We use the
   array notation to avoid issues with a potential small-data area.  */

extern const struct gcov_info *const __gcov_info_start[];
extern const struct gcov_info *const __gcov_info_end[];

/* This function shall produce a reliable in order byte stream to transfer the
   gcov information from the target to the host system.  */

static void
dump (const void *d, unsigned n, void *arg)
{
    (void)arg;
    const unsigned char *c = d;
    unsigned char buf[2];

    for (unsigned i = 0; i < n; ++i)
        fwrite (encode (c[i], buf), sizeof (buf), 1, stderr);
}

```

```

/* The filename is serialized to a gcfn data stream by the
   __gcov_filename_to_gcfn() function. The gcfn data is used by the
   "merge-stream" subcommand of the "gcov-tool" to figure out the filename
   associated with the gcov information. */

static void
filename (const char *f, void *arg)
{
    __gcov_filename_to_gcfn (f, dump, arg);
}

/* The __gcov_info_to_gcda() function may have to allocate memory under
   certain conditions. Simply try it out if it is needed for your application
   or not. */

static void *
allocate (unsigned length, void *arg)
{
    (void)arg;
    return malloc (length);
}

/* Dump the gcov information of all translation units. */

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);
        fputc ('\n', stderr);
        ++info;
    }
}

/* The main() function just runs the application and then dumps the gcov
   information to stderr. */

int
main (void)
{
    application ();
    dump_gcov_info ();
    return 0;
}

```

If we compile `app.c` with test coverage and no extra profiling options, then a global constructor (`_sub_I_00100_0` here, it may have a different name in your environment) and destructor (`_sub_D_00100_1`) is used to register and dump the gcov information, respectively. We also see undefined references to `__gcov_init` and `__gcov_exit`:

```
$ gcc --coverage -c app.c
```

```
$ nm app.o
0000000000000000 r a
0000000000000030 T application
0000000000000000 t can_decode
                U fgetc
                U fputc
0000000000000000 b __gcov0.application
0000000000000038 b __gcov0.can_decode
0000000000000000 d __gcov_.application
00000000000000c0 d __gcov_.can_decode
                U __gcov_exit
                U __gcov_init
                U __gcov_merge_add
                U stdin
                U stdout
0000000000000161 t _sub_D_00100_1
0000000000000151 t _sub_I_00100_0
```

Compile `app.c` and `main.c` with test coverage and `-fprofile-info-section`. Now, a read-only pointer size object is present in the `.gcov_info` section and there are no undefined references to `__gcov_init` and `__gcov_exit`:

```
$ gcc --coverage -fprofile-info-section -c main.c
$ gcc --coverage -fprofile-info-section -c app.c
$ objdump -h app.o
```

```
app.o:      file format elf64-x86-64
```

Sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	00000151	0000000000000000	0000000000000000	00000040	2**0
	CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE					
1	.data	00000100	0000000000000000	0000000000000000	000001a0	2**5
	CONTENTS, ALLOC, LOAD, RELOC, DATA					
2	.bss	00000040	0000000000000000	0000000000000000	000002a0	2**5
	ALLOC					
3	.rodata	0000003c	0000000000000000	0000000000000000	000002a0	2**3
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
4	.gcov_info	00000008	0000000000000000	0000000000000000	000002e0	2**3
	CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA					
5	.comment	0000004e	0000000000000000	0000000000000000	000002e8	2**0
	CONTENTS, READONLY					
6	.note.GNU-stack	00000000	0000000000000000	0000000000000000	00000336	2**0
	CONTENTS, READONLY					
7	.eh_frame	00000058	0000000000000000	0000000000000000	00000338	2**3
	CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA					

We have to customize the program link procedure so that all the `.gcov_info` linker input sections are placed in a contiguous memory block with a begin and end symbol. Firstly, get the default linker script using the following commands (we assume a GNU linker):

```
$ ld --verbose | sed '1,/^===/d' | sed '/^===/d' > linkcmds
```

Secondly, open the file `linkcmds` with a text editor and place the linker output section definition from the overview after the `.rodata` section definition. Link the program executable using the customized linker script:

```
$ gcc --coverage main.o app.o -T linkcmds -Wl,-Map,app.map
```

In the linker map file `app.map`, we see that the linker placed the read-only pointer size objects of our objects files `main.o` and `app.o` into a contiguous memory block and provided the symbols `__gcov_info_start` and `__gcov_info_end`:

```
$ grep -C 1 "\.gcov_info" app.map

.gcov_info      0x0000000000403ac0      0x10
                0x0000000000403ac0      PROVIDE (__gcov_info_start = .)
*(.gcov_info)
.gcov_info      0x0000000000403ac0      0x8 main.o
.gcov_info      0x0000000000403ac8      0x8 app.o
                0x0000000000403ad0      PROVIDE (__gcov_info_end = .)
```

Make sure no `.gcda` files are present. Run the program with nothing to decode and dump `stderr` to the file `gcda-0.txt` (first run). Run the program to decode `gcda-0.txt` and send it to the `gcov-tool` using the `merge-stream` subcommand to create the `.gcda` files (second run). Run `gcov` to produce a report for `app.c`. We see that the first run with nothing to decode results in a partially covered application:

```
$ rm -f app.gcda main.gcda
$ echo "" | ./a.out 2>gcda-0.txt
$ ./a.out <gcda-0.txt 2>gcda-1.txt | gcov-tool merge-stream
$ gcov -bc app.c
File 'app.c'
Lines executed:69.23% of 13
Branches executed:66.67% of 6
Taken at least once:50.00% of 6
Calls executed:66.67% of 3
Creating 'app.c.gcov'

Lines executed:69.23% of 13
```

Run the program to decode `gcda-1.txt` and send it to the `gcov-tool` using the `merge-stream` subcommand to update the `.gcda` files. Run `gcov` to produce a report for `app.c`. Since the second run decoded the gcov information of the first run, we have now a fully covered application:

```
$ ./a.out <gcda-1.txt 2>gcda-2.txt | gcov-tool merge-stream
$ gcov -bc app.c
File 'app.c'
Lines executed:100.00% of 13
Branches executed:100.00% of 6
Taken at least once:100.00% of 6
Calls executed:100.00% of 3
Creating 'app.c.gcov'

Lines executed:100.00% of 13
```

11.6.3 System Initialization Caveats

The gcov information of a translation unit consists of several global data structures. For example, the instrumented code may update program flow graph edge counters in a zero-initialized data structure. It is safe to run instrumented code before the zero-initialized data is cleared to zero. The coverage information obtained before the zero-initialized data is cleared to zero is unusable. Dumping the gcov information using `__gcov_info_to_gcda()` before the zero-initialized data is cleared to zero or the initialized data is loaded, is undefined behaviour. Clearing the zero-initialized data to zero through a function instrumented for

profiling or test coverage is undefined behaviour, since it may produce inconsistent program flow graph edge counters for example.

12 gcov-tool—an Offline Gcda Profile Processing Tool

`gcov-tool` is a tool you can use in conjunction with GCC to manipulate or process gcda profile files offline.

12.1 Introduction to gcov-tool

`gcov-tool` is an offline tool to process gcc's gcda profile files.

Current `gcov-tool` supports the following functionalities:

- merge two sets of profiles with weights.
- read a stream of profiles with associated filenames and merge it with a set of profiles with weights.
- read one set of profile and rewrite profile contents. One can scale or normalize the count values.

Examples of the use cases for this tool are:

- Collect the profiles for different set of inputs, and use this tool to merge them. One can specify the weight to factor in the relative importance of each input.
- Collect profiles from target systems without a filesystem (freestanding environments). Merge the collected profiles with associated profiles present on the host system. One can specify the weight to factor in the relative importance of each input.
- Rewrite the profile after removing a subset of the gcda files, while maintaining the consistency of the summary and the histogram.
- It can also be used to debug or libgcov code as the tools shares the majority code as the runtime library.

Note that for the merging operation, this profile generated offline may contain slight different values from the online merged profile. Here are a list of typical differences:

- histogram difference: This offline tool recomputes the histogram after merging the counters. The resulting histogram, therefore, is precise. The online merging does not have this capability – the histogram is merged from two histograms and the result is an approximation.
- summary checksum difference: Summary checksum uses a CRC32 operation. The value depends on the link list order of gcov-info objects. This order is different in `gcov-tool` from that in the online merge. It's expected to have different summary checksums. It does not really matter as the compiler does not use this checksum anywhere.
- value profile counter values difference: Some counter values for value profile are runtime dependent, like heap addresses. It's normal to see some difference in these kind of counters.

12.2 Invoking gcov-tool

```
gcov-tool [global-options] SUB_COMMAND [sub_command-options] profile_dir
```

`gcov-tool` accepts the following options:

- h
- help Display help about using `gcov-tool` (on the standard output), and exit without doing any further processing.

```

-v
--version
    Display the gcov-tool version number (on the standard output), and exit
    without doing any further processing.

merge
    Merge two profile directories.

    -o directory
    --output directory
        Set the output profile directory. Default output directory name is
        merged_profile.

    -v
    --verbose
        Set the verbose mode.

    -w w1,w2
    --weight w1,w2
        Set the merge weights of the directory1 and directory2, respectively.
        The default weights are 1 for both.

merge-stream
    Collect profiles with associated filenames from a gcfn and gcda data stream.
    Read the stream from the file specified by file or from stdin. Merge the profiles
    with associated profiles in the host filesystem. Apply the optional weights while
    merging profiles.

    For the generation of a gcfn and gcda data stream on the target system, please
    have a look at the __gcov_filename_to_gcfn() and __gcov_info_to_gcda()
    functions declared in #include <gcov.h>.

    -v
    --verbose
        Set the verbose mode.

    -w w1,w2
    --weight w1,w2
        Set the merge weights of the profiles from the gcfn and gcda data
        stream and the associated profiles in the host filesystem, respec-
        tively. The default weights are 1 for both.

rewrite
    Read the specified profile directory and rewrite to a new directory.

    -n long_long_value
    --normalize <long_long_value>
        Normalize the profile. The specified value is the max counter value
        in the new profile.

    -o directory
    --output directory
        Set the output profile directory. Default output name is
        rewrite_profile.

```

```

-s float_or_simple-frac_value
--scale float_or_simple-frac_value
    Scale the profile counters. The specified value can be in floating
    point value, or simple fraction value form, such 1, 2, 2/3, and 5/3.

-v
--verbose
    Set the verbose mode.

overlap Compute the overlap score between the two specified profile directories. The
overlap score is computed based on the arc profiles. It is defined as the sum
of min (p1_counter[i] / p1_sum_all, p2_counter[i] / p2_sum_all), for all arc
counter i, where p1_counter[i] and p2_counter[i] are two matched counters and
p1_sum_all and p2_sum_all are the sum of counter values in profile 1 and profile
2, respectively.

-f
--function
    Print function level overlap score.

-F
--fullname
    Print full gcda filename.

-h
--hotonly
    Only print info for hot objects/functions.

-o
--object Print object level overlap score.

-t float
--hot_threshold <float>
    Set the threshold for hot counter value.

-v
--verbose
    Set the verbose mode.

```


13 gcov-dump—an Offline Gcda and Gcno Profile Dump Tool

13.1 Introduction to gcov-dump

gcov-dump is a tool you can use in conjunction with GCC to dump content of gcda and gcno profile files offline.

13.2 Invoking gcov-dump

Usage: **gcov-dump** [*OPTION*] ... *gcovfiles*

gcov-dump accepts the following options:

- h
- help Display help about using **gcov-dump** (on the standard output), and exit without doing any further processing.
- l
- long Dump content of records.
- p
- positions Dump positions of records.
- r
- raw Print content records in raw format.
- s
- stable Print content in stable format usable for comparison.
- v
- version Display the **gcov-dump** version number (on the standard output), and exit without doing any further processing.

14 lto-dump—Tool for dumping LTO object files.

14.1 Introduction to lto-dump

`lto-dump` is a tool you can use in conjunction with GCC to dump link time optimization object files.

14.2 Invoking lto-dump

Usage: `lto-dump [OPTION] ... objfiles`

`lto-dump` accepts the following options:

- `-list` Dumps list of details of functions and variables.
- `-demangle` Dump the demangled output.
- `-defined-only` Dump only the defined symbols.
- `-print-value` Dump initial values of the variables.
- `-name-sort` Sort the symbols alphabetically.
- `-size-sort` Sort the symbols according to size.
- `-reverse-sort` Dump the symbols in reverse order.
- `-no-sort` Dump the symbols in order of occurrence.
- `-symbol=` Dump the details of specific symbol.
- `-objects` Dump the details of LTO objects.
- `-type-stats` Dump the statistics of tree types.
- `-tree-stats` Dump the statistics of trees.
- `-gimple-stats` Dump the statistics of gimple statements.
- `-dump-level=` For deciding the optimization level of body.
- `-dump-body=` Dump the specific gimple body.
- `-help` Display the dump tool help.

15 Known Causes of Trouble with GCC

This section describes known problems that affect users of GCC. Most of these are not GCC bugs per se—if they were, we would fix them. But the result for a user may be like the result of a bug.

Some of these problems are due to bugs in other software, some are missing features that are too much work to add, and some are places where people’s opinions differ as to what is best.

15.1 Actual Bugs We Haven’t Fixed Yet

- The `fixincludes` script interacts badly with automounters; if the directory of system header files is automounted, it tends to be unmounted while `fixincludes` is running. This would seem to be a bug in the automounter. We don’t know any good way to work around it.

15.2 Interoperation

This section lists various difficulties encountered in using GCC together with other compilers or with the assemblers, linkers, libraries and debuggers on certain systems.

- On many platforms, GCC supports a different ABI for C++ than do other compilers, so the object files compiled by GCC cannot be used with object files generated by another C++ compiler.

An area where the difference is most apparent is name mangling. The use of different name mangling is intentional, to protect you from more subtle problems. Compilers differ as to many internal details of C++ implementation, including: how class instances are laid out, how multiple inheritance is implemented, and how virtual function calls are handled. If the name encoding were made the same, your programs would link against libraries provided from other compilers—but the programs would then crash when run. Incompatible libraries are then detected at link time, rather than at run time.

- On some BSD systems, including some versions of Ultrix, use of profiling causes static variable destructors (currently used only in C++) not to be run.
- On a SPARC, GCC aligns all values of type `double` on an 8-byte boundary, and it expects every `double` to be so aligned. The Sun compiler usually gives `double` values 8-byte alignment, with one exception: function arguments of type `double` may not be aligned.

As a result, if a function compiled with Sun CC takes the address of an argument of type `double` and passes this pointer of type `double *` to a function compiled with GCC, dereferencing the pointer may cause a fatal signal.

One way to solve this problem is to compile your entire program with GCC. Another solution is to modify the function that is compiled with Sun CC to copy the argument into a local variable; local variables are always properly aligned. A third solution is to modify the function that uses the pointer to dereference it via the following function `access_double` instead of directly with `*`:

```
inline double
```

```

access_double (double *unaligned_ptr)
{
    union d2i { double d; int i[2]; };

    union d2i *p = (union d2i *) unaligned_ptr;
    union d2i u;

    u.i[0] = p->i[0];
    u.i[1] = p->i[1];

    return u.d;
}

```

Storing into the pointer can be done likewise with the same union.

- On Solaris, the `malloc` function in the `libmalloc.a` library may allocate memory that is only 4 byte aligned. Since GCC on the SPARC assumes that doubles are 8 byte aligned, this may result in a fatal signal if doubles are stored in memory allocated by the `libmalloc.a` library.

The solution is to not use the `libmalloc.a` library. Use instead `malloc` and related functions from `libc.a`; they do not have this problem.

- On the HP PA machine, ADB sometimes fails to work on functions compiled with GCC. Specifically, it fails to work on functions that use `alloca` or variable-size arrays. This is because GCC doesn't generate HP-UX unwind descriptors for such functions. It may even be impossible to generate them.
- Debugging (`-g`) is not supported on the HP PA machine, unless you use the preliminary GNU tools.
- Taking the address of a label may generate errors from the HP-UX PA assembler. GAS for the PA does not have this problem.
- Using floating point parameters for indirect calls to static functions will not work when using the HP assembler. There simply is no way for GCC to specify what registers hold arguments for static functions when using the HP assembler. GAS for the PA does not have this problem.
- In extremely rare cases involving some very large functions you may receive errors from the HP linker complaining about an out of bounds unconditional branch offset. This used to occur more often in previous versions of GCC, but is now exceptionally rare. If you should run into it, you can work around by making your function smaller.
- GCC compiled code sometimes emits warnings from the HP-UX assembler of the form:

```

(warning) Use of GR3 when
         frame >= 8192 may cause conflict.

```

These warnings are harmless and can be safely ignored.

- In extremely rare cases involving some very large functions you may receive errors from the AIX Assembler complaining about a displacement that is too large. If you should run into it, you can work around by making your function smaller.
- The `libstdc++.a` library in GCC relies on the SVR4 dynamic linker semantics which merges global symbols between libraries and applications, especially necessary for C++ streams functionality. This is not the default behavior of AIX shared libraries and dynamic linking. `libstdc++.a` is built on AIX with "runtime-linking" enabled so that symbol merging can occur. To utilize this feature, the application linked with

`libstdc++.a` must include the `-Wl,-brtl` flag on the link line. G++ cannot impose this because this option may interfere with the semantics of the user program and users may not always use ‘g++’ to link his or her application. Applications are not required to use the `-Wl,-brtl` flag on the link line—the rest of the `libstdc++.a` library which is not dependent on the symbol merging semantics will continue to function correctly.

- An application can interpose its own definition of functions for functions invoked by `libstdc++.a` with “runtime-linking” enabled on AIX. To accomplish this the application must be linked with “runtime-linking” option and the functions explicitly must be exported by the application (`-Wl,-brtl,-bE:exportfile`).
- AIX on the RS/6000 provides support (NLS) for environments outside of the United States. Compilers and assemblers use NLS to support locale-specific representations of various objects including floating-point numbers (‘.’ vs ‘,’ for separating decimal fractions). There have been problems reported where the library linked with GCC does not produce the same floating-point formats that the assembler accepts. If you have this problem, set the `LANG` environment variable to ‘C’ or ‘En_US’.
- Even if you specify `-fdollars-in-identifiers`, you cannot successfully use ‘\$’ in identifiers on the RS/6000 due to a restriction in the IBM assembler. GAS supports these identifiers.

15.3 Incompatibilities of GCC

There are several noteworthy incompatibilities between GNU C and K&R (non-ISO) versions of C.

- GCC normally makes string constants read-only. If several identical-looking string constants are used, GCC stores only one copy of the string.

One consequence is that you cannot call `mktemp` with a string constant argument. The function `mktemp` always alters the string its argument points to.

Another consequence is that `sscanf` does not work on some very old systems when passed a string constant as its format control string or input. This is because `sscanf` incorrectly tries to write into the string constant. Likewise `fscanf` and `scanf`.

The solution to these problems is to change the program to use `char`-array variables with initialization strings for these purposes instead of string constants.

- `-2147483648` is positive.

This is because `2147483648` cannot fit in the type `int`, so (following the ISO C rules) its data type is `unsigned long int`. Negating this value yields `2147483648` again.

- GCC does not substitute macro arguments when they appear inside of string constants. For example, the following macro in GCC

```
#define foo(a) "a"
```

will produce output “a” regardless of what the argument `a` is.

- When you use `setjmp` and `longjmp`, the only automatic variables guaranteed to remain valid are those declared `volatile`. This is a consequence of automatic register allocation. Consider this function:

```
jmp_buf j;
```

```
foo ()
```

```

{
    int a, b;

    a = fun1 ();
    if (setjmp (j))
        return a;

    a = fun2 ();
    /* longjmp (j) may occur in fun3. */
    return a + fun3 ();
}

```

Here `a` may or may not be restored to its first value when the `longjmp` occurs. If `a` is allocated in a register, then its first value is restored; otherwise, it keeps the last value stored in it.

If you use the `-W` option with the `-O` option, you will get a warning when GCC thinks such a problem might be possible.

- Programs that use preprocessing directives in the middle of macro arguments do not work with GCC. For example, a program like this will not work:

```

foobar (
    #define luser
    hack)

```

ISO C does not permit such a construct.

- K&R compilers allow comments to cross over an inclusion boundary (i.e. started in an include file and ended in the including file).
- Declarations of external variables and functions within a block apply only to the block containing the declaration. In other words, they have the same scope as any other declaration in the same place.

In some other C compilers, an `extern` declaration affects all the rest of the file even if it happens within a block.

- In traditional C, you can combine `long`, etc., with a typedef name, as shown here:

```

typedef int foo;
typedef long foo bar;

```

In ISO C, this is not allowed: `long` and other type modifiers require an explicit `int`.

- PCC allows typedef names to be used as function parameters.
- Traditional C allows the following erroneous pair of declarations to appear together in a given scope:

```

typedef int foo;
typedef foo foo;

```

- GCC treats all characters of identifiers as significant. According to K&R-1 (2.2), “No more than the first eight characters are significant, although more may be used.”. Also according to K&R-1 (2.2), “An identifier is a sequence of letters and digits; the first character must be a letter. The underscore `_` counts as a letter.”, but GCC also allows dollar signs in identifiers.
- PCC allows whitespace in the middle of compound assignment operators such as `‘+=’`. GCC, following the ISO standard, does not allow this.
- GCC complains about unterminated character constants inside of preprocessing conditionals that fail. Some programs have English comments enclosed in conditionals

that are guaranteed to fail; if these comments contain apostrophes, GCC will probably report an error. For example, this code would produce an error:

```
#if 0
You can't expect this to work.
#endif
```

The best solution to such a problem is to put the text into an actual C comment delimited by `/*...*/`.

- Many user programs contain the declaration `'long time ();'`. In the past, the system header files on many systems did not actually declare `time`, so it did not matter what type your program declared it to return. But in systems with ISO C headers, `time` is declared to return `time_t`, and if that is not the same as `long`, then `'long time ();'` is erroneous.

The solution is to change your program to use appropriate system headers (`<time.h>` on systems with ISO C headers) and not to declare `time` if the system header files declare it, or failing that to use `time_t` as the return type of `time`.

- When compiling functions that return `float`, PCC converts it to a double. GCC actually returns a `float`. If you are concerned with PCC compatibility, you should declare your functions to return `double`; you might as well say what you mean.
- When compiling functions that return structures or unions, GCC output code normally uses a method different from that used on most versions of Unix. As a result, code compiled with GCC cannot call a structure-returning function compiled with PCC, and vice versa.

The method used by GCC is as follows: a structure or union which is 1, 2, 4 or 8 bytes long is returned like a scalar. A structure or union with any other size is stored into an address supplied by the caller (usually in a special, fixed register, but on some machines it is passed on the stack). The target hook `TARGET_STRUCT_VALUE_RTX` tells GCC where to pass this address.

By contrast, PCC on most target machines returns structures and unions of any size by copying the data into an area of static storage, and then returning the address of that storage as if it were a pointer value. The caller must copy the data from that memory area to the place where the value is wanted. GCC does not use this method because it is slower and nonreentrant.

On some newer machines, PCC uses a reentrant convention for all structure and union returning. GCC on most of these machines uses a compatible convention when returning structures and unions in memory, but still returns small structures and unions in registers.

You can tell GCC to use a compatible convention for all structure and union returning with the option `-fpcc-struct-return`.

- GCC complains about program fragments such as `'0x74ae-0x4000'` which appear to be two hexadecimal constants separated by the minus operator. Actually, this string is a single *preprocessing token*. Each such token must correspond to one token in C. Since this does not, GCC prints an error message. Although it may appear obvious that what is meant is an operator and two values, the ISO C standard specifically requires that this be treated as erroneous.

A *preprocessing token* is a *preprocessing number* if it begins with a digit and is followed by letters, underscores, digits, periods and ‘e+’, ‘e-’, ‘E+’, ‘E-’, ‘p+’, ‘p-’, ‘P+’, or ‘P-’ character sequences. (In strict C90 mode, the sequences ‘p+’, ‘p-’, ‘P+’ and ‘P-’ cannot appear in preprocessing numbers.)

To make the above program fragment valid, place whitespace in front of the minus sign. This whitespace will end the preprocessing number.

15.4 Fixed Header Files

GCC needs to install corrected versions of some system header files. This is because most target systems have some header files that won’t work with GCC unless they are changed. Some have bugs, some are incompatible with ISO C, and some depend on special features of other compilers.

Installing GCC automatically creates and installs the fixed header files, by running a program called `fixincludes`. Normally, you don’t need to pay attention to this. But there are cases where it doesn’t do the right thing automatically.

- If you update the system’s header files, such as by installing a new system version, the fixed header files of GCC are not automatically updated. They can be updated using the `mkheaders` script installed in `libexecdir/gcc/target/version/install-tools/`.
- On some systems, header file directories contain machine-specific symbolic links in certain places. This makes it possible to share most of the header files among hosts running the same version of the system on different machine models.

The programs that fix the header files do not understand this special way of using symbolic links; therefore, the directory of fixed header files is good only for the machine model used to build it.

It is possible to make separate sets of fixed header files for the different machine models, and arrange a structure of symbolic links so as to use the proper set, but you’ll have to do this by hand.

15.5 Standard Libraries

GCC by itself attempts to provide the compiler part of a conforming implementation, but only a limited subset of the library part of such an implementation. See Chapter 2 [Language Standards Supported by GCC], page 3, for details of what this means. Beyond the limited library facilities described there, the rest of the C library is supplied by the vendor of the operating system. If that C library doesn’t conform to the C standards, then your programs might get warnings (especially when using `-Wall`) that you don’t expect.

For example, the `sprintf` function on SunOS 4.1.3 returns `char *` while the C standard says that `sprintf` returns an `int`. The `fixincludes` program could make the prototype for this function match the Standard, but that would be wrong, since the function will still return `char *`.

If you need a Standard compliant library, then you need to find one, as GCC does not provide one. The GNU C library (called `glibc`) provides ISO C, POSIX, BSD, SystemV and X/Open compatibility for GNU/Linux and HURD-based GNU systems; no recent version of it supports other systems, though some very old versions did. Version 2.2 of the GNU

C library includes nearly complete C99 support. You could also ask your operating system vendor if newer libraries are available.

15.6 Disappointments and Misunderstandings

These problems are perhaps regrettable, but we don't know any practical way around them.

- Certain local variables aren't recognized by debuggers when you compile with optimization.

This occurs because sometimes GCC optimizes the variable out of existence. There is no way to tell the debugger how to compute the value such a variable “would have had”, and it is not clear that would be desirable anyway. So GCC simply does not mention the eliminated variable when it writes debugging information.

You have to expect a certain amount of disagreement between the executable and your source code, when you use optimization.

- Users often think it is a bug when GCC reports an error for code like this:

```
int foo (struct mumble *);

struct mumble { ... };

int foo (struct mumble *x)
{ ... }
```

This code really is erroneous, because the scope of `struct mumble` in the prototype is limited to the argument list containing it. It does not refer to the `struct mumble` defined with file scope immediately below—they are two unrelated types with similar names in different scopes.

But in the definition of `foo`, the file-scope type is used because that is available to be inherited. Thus, the definition and the prototype do not match, and you get an error.

This behavior may seem silly, but it's what the ISO standard specifies. It is easy enough for you to make your code work by moving the definition of `struct mumble` above the prototype. It's not worth being incompatible with ISO C just to avoid an error for the example shown above.

- Accesses to bit-fields even in volatile objects works by accessing larger objects, such as a byte or a word. You cannot rely on what size of object is accessed in order to read or write the bit-field; it may even vary for a given bit-field according to the precise usage. If you care about controlling the amount of memory that is accessed, use volatile but do not use bit-fields.
- GCC comes with shell scripts to fix certain known problems in system header files. They install corrected copies of various header files in a special directory where only GCC will normally look for them. The scripts adapt to various systems by searching all the system header files for the problem cases that we know about.

If new system header files are installed, nothing automatically arranges to update the corrected header files. They can be updated using the `mkheaders` script installed in `libexecdir/gcc/target/version/install-tools/`.

- On 68000 and x86 systems, for instance, you can get paradoxical results if you test the precise values of floating point numbers. For example, you can find that a floating point value which is not a NaN is not equal to itself. This results from the fact that

the floating point registers hold a few more bits of precision than fit in a `double` in memory. Compiled code moves values between memory and floating point registers at its convenience, and moving them into memory truncates them.

You can partially avoid this problem by using the `-ffloat-store` option (see Section 3.12 [Optimize Options], page 197).

- On AIX and other platforms without weak symbol support, templates need to be instantiated explicitly and symbols for static members of templates will not be generated.
- On AIX, GCC scans object files and library archives for static constructors and destructors when linking an application before the linker prunes unreferenced symbols. This is necessary to prevent the AIX linker from mistakenly assuming that static constructor or destructor are unused and removing them before the scanning can occur. All static constructors and destructors found will be referenced even though the modules in which they occur may not be used by the program. This may lead to both increased executable size and unexpected symbol references.

15.7 Common Misunderstandings with GNU C++

C++ is a complex language and an evolving one, and its standard definition (the ISO C++ standard) was only recently completed. As a result, your C++ compiler may occasionally surprise you, even when its behavior is correct. This section discusses some areas that frequently give rise to questions of this sort.

15.7.1 Declare *and* Define Static Members

When a class has static data members, it is not enough to *declare* the static member; you must also *define* it. For example:

```
class Foo
{
    ...
    void method();
    static int bar;
};
```

This declaration only establishes that the class `Foo` has an `int` named `Foo::bar`, and a member function named `Foo::method`. But you still need to define *both* `method` and `bar` elsewhere. According to the ISO standard, you must supply an initializer in one (and only one) source file, such as:

```
int Foo::bar = 0;
```

Other C++ compilers may not correctly implement the standard behavior. As a result, when you switch to `g++` from one of these compilers, you may discover that a program that appeared to work correctly in fact does not conform to the standard: `g++` reports as undefined symbols any static data members that lack definitions.

15.7.2 Name Lookup, Templates, and Accessing Members of Base Classes

The C++ standard prescribes that all names that are not dependent on template parameters are bound to their present definitions when parsing a template function or class.¹ Only names that are dependent are looked up at the point of instantiation. For example, consider

```
void foo(double);

struct A {
    template <typename T>
    void f () {
        foo (1);           // 1
        int i = N;         // 2
        T t;
        t.bar();           // 3
        foo (t);           // 4
    }

    static const int N;
};
```

Here, the names `foo` and `N` appear in a context that does not depend on the type of `T`. The compiler will thus require that they are defined in the context of use in the template, not only before the point of instantiation, and will here use `::foo(double)` and `A::N`, respectively. In particular, it will convert the integer value to a `double` when passing it to `::foo(double)`.

Conversely, `bar` and the call to `foo` in the fourth marked line are used in contexts that do depend on the type of `T`, so they are only looked up at the point of instantiation, and you can provide declarations for them after declaring the template, but before instantiating it. In particular, if you instantiate `A::f<int>`, the last line will call an overloaded `::foo(int)` if one was provided, even if after the declaration of `struct A`.

This distinction between lookup of dependent and non-dependent names is called two-stage (or dependent) name lookup. G++ implements it since version 3.4.

Two-stage name lookup sometimes leads to situations with behavior different from non-template codes. The most common is probably this:

```
template <typename T> struct Base {
    int i;
};

template <typename T> struct Derived : public Base<T> {
    int get_i() { return i; }
};
```

In `get_i()`, `i` is not used in a dependent context, so the compiler will look for a name declared at the enclosing namespace scope (which is the global scope here). It will not look into the base class, since that is dependent and you may declare specializations of `Base` even after declaring `Derived`, so the compiler cannot really know what `i` would refer to. If there is no global variable `i`, then you will get an error message.

In order to make it clear that you want the member of the base class, you need to defer lookup until instantiation time, at which the base class is known. For this, you need to

¹ The C++ standard just uses the term “dependent” for names that depend on the type or value of template parameters. This shorter term will also be used in the rest of this section.

access `i` in a dependent context, by either using `this->i` (remember that `this` is of type `Derived<T>*`, so is obviously dependent), or using `Base<T>::i`. Alternatively, `Base<T>::i` might be brought into scope by a `using`-declaration.

Another, similar example involves calling member functions of a base class:

```
template <typename T> struct Base {
    int f();
};

template <typename T> struct Derived : Base<T> {
    int g() { return f(); };
};
```

Again, the call to `f()` is not dependent on template arguments (there are no arguments that depend on the type `T`, and it is also not otherwise specified that the call should be in a dependent context). Thus a global declaration of such a function must be available, since the one in the base class is not visible until instantiation time. The compiler will consequently produce the following error message:

```
x.cc: In member function `int Derived<T>::g()':
x.cc:6: error: there are no arguments to `f' that depend on a template
parameter, so a declaration of `f' must be available
x.cc:6: error: (if you use `-fpermissive', G++ will accept your code, but
allowing the use of an undeclared name is deprecated)
```

To make the code valid either use `this->f()`, or `Base<T>::f()`. Using the `-fpermissive` flag will also let the compiler accept the code, by marking all function calls for which no declaration is visible at the time of definition of the template for later lookup at instantiation time, as if it were a dependent call. We do not recommend using `-fpermissive` to work around invalid code, and it will also only catch cases where functions in base classes are called, not where variables in base classes are used (as in the example above).

Note that some compilers (including G++ versions prior to 3.4) get these examples wrong and accept above code without an error. Those compilers do not implement two-stage name lookup correctly.

15.7.3 Temporaries May Vanish Before You Expect

It is dangerous to use pointers or references to *portions* of a temporary object. The compiler may very well delete the object before you expect it to, leaving a pointer to garbage. The most common place where this problem crops up is in classes like string classes, especially ones that define a conversion function to type `char *` or `const char *`—which is one reason why the standard `string` class requires you to call the `c_str` member function. However, any class that returns a pointer to some internal structure is potentially subject to this problem.

For example, a program may use a function `strfunc` that returns `string` objects, and another function `charfunc` that operates on pointers to `char`:

```
string strfunc ();
void charfunc (const char *);

void
f ()
{
    const char *p = strfunc().c_str();
    ...
}
```

```

    charfunc (p);
    ...
    charfunc (p);
}

```

In this situation, it may seem reasonable to save a pointer to the C string returned by the `c_str` member function and use that rather than call `c_str` repeatedly. However, the temporary string created by the call to `strfunc` is destroyed after `p` is initialized, at which point `p` is left pointing to freed memory.

Code like this may run successfully under some other compilers, particularly obsolete cfront-based compilers that delete temporaries along with normal local variables. However, the GNU C++ behavior is standard-conforming, so if your program depends on late destruction of temporaries it is not portable.

The safe way to write such code is to give the temporary a name, which forces it to remain until the end of the scope of the name. For example:

```

const string& tmp = strfunc ();
charfunc (tmp.c_str ());

```

15.7.4 Implicit Copy-Assignment for Virtual Bases

When a base class is virtual, only one subobject of the base class belongs to each full object. Also, the constructors and destructors are invoked only once, and called from the most-derived class. However, such objects behave unspecified when being assigned. For example:

```

struct Base{
    char *name;
    Base(const char *n) : name(strdup(n)){}
    Base& operator= (const Base& other){
        free (name);
        name = strdup (other.name);
        return *this;
    }
};

struct A:virtual Base{
    int val;
    A():Base("A"){ }
};

struct B:virtual Base{
    int bval;
    B():Base("B"){ }
};

struct Derived:public A, public B{
    Derived():Base("Derived"){ }
};

void func(Derived &d1, Derived &d2)
{
    d1 = d2;
}

```

The C++ standard specifies that `Base::Base` is only called once when constructing or copy-constructing a `Derived` object. It is unspecified whether `Base::operator=` is called

more than once when the implicit copy-assignment for Derived objects is invoked (as it is inside `'func'` in the example).

G++ implements the “intuitive” algorithm for copy-assignment: assign all direct bases, then assign all members. In that algorithm, the virtual base subobject can be encountered more than once. In the example, copying proceeds in the following order: `'name'` (via `strdup`), `'val'`, `'name'` again, and `'bval'`.

If application code relies on copy-assignment, a user-defined copy-assignment operator removes any uncertainties. With such an operator, the application can define whether and how the virtual base subobject is assigned.

15.8 Certain Changes We Don't Want to Make

This section lists changes that people frequently request, but which we do not make because we think GCC is better without them.

- Checking the number and type of arguments to a function which has an old-fashioned definition and no prototype.

Such a feature would work only occasionally—only for calls that appear in the same file as the called function, following the definition. The only way to check all calls reliably is to add a prototype for the function. But adding a prototype eliminates the motivation for this feature. So the feature is not worthwhile.

- Warning about using an expression whose type is signed as a shift count.

Shift count operands are probably signed more often than unsigned. Warning about this would cause far more annoyance than good.

- Warning about assigning a signed value to an unsigned variable.

Such assignments must be very common; warning about them would cause more annoyance than good.

- Warning when a non-void function value is ignored.

C contains many standard functions that return a value that most programs choose to ignore. One obvious example is `printf`. Warning about this practice only leads the defensive programmer to clutter programs with dozens of casts to `void`. Such casts are required so frequently that they become visual noise. Writing those casts becomes so automatic that they no longer convey useful information about the intentions of the programmer. For functions where the return value should never be ignored, use the `warn_unused_result` function attribute (see Section 6.4 [Attributes], page 593).

- Making `-fshort-enums` the default.

This would cause storage layout to be incompatible with most other C compilers. And it doesn't seem very important, given that you can get the same result in other ways. The case where it matters most is when the enumeration-valued object is inside a structure, and in that case you can specify a field width explicitly.

- Making bit-fields unsigned by default on particular machines where “the ABI standard” says to do so.

The ISO C standard leaves it up to the implementation whether a bit-field declared plain `int` is signed or not. This in effect creates two alternative dialects of C.

The GNU C compiler supports both dialects; you can specify the signed dialect with `-fsigned-bitfields` and the unsigned dialect with `-funsigned-bitfields`. However, this leaves open the question of which dialect to use by default.

Currently, the preferred dialect makes plain bit-fields signed, because this is simplest. Since `int` is the same as `signed int` in every other context, it is cleanest for them to be the same in bit-fields as well.

Some computer manufacturers have published Application Binary Interface standards which specify that plain bit-fields should be unsigned. It is a mistake, however, to say anything about this issue in an ABI. This is because the handling of plain bit-fields distinguishes two dialects of C. Both dialects are meaningful on every type of machine. Whether a particular object file was compiled using signed bit-fields or unsigned is of no concern to other object files, even if they access the same bit-fields in the same data structures.

A given program is written in one or the other of these two dialects. The program stands a chance to work on most any machine if it is compiled with the proper dialect. It is unlikely to work at all if compiled with the wrong dialect.

Many users appreciate the GNU C compiler because it provides an environment that is uniform across machines. These users would be inconvenienced if the compiler treated plain bit-fields differently on certain machines.

Occasionally users write programs intended only for a particular machine type. On these occasions, the users would benefit if the GNU C compiler were to support by default the same dialect as the other compilers on that machine. But such applications are rare. And users writing a program to run on more than one type of machine cannot possibly benefit from this kind of compatibility.

This is why GCC does and will treat plain bit-fields in the same fashion on all types of machines (by default).

There are some arguments for making bit-fields unsigned by default on all machines. If, for example, this becomes a universal de facto standard, it would make sense for GCC to go along with it. This is something to be considered in the future.

(Of course, users strongly concerned about portability should indicate explicitly in each bit-field whether it is signed or not. In this way, they write programs which have the same meaning in both C dialects.)

- Undefined `__STDC__` when `-ansi` is not used.

Currently, GCC defines `__STDC__` unconditionally. This provides good results in practice.

Programmers normally use conditionals on `__STDC__` to ask whether it is safe to use certain features of ISO C, such as function prototypes or ISO token concatenation. Since plain gcc supports all the features of ISO C, the correct answer to these questions is “yes”.

Some users try to use `__STDC__` to check for the availability of certain library facilities. This is actually incorrect usage in an ISO C program, because the ISO C standard says that a conforming freestanding implementation should define `__STDC__` even though it does not have the library facilities. ‘gcc -ansi -pedantic’ is a conforming freestanding implementation, and it is therefore required to define `__STDC__`, even though it does not come with an ISO C library.

Sometimes people say that defining `__STDC__` in a compiler that does not completely conform to the ISO C standard somehow violates the standard. This is illogical. The standard is a standard for compilers that claim to support ISO C, such as ‘`gcc -ansi`’—not for other compilers such as plain `gcc`. Whatever the ISO C standard says is relevant to the design of plain `gcc` without `-ansi` only for pragmatic reasons, not as a requirement.

GCC normally defines `__STDC__` to be 1, and in addition defines `__STRICT_ANSI__` if you specify the `-ansi` option, or a `-std` option for strict conformance to some version of ISO C. On some hosts, system include files use a different convention, where `__STDC__` is normally 0, but is 1 if the user specifies strict conformance to the C Standard. GCC follows the host convention when processing system include files, but when processing user files it follows the usual GNU C convention.

- Undefined `__STDC__` in C++.

Programs written to compile with C++-to-C translators get the value of `__STDC__` that goes with the C compiler that is subsequently used. These programs must test `__STDC__` to determine what kind of C preprocessor that compiler uses: whether they should concatenate tokens in the ISO C fashion or in the traditional fashion.

These programs work properly with GNU C++ if `__STDC__` is defined. They would not work otherwise.

In addition, many header files are written to provide prototypes in ISO C but not in traditional C. Many of these header files can work without change in C++ provided `__STDC__` is defined. If `__STDC__` is not defined, they will all fail, and will all need to be changed to test explicitly for C++ as well.

- Deleting “empty” loops.

Historically, GCC has not deleted “empty” loops under the assumption that the most likely reason you would put one in a program is to have a delay, so deleting them will not make real programs run any faster.

However, the rationale here is that optimization of a nonempty loop cannot produce an empty one. This held for carefully written C compiled with less powerful optimizers but is not always the case for carefully written C++ or with more powerful optimizers. Thus GCC will remove operations from loops whenever it can determine those operations are not externally visible (apart from the time taken to execute them, of course). In case the loop can be proved to be finite, GCC will also remove the loop itself.

Be aware of this when performing timing tests, for instance the following loop can be completely removed, provided `some_expression` can provably not change any global state.

```
{
    int sum = 0;
    int ix;

    for (ix = 0; ix != 10000; ix++)
        sum += some_expression;
}
```

Even though `sum` is accumulated in the loop, no use is made of that summation, so the accumulation can be removed.

- Making side effects happen in the same order as in some other compiler.

It is never safe to depend on the order of evaluation of side effects. For example, a function call like this may very well behave differently from one compiler to another:

```
void func (int, int);
```

```
int i = 2;
func (i++, i++);
```

There is no guarantee (in either the C or the C++ standard language definitions) that the increments will be evaluated in any particular order. Either increment might happen first. `func` might get the arguments ‘2, 3’, or it might get ‘3, 2’, or even ‘2, 2’.

- Making certain warnings into errors by default.

Some ISO C test suites report failure when the compiler does not produce an error message for a certain program.

ISO C requires a “diagnostic” message for certain kinds of invalid programs, but a warning is defined by GCC to count as a diagnostic. If GCC produces a warning but not an error, that is correct ISO C support. If test suites call this “failure”, they should be run with the GCC option `-pedantic-errors`, which will turn these warnings into errors.

15.9 Warning Messages and Error Messages

The GNU compiler can produce two kinds of diagnostics: errors and warnings. Each kind has a different purpose:

Errors report problems that make it impossible to compile your program. GCC reports errors with the source file name and line number where the problem is apparent.

Warnings report other unusual conditions in your code that *may* indicate a problem, although compilation can (and does) proceed. Warning messages also report the source file name and line number, but include the text ‘**warning:**’ to distinguish them from error messages.

Warnings may indicate danger points where you should check to make sure that your program really does what you intend; or the use of obsolete features; or the use of nonstandard features of GNU C or C++. Many warnings are issued only if you ask for them, with one of the `-W` options (for instance, `-Wall` requests a variety of useful warnings).

GCC always tries to compile your program if possible; it never gratuitously rejects a program whose meaning is clear merely because (for instance) it fails to conform to a standard. In some cases, however, the C and C++ standards specify that certain extensions are forbidden, and a diagnostic *must* be issued by a conforming compiler. The `-pedantic` option tells GCC to issue warnings in such cases; `-pedantic-errors` says to make them errors instead. This does not mean that *all* non-ISO constructs get warnings or errors.

See Section 3.9 [Options to Request or Suppress Warnings], page 101, for more detail on these and related command-line options.

16 Reporting Bugs

Your bug reports play an essential role in making GCC reliable.

When you encounter a problem, the first thing to do is to see if it is already known. See Chapter 15 [Trouble], page 1099. If it isn't known, then you should report the problem.

16.1 Have You Found a Bug?

If you are not sure whether you have found a bug, here are some guidelines:

- If the compiler gets a fatal signal, for any input whatever, that is a compiler bug. Reliable compilers never crash.
- If the compiler produces invalid assembly code, for any input whatever (except an `asm` statement), that is a compiler bug, unless the compiler reports errors (not just warnings) which would ordinarily prevent the assembler from being run.
- If the compiler produces valid assembly code that does not correctly execute the input source code, that is a compiler bug.

However, you must double-check to make sure, because you may have a program whose behavior is undefined, which happened by chance to give the desired results with another C or C++ compiler.

Problems often result from expressions with two increment operators, as in `f (*p++, *p++)`. Your previous compiler might have interpreted that expression the way you intended; GCC might interpret it another way. Neither compiler is wrong. The bug is in your code.

After you have localized the error to a single source line, it should be easy to check for these things. If your program is correct and well defined, you have found a compiler bug.

- If the compiler produces an error message for valid input, that is a compiler bug.
- If the compiler does not produce an error message for invalid input, that is a compiler bug. However, you should note that your idea of “invalid input” might be someone else's idea of “an extension” or “support for traditional practice”.
- If you are an experienced user of one of the languages GCC supports, your suggestions for improvement of GCC are welcome in any case.

16.2 How and Where to Report Bugs

Bugs should be reported to the bug database at <https://gcc.gnu.org/bugs/>.

17 How To Get Help with GCC

If you need help installing, using or changing GCC, there are two ways to find it:

- Send a message to a suitable network mailing list. First try `gcc-help@gcc.gnu.org` (for help installing or using GCC), and if that brings no response, try `gcc@gcc.gnu.org`. For help changing GCC, ask `gcc@gcc.gnu.org`. If you think you have found a bug in GCC, please report it following the instructions at see Section 16.2 [Bug Reporting], page 1115.
- Look in the service directory for someone who might help you for a fee. The service directory is found at <https://www.fsf.org/resources/service>.

For further information, see <https://gcc.gnu.org/faq.html#support>.

18 Contributing to GCC Development

If you would like to help pretest GCC releases to assure they work well, current development sources are available via Git (see <https://gcc.gnu.org/git.html>). Source and binary snapshots are also available for FTP; see <https://gcc.gnu.org/snapshots.html>.

If you would like to work on improvements to GCC, please read the advice at these URLs:

<https://gcc.gnu.org/contribute.html>
<https://gcc.gnu.org/contributewhy.html>

for information on how to make useful contributions and avoid duplication of effort. Suggested projects are listed at <https://gcc.gnu.org/projects/>.

Funding Free Software

If you want to have more free software a few years from now, it makes sense for you to help encourage people to contribute funds for its development. The most effective approach known is to encourage commercial redistributors to donate.

Users of free software systems can boost the pace of development by encouraging for-a-fee distributors to donate part of their selling price to free software developers—the Free Software Foundation, and others.

The way to convince distributors to do this is to demand it and expect it from them. So when you compare distributors, judge them partly by how much they give to free software development. Show distributors they must compete to be the one who gives the most.

To make this approach work, you must insist on numbers that you can compare, such as, “We will donate ten dollars to the Frobnitz project for each disk sold.” Don’t be satisfied with a vague promise, such as “A portion of the profits are donated,” since it doesn’t give a basis for comparison.

Even a precise fraction “of the profits from this disk” is not very meaningful, since creative accounting and unrelated business decisions can greatly alter what fraction of the sales price counts as profit. If the price you pay is \$50, ten percent of the profit is probably less than a dollar; it might be a few cents, or nothing at all.

Some redistributors do development work themselves. This is useful too; but to keep everyone honest, you need to inquire how much they do, and what kind. Some kinds of development make much more long-term difference than others. For example, maintaining a separate version of a program contributes very little; maintaining the standard version of a program for the whole community contributes much. Easy new ports contribute little, since someone else would surely do them; difficult ports such as adding a new CPU to the GNU Compiler Collection contribute more; major new features or packages contribute the most.

By establishing the idea that supporting further development is “the proper thing to do” when distributing free software for a fee, we can assure a steady flow of resources into making more free software.

Copyright © 1994 Free Software Foundation, Inc.

Verbatim copying and redistribution of this section is permitted without royalty; alteration is not permitted.

The GNU Project and GNU/Linux

The GNU Project was launched in 1984 to develop a complete Unix-like operating system which is free software: the GNU system. (GNU is a recursive acronym for “GNU’s Not Unix”; it is pronounced “guh-NEW”.) Variants of the GNU operating system, which use the kernel Linux, are now widely used; though these systems are often referred to as “Linux”, they are more accurately called GNU/Linux systems.

For more information, see:

<https://www.gnu.org/>

<https://www.gnu.org/gnu/linux-and-gnu.html>

GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <https://www.fsf.org>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

- d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so

available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author
```

```
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.
```

```
This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program. If not, see https://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
<https://www.fsf.org>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

- be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
 - C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
 - D. Preserve all the copyright notices of the Document.
 - E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
 - F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
 - G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
 - H. Include an unaltered copy of this License.
 - I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
 - J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
 - K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
 - L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
 - M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
 - N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
 - O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <https://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Contributors to GCC

The GCC project would like to thank its many contributors. Without them the project would not have been nearly as successful as it has been. Any omissions in this list are accidental. Feel free to contact jlaw@ventanamicro.com or gerald@pfeifer.com if you have been left out or some of your contributions are not listed. Please keep this list in alphabetical order.

- Analog Devices helped implement the support for complex data types and iterators.
- John David Anglin for threading-related fixes and improvements to libstdc++-v3, and the HP-UX port.
- James van Artsdalen wrote the code that makes efficient use of the Intel 80387 register stack.
- Abramo and Roberto Bagnara for the SysV68 Motorola 3300 Delta Series port.
- Alasdair Baird for various bug fixes.
- Giovanni Bajo for analyzing lots of complicated C++ problem reports.
- Peter Barada for his work to improve code generation for new ColdFire cores.
- Gerald Baumgartner added the signature extension to the C++ front end.
- Godmar Back for his Java improvements and encouragement.
- Scott Bambrough for help porting the Java compiler.
- Wolfgang Bangerth for processing tons of bug reports.
- Jon Beniston for his Microsoft Windows port of Java and port to Lattice Mico32.
- Daniel Berlin for better DWARF 2 support, faster/better optimizations, improved alias analysis, plus migrating GCC to Bugzilla.
- Geoff Berry for his Java object serialization work and various patches.
- Richard Biener for his ongoing middle-end contributions and bug fixes and for release management.
- David Binderman for testing GCC trunk against Fedora Rawhide and csmith.
- Laurynas Biveinis for memory management work and DJGPP port fixes.
- Uros Bizjak for the implementation of x87 math built-in functions and for various middle end and i386 back end improvements and bug fixes.
- Eric Blake for helping to make GCJ and libgcj conform to the specifications.
- Janne Blomqvist for contributions to GNU Fortran.
- Hans-J. Boehm for his garbage collector, IA-64 libffi port, and other Java work.
- Segher Boessenkool for helping maintain the PowerPC port and the instruction combiner plus various contributions to the middle end.
- Neil Booth for work on cpplib, lang hooks, debug hooks and other miscellaneous clean-ups.
- Steven Bosscher for integrating the GNU Fortran front end into GCC and for contributing to the tree-ssa branch.
- Eric Botcazou for fixing middle- and backend bugs left and right.

- Per Bothner for his direction via the steering committee and various improvements to the infrastructure for supporting new languages. Chill front end implementation. Initial implementations of cpplib, fix-header, config.guess, libio, and past C++ library (libg++) maintainer. Dreaming up, designing and implementing much of GCJ.
- Devon Bowen helped port GCC to the Tahoe.
- Don Bowman for mips-vxworks contributions.
- James Bowman for the FT32 port.
- Dave Brolley for work on cpplib and Chill.
- Paul Brook for work on the ARM architecture and maintaining GNU Fortran.
- Robert Brown implemented the support for Encore 32000 systems.
- Christian Bruel for improvements to local store elimination.
- Herman A.J. ten Brugge for various fixes.
- Joerg Brunsmann for Java compiler hacking and help with the GCJ FAQ.
- Joe Buck for his direction via the steering committee from its creation to 2013.
- Iain Buclaw for the D frontend.
- Craig Burley for leadership of the G77 Fortran effort.
- Tobias Burnus for contributions to GNU Fortran.
- Stephan Buys for contributing Doxygen notes for libstdc++.
- Paolo Carlini for libstdc++ work: lots of efficiency improvements to the C++ strings, streambufs and formatted I/O, hard detective work on the frustrating localization issues, and keeping up with the problem reports.
- John Carr for his alias work, SPARC hacking, infrastructure improvements, previous contributions to the steering committee, loop optimizations, etc.
- Stephane Carrez for 68HC11 and 68HC12 ports.
- Steve Chamberlain for support for the Renesas SH and H8 processors and the PicoJava processor, and for GCJ config fixes.
- Glenn Chambers for help with the GCJ FAQ.
- John-Marc Chandonia for various libgcj patches.
- Denis Chertykov for contributing and maintaining the AVR port, the first GCC port for an 8-bit architecture.
- Kito Cheng for his work on the RISC-V port, including bringing up the test suite and maintenance.
- Scott Christley for his Objective-C contributions.
- Eric Christopher for his Java porting help and clean-ups.
- Branko Cibej for more warning contributions.
- The GNU Classpath project for all of their merged runtime code.
- Nick Clifton for arm, mcore, fr30, v850, m32r, msp430 rx work, `--help`, and other random hacking.
- Michael Cook for libstdc++ cleanup patches to reduce warnings.
- R. Kelley Cook for making GCC buildable from a read-only directory as well as other miscellaneous build process and documentation clean-ups.

- Ralf Corsepius for SH testing and minor bug fixing.
- François-Xavier Coudert for contributions to GNU Fortran.
- Stan Cox for care and feeding of the x86 port and lots of behind the scenes hacking.
- Alex Crain provided changes for the 3b1.
- Ian Dall for major improvements to the NS32k port.
- Paul Dale for his work to add uClinux platform support to the m68k backend.
- Palmer Dabbelt for his work maintaining the RISC-V port.
- Dario Dariol contributed the four varieties of sample programs that print a copy of their source.
- Russell Davidson for fstream and stringstream fixes in libstdc++.
- Bud Davis for work on the G77 and GNU Fortran compilers.
- Mo DeJong for GCJ and libgcj bug fixes.
- Jerry DeLisle for contributions to GNU Fortran.
- DJ Delorie for the DJGPP port, build and libiberty maintenance, various bug fixes, and the M32C, MeP, MSP430, and RL78 ports.
- Arnaud Desitter for helping to debug GNU Fortran.
- Gabriel Dos Reis for contributions to G++, contributions and maintenance of GCC diagnostics infrastructure, libstdc++-v3, including `valarray<>`, `complex<>`, maintaining the numerics library (including that pesky `<limits>` :-)) and keeping up-to-date anything to do with numbers.
- Ulrich Drepper for his work on glibc, testing of GCC using glibc, ISO C99 support, CFG dumping support, etc., plus support of the C++ runtime libraries including for all kinds of C interface issues, contributing and maintaining `complex<>`, sanity checking and disbursement, configuration architecture, libio maintenance, and early math work.
- Robert J. Dubner for his work on the COBOL front end, mating the parser output to the GENERIC tree.
- François Dumont for his work on libstdc++-v3, especially maintaining and improving `debug-mode` and associative and unordered containers.
- Zdenek Dvorak for a new loop unroller and various fixes.
- Michael Eager for his work on the Xilinx MicroBlaze port.
- Richard Earnshaw for his ongoing work with the ARM.
- David Edelsohn for his direction via the steering committee, ongoing work with the RS6000/PowerPC port, help cleaning up Haifa loop changes, doing the entire AIX port of libstdc++ with his bare hands, and for ensuring GCC properly keeps working on AIX.
- Kevin Ediger for the floating point formatting of `num_put::do_put` in libstdc++.
- Phil Edwards for libstdc++ work including configuration hackery, documentation maintainer, chief breaker of the web pages, the occasional iostream bug fix, and work on shared library symbol versioning.
- Paul Eggert for random hacking all over GCC.
- Mark Elbrecht for various DJGPP improvements, and for libstdc++ configuration support for locales and fstream-related fixes.

- Vadim Egorov for libstdc++ fixes in strings, streambufs, and iostreams.
- Christian Ehrhardt for dealing with bug reports.
- Ben Elliston for his work to move the Objective-C runtime into its own subdirectory and for his work on autoconf.
- Oleg Endo for continued development and maintenance of the SuperH back-end.
- Revital Eres for work on the PowerPC 750CL port.
- Marc Espie for OpenBSD support.
- Doug Evans for much of the global optimization framework, arc, m32r, and SPARC work.
- Christopher Faylor for his work on the Cygwin port and for caring and feeding the gcc.gnu.org box and saving its users tons of spam.
- Fred Fish for BeOS support and Ada fixes.
- Ivan Fontes Garcia for the Portuguese translation of the GCJ FAQ.
- Peter Gerwinski for various bug fixes and the Pascal front end.
- Kaveh R. Ghazi for his direction via the steering committee, amazing work to make ‘-W -Wall -W* -Werror’ useful, and testing GCC on a plethora of platforms. Kaveh extends his gratitude to the CAIP Center at Rutgers University for providing him with computing resources to work on Free Software from the late 1980s to 2010.
- John Gilmore for a donation to the FSF earmarked improving GNU Java.
- Judy Goldberg for c++ contributions.
- Torbjorn Granlund for various fixes and the c-torture testsuite, multiply- and divide-by-constant optimization, improved long long support, improved leaf function register allocation, and his direction via the steering committee.
- Jonny Grant for improvements to collect2's --help documentation.
- Anthony Green for his -Os contributions, the moxie port, and Java front end work.
- Stu Grossman for gdb hacking, allowing GCJ developers to debug Java code.
- Michael K. Gschwind contributed the port to the PDP-11.
- Ron Guilmette implemented the protoize and unprotoize tools, the support for DWARF 1 symbolic debugging information, and much of the support for System V Release 4. He has also worked heavily on the Intel 386 and 860 support.
- Sumanth Gundapaneni for contributing the CR16 port.
- Mostafa Hagog for Swing Modulo Scheduling (SMS) and post reload GCSE.
- Bruno Haible for improvements in the runtime overhead for EH, new warnings and assorted bug fixes.
- Andrew Haley for his amazing Java compiler and library efforts.
- Chris Hanson assisted in making GCC work on HP-UX for the 9000 series 300.
- Michael Hayes for various thankless work he's done trying to get the c30/c40 ports functional. Lots of loop and unroll improvements and fixes.
- Dara Hazeghi for wading through myriads of target-specific bug reports.
- Kate Hedstrom for staking the G77 folks with an initial testsuite.

- Richard Henderson for his ongoing SPARC, alpha, ia32, and ia64 work, loop opts, and generally fixing lots of old problems we've ignored for years, flow rewrite and lots of further stuff, including reviewing tons of patches.
- Aldy Hernandez for working on the PowerPC port, SIMD support, and various fixes.
- Nobuyuki Hikichi of Software Research Associates, Tokyo, contributed the support for the Sony NEWS machine.
- Kazu Hirata for caring and feeding the Renesas H8/300 port and various fixes.
- Katherine Holcomb for work on GNU Fortran.
- Manfred Hollstein for his ongoing work to keep the m88k alive, lots of testing and bug fixing, particularly of GCC configury code.
- Steve Holmgren for MachTen patches.
- Mat Hostetter for work on the TILE-Gx and TILEPro ports.
- Jan Hubicka for his x86 port improvements.
- Falk Hueffner for working on C and optimization bug reports.
- Bernardo Innocenti for his m68k work, including merging of ColdFire improvements and uClinux support.
- Christian Iseli for various bug fixes.
- Kamil Iskra for general m68k hacking.
- Lee Iversen for random fixes and MIPS testing.
- Balaji V. Iyer for Cilk+ development and merging.
- Andreas Jaeger for testing and benchmarking of GCC and various bug fixes.
- Martin Jambor for his work on inter-procedural optimizations, the switch conversion pass, and scalar replacement of aggregates.
- Jakub Jelinek for his SPARC work and sibling call optimizations as well as lots of bug fixes and test cases, and for improving the Java build system.
- Janis Johnson for ia64 testing and fixes, her quality improvement sidetracks, and web page maintenance.
- Kean Johnston for SCO OpenServer support and various fixes.
- Tim Josling for the sample language treelang based originally on Richard Kenner's "toy" language.
- Nicolai Josuttis for additional libstdc++ documentation.
- Klaus Kaempf for his ongoing work to make alpha-vms a viable target.
- Steven G. Kargl for work on GNU Fortran.
- David Kashtan of SRI adapted GCC to VMS.
- Ryszard Kabatek for many, many libstdc++ bug fixes and optimizations of strings, especially member functions, and for auto_ptr fixes.
- Geoffrey Keating for his ongoing work to make the PPC work for GNU/Linux and his automatic regression tester.
- Brendan Kehoe for his ongoing work with G++ and for a lot of early work in just about every part of libstdc++.
- Oliver M. Kellogg of Deutsche Aerospace contributed the port to the MIL-STD-1750A.

- Richard Kenner of the New York University Ultracomputer Research Laboratory wrote the machine descriptions for the AMD 29000, the DEC Alpha, the IBM RT PC, and the IBM RS/6000 as well as the support for instruction attributes. He also made changes to better support RISC processors including changes to common subexpression elimination, strength reduction, function calling sequence handling, and condition code support, in addition to generalizing the code for frame pointer elimination and delay slot scheduling. Richard Kenner was also the head maintainer of GCC for several years.
- Mumit Khan for various contributions to the Cygwin and Mingw32 ports and maintaining binary releases for Microsoft Windows hosts, and for massive libstdc++ porting work to Cygwin/Mingw32.
- Robin Kirkham for cpu32 support.
- Mark Klein for PA improvements.
- Thomas Koenig for various bug fixes.
- Kazumoto Kojima for continued development and maintenance of the SuperH back-end.
- Bruce Korb for the new and improved fixincludes code.
- Benjamin Kosnik for his G++ work and for leading the libstdc++-v3 effort.
- Maxim Kuvyrkov for contributions to the instruction scheduler, the Android and m68k/Coldfire ports, and optimizations.
- Charles LaBrec contributed the support for the Integrated Solutions 68020 system.
- Asher Langton and Mike Kumbera for contributing Cray pointer support to GNU Fortran, and for other GNU Fortran improvements.
- Jeff Law for his direction via the steering committee, coordinating the entire egcs project and GCC 2.95, rolling out snapshots and releases, handling merges from GCC2, reviewing tons of patches that might have fallen through the cracks else, and random but extensive hacking.
- Walter Lee for work on the TILE-Gx and TILEPro ports.
- Marc Lehmann for his direction via the steering committee and helping with analysis and improvements of x86 performance.
- Victor Leikehman for work on GNU Fortran.
- Ted Lemon wrote parts of the RTL reader and printer.
- Kriang Lerdsuwanakij for C++ improvements including template as template parameter support, and many C++ fixes.
- Warren Levy for tremendous work on libgcj (Java Runtime Library) and random work on the Java front end.
- Alain Lichnewsky ported GCC to the MIPS CPU.
- Oskar Liljeblad for hacking on AWT and his many Java bug reports and patches.
- Robert Lipe for OpenServer support, new testsuites, testing, etc.
- Chen Liqin for various S+core related fixes/improvement, and for maintaining the S+core port.
- Martin Liska for his work on identical code folding, the sanitizers, HSA, general bug fixing and for running automated regression testing of GCC and reporting numerous bugs.

- Weiwen Liu for testing and various bug fixes.
- Manuel López-Ibáñez for improving `-Wconversion` and many other diagnostics fixes and improvements.
- Dave Love for his ongoing work with the Fortran front end and runtime libraries.
- James K. Lowden for his work on the COBOL front end, mainly the parser and CDF.
- Martin von Löwis for internal consistency checking infrastructure, various C++ improvements including namespace support, and tons of assistance with libstdc++/compiler merges.
- H.J. Lu for his previous contributions to the steering committee, many x86 bug reports, prototype patches, and keeping the GNU/Linux ports working.
- Greg McGary for random fixes and (someday) bounded pointers.
- Andrew MacLeod for his ongoing work in building a real EH system, various code generation improvements, work on the global optimizer, etc.
- Vladimir Makarov for hacking some ugly i960 problems, PowerPC hacking improvements to compile-time performance, overall knowledge and direction in the area of instruction scheduling, design and implementation of the automaton based instruction scheduler and design and implementation of the integrated and local register allocators.
- David Malcolm for his work on improving GCC diagnostics, JIT, self-tests and unit testing.
- Bob Manson for his behind the scenes work on dejagnu.
- Jose E. Marchesi for contributing the eBPF backend and his ongoing work maintaining it.
- John Marino for contributing the DragonFly BSD port.
- Philip Martin for lots of libstdc++ string and vector iterator fixes and improvements, and string clean up and testsuites.
- Dhruv Matani for work on libstdc++ allocators.
- Michael Matz for his work on dominance tree discovery, the x86-64 port, link-time optimization framework and general optimization improvements.
- All of the Mauve project contributors for Java test code.
- Bryce McKinlay for numerous GCJ and libgcj fixes and improvements.
- Adam Megacz for his work on the Microsoft Windows port of GCJ.
- Michael Meissner for LRS framework, ia32, m32r, v850, m88k, MIPS, powerpc, haifa, ECOFF debug support, and other assorted hacking.
- Jason Merrill for his direction via the steering committee and leading the G++ effort.
- Martin Michlmayr for testing GCC on several architectures using the entire Debian archive.
- David Miller for his direction via the steering committee, lots of SPARC work, improvements in jump.cc and interfacing with the Linux kernel developers.
- Gary Miller ported GCC to Charles River Data Systems machines.
- Alfred Minarik for libstdc++ string and ios bug fixes, and turning the entire libstdc++ testsuite namespace-compatible.

- Mark Mitchell for his direction via the steering committee, mountains of C++ work, load/store hoisting out of loops, alias analysis improvements, ISO C `restrict` support, and serving as release manager from 2000 to 2011.
- Alan Modra for various GNU/Linux bits and testing.
- Toon Moene for his direction via the steering committee, Fortran maintenance, and his ongoing work to make us make Fortran run fast.
- Jason Molenda for major help in the care and feeding of all the services on the gcc.gnu.org (formerly egcs.cygnus.com) machine—mail, web services, ftp services, etc etc. Doing all this work on scrap paper and the backs of envelopes would have been . . . difficult.
- Catherine Moore for fixing various ugly problems we have sent her way, including the haifa bug which was killing the Alpha & PowerPC Linux kernels.
- Mike Moreton for his various Java patches.
- David Mosberger-Tang for various Alpha improvements, and for the initial IA-64 port.
- Stephen Moshier contributed the floating point emulator that assists in cross-compilation and permits support for floating point numbers wider than 64 bits and for ISO C99 support.
- Bill Moyer for his behind the scenes work on various issues.
- Philippe De Muyter for his work on the m68k port.
- Joseph S. Myers for his work on the PDP-11 port, format checking and ISO C99 support, and continuous emphasis on (and contributions to) documentation.
- Nathan Myers for his work on libstdc++-v3: architecture and authorship through the first three snapshots, including implementation of locale infrastructure, string, shadow C headers, and the initial project documentation (DESIGN, CHECKLIST, and so forth). Later, more work on MT-safe string and shadow headers.
- Felix Natter for documentation on porting libstdc++.
- Nathanael Nerode for cleaning up the configuration/build process.
- NeXT, Inc. donated the front end that supports the Objective-C language.
- Hans-Peter Nilsson for the CRIS and MMIX ports, improvements to the search engine setup, various documentation fixes and other small fixes.
- Geoff Noer for his work on getting cygwin native builds working.
- Vegard Nossum for running automated regression testing of GCC and reporting numerous bugs.
- Diego Novillo for his work on Tree SSA, OpenMP, SPEC performance tracking web pages, GIMPLE tuples, and assorted fixes.
- David O'Brien for the FreeBSD/alpha, FreeBSD/AMD x86-64, FreeBSD/ARM, FreeBSD/PowerPC, and FreeBSD/SPARC64 ports and related infrastructure improvements.
- Alexandre Oliva for various build infrastructure improvements, scripts and amazing testing work, including keeping libtool issues sane and happy.
- Stefan Olsson for work on mt_alloc.
- Melissa O'Neill for various NeXT fixes.

- Rainer Orth for random MIPS work, including improvements to GCC's o32 ABI support, improvements to dejagnu's MIPS support, Java configuration clean-ups and porting work, and maintaining the IRIX, Solaris 2, and Tru64 UNIX ports.
- Patrick Palka for contributions to the C++ library and front end.
- Steven Pemberton for his contribution of `enquire` which allowed GCC to determine various properties of the floating point unit and generate `float.h` in older versions of GCC.
- Hartmut Penner for work on the s390 port.
- Paul Petersen wrote the machine description for the Alliant FX/8.
- Alexandre Petit-Bianco for implementing much of the Java compiler and continued Java maintainership.
- Matthias Pfaller for major improvements to the NS32k port.
- Gerald Pfeifer for his direction via the steering committee, pointing out lots of problems we need to solve, maintenance of the web pages, and taking care of documentation maintenance in general.
- Marek Polacek for his work on the C front end, the sanitizers and general bug fixing.
- Andrew Pinski for processing bug reports by the dozen, maintenance of the Objective-C runtime libraries, and many scalar optimizations.
- Ovidiu Predescu for his work on the Objective-C front end and runtime libraries.
- Jerry Quinn for major performance improvements in C++ formatted I/O.
- Ken Raeburn for various improvements to checker, MIPS ports and various cleanups in the compiler.
- Rolf W. Rasmussen for hacking on AWT.
- David Reese of Sun Microsystems contributed to the Solaris on PowerPC port.
- John Regehr for running automated regression testing of GCC and reporting numerous bugs.
- Volker Reichelt for running automated regression testing of GCC and reporting numerous bugs and for keeping up with the problem reports.
- Joern Rennecke for maintaining the sh port, loop, regmove & reload hacking and developing and maintaining the Epiphany port.
- Loren J. Rittle for improvements to libstdc++-v3 including the FreeBSD port, threading fixes, thread-related configury changes, critical threading documentation, and solutions to really tricky I/O problems, as well as keeping GCC properly working on FreeBSD and continuous testing.
- Craig Rodrigues for processing tons of bug reports.
- Ola Rönnerup for work on `mt_alloc`.
- Gavin Romig-Koch for lots of behind the scenes MIPS work.
- David Ronis inspired and encouraged Craig to rewrite the G77 documentation in texinfo format by contributing a first pass at a translation of the old `g77-0.5.16/f/DOC` file.
- Ken Rose for fixes to GCC's delay slot filling code.
- Ira Rosen for her contributions to the auto-vectorizer.

- Paul Rubin wrote most of the preprocessor.
- Pétur Runólfsson for major performance improvements in C++ formatted I/O and large file support in C++ filebuf.
- Chip Salzenberg for libstdc++ patches and improvements to locales, traits, Makefiles, libio, libtool hackery, and “long long” support.
- Juha Sarlin for improvements to the H8 code generator.
- Greg Satz assisted in making GCC work on HP-UX for the 9000 series 300.
- Roger Sayle for improvements to constant folding and GCC’s RTL optimizers as well as for fixing numerous bugs.
- Bradley Schatz for his work on the GCJ FAQ.
- Peter Schauer wrote the code to allow debugging to work on the Alpha.
- William Schelter did most of the work on the Intel 80386 support.
- Tobias Schlüter for work on GNU Fortran.
- Bernd Schmidt for various code generation improvements and major work in the reload pass, serving as release manager for GCC 2.95.3, and work on the Blackfin and C6X ports.
- Peter Schmid for constant testing of libstdc++—especially application testing, going above and beyond what was requested for the release criteria—and libstdc++ header file tweaks.
- Jason Schroeder for jcf-dump patches.
- Andreas Schwab for his work on the m68k port.
- Lars Segerlund for work on GNU Fortran.
- Dodji Seketeli for numerous C++ bug fixes and debug info improvements.
- Tim Shen for major work on `<regex>`.
- Joel Sherrill for his direction via the steering committee, RTEMS contributions and RTEMS testing.
- Nathan Sidwell for many C++ fixes/improvements.
- Jeffrey Siegal for helping RMS with the original design of GCC, some code which handles the parse tree and RTL data structures, constant folding and help with the original VAX & m68k ports.
- Kenny Simpson for prompting libstdc++ fixes due to defect reports from the LWG (thereby keeping GCC in line with updates from the ISO).
- Franz Sirl for his ongoing work with making the PPC port stable for GNU/Linux.
- Andrey Slepukhin for assorted AIX hacking.
- Trevor Smigiel for contributing the SPU port.
- Christopher Smith did the port for Convex machines.
- Danny Smith for his major efforts on the Mingw (and Cygwin) ports. Retired from GCC maintainership August 2010, having mentored two new maintainers into the role.
- Randy Smith finished the Sun FPA support.
- Ed Smith-Rowland for his continuous work on libstdc++-v3, special functions, `<random>`, and various improvements to C++11 features.

- Scott Snyder for queue, iterator, istream, and string fixes and libstdc++ testsuite entries. Also for providing the patch to G77 to add rudimentary support for `INTEGER*1`, `INTEGER*2`, and `LOGICAL*1`.
- Zdenek Sojka for running automated regression testing of GCC and reporting numerous bugs.
- Arseny Solokha for running automated regression testing of GCC and reporting numerous bugs.
- Jayant Sonar for contributing the CR16 port.
- Brad Spencer for contributions to the `GLIBCPP_FORCE_NEW` technique.
- Richard Stallman, for writing the original GCC and launching the GNU project.
- Jan Stein of the Chalmers Computer Society provided support for Genix, as well as part of the 32000 machine description.
- Gerhard Steinmetz for running automated regression testing of GCC and reporting numerous bugs.
- Nigel Stephens for various mips16 related fixes/improvements.
- Jonathan Stone wrote the machine description for the Pyramid computer.
- Graham Stott for various infrastructure improvements.
- John Stracke for his Java HTTP protocol fixes.
- Mike Stump for his Elxsi port, G++ contributions over the years and more recently his vxworks contributions
- Jeff Sturm for Java porting help, bug fixes, and encouragement.
- Zhendong Su for running automated regression testing of GCC and reporting numerous bugs.
- Chengnian Sun for running automated regression testing of GCC and reporting numerous bugs.
- Shigeya Suzuki for this fixes for the bsdi platforms.
- Ian Lance Taylor for the Go frontend, the initial mips16 and mips64 support, general configury hacking, fixincludes, etc.
- Holger Teutsch provided the support for the Clipper CPU.
- Gary Thomas for his ongoing work to make the PPC work for GNU/Linux.
- Paul Thomas for contributions to GNU Fortran.
- Philipp Thomas for random bug fixes throughout the compiler
- Jason Thorpe for thread support in libstdc++ on NetBSD.
- Kresten Krab Thorup wrote the run time support for the Objective-C language and the fantastic Java bytecode interpreter.
- Michael Tiemann for random bug fixes, the first instruction scheduler, initial C++ support, function integration, NS32k, SPARC and M88k machine description work, delay slot scheduling.
- Andreas Tobler for his work porting libgcj to Darwin.
- Teemu Torma for thread safe exception handling support.
- Leonard Tower wrote parts of the parser, RTL generator, and RTL definitions, and of the VAX machine description.

- Daniel Towner and Hariharan Sandanagobalane contributed and maintain the picoChip port.
- Tom Tromey for internationalization support and for his many Java contributions and libgcj maintainership.
- Lassi Tuura for improvements to config.guess to determine HP processor types.
- Petter Urkedal for libstdc++ CXXFLAGS, math, and algorithms fixes.
- Andy Vaught for the design and initial implementation of the GNU Fortran front end.
- Brent Verner for work with the libstdc++ cshadow files and their associated configure steps.
- Todd Vierling for contributions for NetBSD ports.
- Andrew Waterman for contributing the RISC-V port, as well as maintaining it.
- Jonathan Wakely for contributing to and maintaining libstdc++.
- Dean Wakerley for converting the install documentation from HTML to texinfo in time for GCC 3.0.
- Krister Walfridsson for random bug fixes.
- Feng Wang for contributions to GNU Fortran.
- Stephen M. Webb for time and effort on making libstdc++ shadow files work with the tricky Solaris 8+ headers, and for pushing the build-time header tree. Also, for starting and driving the <regex> effort.
- John Wehle for various improvements for the x86 code generator, related infrastructure improvements to help x86 code generation, value range propagation and other work, WE32k port.
- Ulrich Weigand for work on the s390 port.
- Janus Weil for contributions to GNU Fortran.
- Zack Weinberg for major work on cpplib and various other bug fixes.
- Matt Welsh for help with Linux Threads support in GCJ.
- Urban Widmark for help fixing java.io.
- Mark Wielaard for new Java library code and his work integrating with Classpath.
- Dale Wiles helped port GCC to the Tahoe.
- Bob Wilson from Tensilica, Inc. for the Xtensa port.
- Jim Wilson for his direction via the steering committee, tackling hard problems in various places that nobody else wanted to work on, strength reduction and other loop optimizations.
- Paul Woegerer and Tal Agmon for the CRX port.
- Carlo Wood for various fixes.
- Tom Wood for work on the m88k port.
- Chung-Ju Wu for his work on the Andes NDS32 port.
- Canqun Yang for work on GNU Fortran.
- Masanobu Yuhara of Fujitsu Laboratories implemented the machine description for the Tron architecture (specifically, the Gmicro).
- Kevin Zachmann helped port GCC to the Tahoe.

- Ayal Zaks for Swing Modulo Scheduling (SMS).
- Qirun Zhang for running automated regression testing of GCC and reporting numerous bugs.
- Xiaoqiang Zhang for work on GNU Fortran.
- Gilles Zunino for help porting Java to Irix.

The following people are recognized for their contributions to GNAT, the Ada front end of GCC:

- Bernard Banner
- Romain Berrendonner
- Geert Bosch
- Emmanuel Briot
- Joel Brobecker
- Ben Brosgol
- Vincent Celier
- Arnaud Charlet
- Chien Chieng
- Cyrille Comar
- Cyrille Crozes
- Robert Dewar
- Gary Dismukes
- Robert Duff
- Ed Falis
- Ramon Fernandez
- Sam Figueroa
- Vasiliy Fofanov
- Michael Friess
- Franco Gasperoni
- Ted Giering
- Matthew Gingell
- Laurent Guerby
- Jerome Guitton
- Olivier Hainque
- Jerome Hugues
- Hristian Kirtchev
- Jerome Lambourg
- Bruno Leclerc
- Albert Lee
- Sean McNeil
- Javier Miranda

- Laurent Nana
- Pascal Obry
- Dong-Ik Oh
- Laurent Pautet
- Brett Porter
- Thomas Quinot
- Nicolas Roche
- Pat Rogers
- Jose Ruiz
- Douglas Rupp
- Sergey Rybin
- Gail Schenker
- Ed Schonberg
- Nicolas Setton
- Samuel Tardieu

The following people are recognized for their contributions of new features, bug reports, testing and integration of classpath/libgcj for GCC version 4.1:

- Lillian Angel for **JTree** implementation and lots Free Swing additions and bug fixes.
- Wolfgang Baer for **GapContent** bug fixes.
- Anthony Balkissoon for **JList**, Free Swing 1.5 updates and mouse event fixes, lots of Free Swing work including **JTable** editing.
- Stuart Ballard for RMI constant fixes.
- Goffredo Baroncelli for **URLConnection** fixes.
- Gary Benson for **MessageFormat** fixes.
- Daniel Bonniot for **Serialization** fixes.
- Chris Burdess for lots of gnu.xml and http protocol fixes, **StAX** and **DOM xml:id** support.
- Ka-Hing Cheung for **TreePath** and **TreeSelection** fixes.
- Archie Cobbs for build fixes, VM interface updates, **URLClassLoader** updates.
- Kelley Cook for build fixes.
- Martin Cordova for Suggestions for better **SocketTimeoutException**.
- David Daney for **BitSet** bug fixes, **URLConnection** rewrite and improvements.
- Thomas Fitzsimmons for lots of upgrades to the gtk+ AWT and Cairo 2D support. Lots of imageio framework additions, lots of AWT and Free Swing bug fixes.
- Jeroen Frijters for **ClassLoader** and nio cleanups, serialization fixes, better **Proxy** support, bug fixes and IKVM integration.
- Santiago Gala for **AccessControlContext** fixes.
- Nicolas Geoffray for **VMClassLoader** and **AccessController** improvements.
- David Gilbert for **basic** and **metal** icon and plaf support and lots of documenting, Lots of Free Swing and metal theme additions. **MetalIconFactory** implementation.

- Anthony Green for MIDI framework, ALSA and DSSI providers.
- Andrew Haley for `Serialization` and `URLClassLoader` fixes, gcj build speedups.
- Kim Ho for `JFileChooser` implementation.
- Andrew John Hughes for `Locale` and net fixes, URI RFC2986 updates, `Serialization` fixes, `Properties` XML support and generic branch work, `VMIntegration` guide update.
- Bastiaan Huisman for `TimeZone` bug fixing.
- Andreas Jaeger for mprec updates.
- Paul Jenner for better `-Werror` support.
- Ito Kazumitsu for `NetworkInterface` implementation and updates.
- Roman Kennke for `BoxLayout`, `GrayFilter` and `SplitPane`, plus bug fixes all over. Lots of Free Swing work including styled text.
- Simon Kitching for `String` cleanups and optimization suggestions.
- Michael Koch for configuration fixes, `Locale` updates, bug and build fixes.
- Guilhem Lavaux for configuration, thread and channel fixes and Kaffe integration. JCL native `Pointer` updates. Logger bug fixes.
- David 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
- Ilewelly
- 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, 795, 1111
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
dumpspeaks 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.....	207	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.....	186	fcontract-checks-outlined.....	57
fanalyzer-feasibility.....	186	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.....	206
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.....	192
fauto-profile.....	237	fdebug-types-section.....	193
fauto-profile-inlining.....	237	fdeclone-ctor-dtor.....	207
favoid-store-forwarding.....	201	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, 1101	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.....	189	fdump-rtl-mode_sw.....	300
fdump-analyzer-state-purge.....	189	fdump-rtl-outof_cfglayout.....	300
fdump-analyzer-stderr.....	188	fdump-rtl-pass.....	298
fdump-analyzer-supergraph.....	189	fdump-rtl-peephole2.....	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.....	196	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.....	203
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, 1106	fipa-cp.....	215
ffold-mem-offsets.....	236	fipa-cp-clone.....	215
ffold-simple-inlines.....	59	fipa-icf.....	216
fforward-propagate.....	201	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.....	205	fipa-ptg.....	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	192
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	210	fmodulo-sched	204
fira-verbose	312	fmodulo-sched-allow-regmoves	204
fiolate-erroneous-paths-attribute	217	fmove-loop-invariants	243
fiolate-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	204, 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	207
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	186
flive-range-shrinkage	209	fno-analyzer-feasibility	186
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	201

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, 795	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	201
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	205
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, 1032	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, 1034	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.....	203	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.....	192
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.....	210	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.....	204	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.....	202
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	211
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-ctors	83
fno-tree-phi-prop	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	202
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, 1103	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, 837	fsanitize=mementag-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	211	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, 1110	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, 1110	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, 1110
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.....	194
gbtf.....	190
gcodeview.....	191
gcolumn-info.....	194
gctf.....	190
gdescribe-dies.....	193
gdwarf.....	190
gdwarf32.....	192
gdwarf64.....	192
gen-decls.....	85
gfull.....	383
ggdb.....	189
ggnu-pubnames.....	193
ginline-points.....	195
ginternal-reset-location-views.....	195
gno-as-loc-support.....	194
gno-as-locview-support.....	194
gno-codeview.....	191
gno-column-info.....	194
gno-describe-dies.....	193
gno-inline-points.....	195
gno-internal-reset-location-views.....	195
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.....	193
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.....	191
gz.....	195
G.....	339, 408, 412, 427, 481, 507

H

headerpad_max_install_names.....	385
help.....	41
H.....	274

I

I	282
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

K

keep_private_extrns	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, 375, 397, 400, 405, 412, 420,	
mabs=legacy.....	425	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-syntax-unified.....	357
maddress-mode=short.....	548	masm=diagnostic.....	392, 541
madjust-lmul-cost.....	467	matomic.....	335
mads.....	479	matomic-libcalls.....	398
madx.....	530	matomic-model=model.....	497
maes.....	529	matt-stubs.....	383
maix-struct-return.....	478	mauto-litpools.....	550
maix32.....	473	mauto-modify-reg.....	339
maix64.....	473	mauto-pic.....	402
malign-300.....	397	mautovec-preference.....	320
malign-data.....	466, 534	mautovec-segment.....	467
malign-double.....	534	mavoid-indexed-addresses.....	475
malign-functions.....	441	mavx.....	527
malign-int.....	416	mavx10.1.....	529
malign-labels.....	394	mavx10.2.....	529
malign-loops.....	412	mavx2.....	527
malign-natural.....	474	mavx256-split-unaligned-load.....	546
malign-power.....	474	mavx256-split-unaligned-store.....	546
malign-stringops.....	544	mavx512bf16.....	528
malloc-cc.....	393	mavx512bitalg.....	528
mallow-string-insns.....	487	mavx512bmm.....	528
mallregs.....	468	mavx512bw.....	528
maltivec.....	470	mavx512cd.....	528
malu32.....	391	mavx512dq.....	528
malways-align.....	441	mavx512f.....	527
malways-save-lp.....	443	mavx512fp16.....	528
mam33.....	437	mavx512ifma.....	528
mam33-2.....	437	mavx512vbmi.....	528
mam34.....	437	mavx512vbmi2.....	528
mamx-avx512.....	532	mavx512vl.....	528
mamx-bf16.....	531	mavx512vnni.....	528
mamx-complex.....	531	mavx512vp2intersect.....	528
mamx-fp16.....	531	mavx512vpopcntdq.....	528

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	mcbbranch-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, 375, 377, 391, 401, 418, 419, 441, 466, 476	mcix.....	388
mbig-endian-data.....	486	mclld.....	537
mbig-switch.....	509	mclldemote.....	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	mcmove.....	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
mcall-prologues.....	365	mconst-align.....	376
		mconst16.....	550

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	mghs.....	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
mfract-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
mfsrra.....	500		

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
mindexed-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	mkernel.....	383
mindirect-branch-register.....	547	mkl.....	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, 374, 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	mlongcall.....	481
mips64r2.....	422		
mips64r3.....	422		

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.....	376
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
mmanual-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-8bicalls.....	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
mmimal-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
mmovcc.....	463	mno-allow-string-insns.....	487

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-avxifma.....	529	mno-const-align.....	376

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.....	373	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-explicit-relocs.....	389, 408, 429, 465	mno-fprnd.....	471

mno-fpu	501, 511	mno-interlink-compressed	422
mno-frame-header-opt	435	mno-interlink-mips16	422
mno-frecipe	410	mno-interrupts	363
mno-friz	483	mno-isel	472
mno-fsca	499	mno-isize	338
mno-fsgsbase	529	mno-istack	378
mno-fsmuld	505	mno-jmp32	391
mno-fsrra	500	mno-jmpext	391
mno-ft32b	396	mno-jsr	488
mno-full-toc	473	mno-jump-tables-in-data-section	510
mno-fused-madd	430, 475, 491	mno-kernel	383
mno-fxsr	530	mno-kl	531
mno-gas	398	mno-knuthdiv	436
mno-gather	547	mno-lam-bh	410
mno-gfni	530	mno-lamcas	410
mno-ginv	427	mno-lasx	407
mno-gnu-as	401	mno-late-ldp-fusion	323
mno-gnu-attribute	478	mno-ld-seq-sa	410
mno-gnu-ld	401	mno-leaf-id-shared-library	373
mno-gomp	445	mno-libfuncs	435
mno-gpopt	428	mno-library-pic	394
mno-gprel-ro	394	mno-linked-fp	394
mno-h	397	mno-liw	437
mno-half-reg-file	329	mno-ll64	335
mno-hard-dfp	471, 488	mno-llsc	425
mno-hardlit	418	mno-load-store-pairs	430
mno-high-registers	379	mno-local-sdata	427
mno-hle	530	mno-lock	338
mno-hreset	531	mno-long-calls	329, 339, 354, 374, 375, 394, 399, 430, 507
mno-htm	472, 490	mno-long-jumps	508
mno-hw-abs	441	mno-long-load-store	399
mno-iamcu	548	mno-longcall	481
mno-icplb	374	mno-longcalls	551
mno-id-shared-library	417	mno-loongson-ext	427
mno-id-shared-library	373	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.....	376	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-cmps	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	377	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	mrdrpid.....	530
mpconfig.....	529	mrdrnd.....	529
mpcrel.....	416, 484	mrddseed.....	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	372
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-cmps	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
mshfp	335	msynci	434
mshfp-compact	335	mtarget-align	550
mshfp-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	mthreads	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	377
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
mvvsx	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	364
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
00	199
01	197
02	198
03	199
ObjC	384
ObjC++	384
Ofast	200
Og	200
optimize	197
0	197
Os	200
oslib	447
output	37
Oz	200

P

p	245, 620
pagezero_size	385
param	316
pass-exit-codes	43
pedantic	3, 102, 575, 791, 1113
pedantic-errors	3, 102, 1113
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, 795, 1111
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, 1101
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, 164, 1102
Wa	275
Wabbreviated-auto-in-template-arg	65
Wabi	106
Wabi-tag	65
Wabsolute-value	148
Waddr-space-convert	364
Waddress	157
Waddress-of-packed-member	158
Waggregate-return	158
Waggressive-loop-optimizations	158
Waligned-new	79
Wall	104, 1104
Walloc-size	136
Walloc-size-larger-than=	136
Walloc-zero	136
Walloca	136
Walloca-larger-than=	137
Wanalyzer-allocation-size	172
Wanalyzer-deref-before-check	172

Wanalyzer-div-by-zero	172
Wanalyzer-double-fclose	173
Wanalyzer-double-free	173
Wanalyzer-exposure-through-output-file	173
Wanalyzer-exposure-through-uninit-copy	173
Wanalyzer-fd-access-mode-mismatch	173
Wanalyzer-fd-double-close	174
Wanalyzer-fd-leak	174
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	175
Wanalyzer-free-of-non-heap	175
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	177
Wanalyzer-mkostemp-redundant-flags	177
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	178
Wanalyzer-possible-null-argument	178
Wanalyzer-possible-null-dereference	178
Wanalyzer-putenv-of-auto-var	179
Wanalyzer-shift-count-negative	179
Wanalyzer-shift-count-overflow	179
Wanalyzer-stale-setjmp-buffer	179
Wanalyzer-symbol-too-complex	172
Wanalyzer-tainted-allocation-size	179
Wanalyzer-tainted-array-index	180
Wanalyzer-tainted-assertion	180
Wanalyzer-tainted-divisor	181
Wanalyzer-tainted-offset	181
Wanalyzer-tainted-size	181
Wanalyzer-throw-of-unexpected-type	181
Wanalyzer-too-complex	172
Wanalyzer-undefined-behavior-ptrdiff	181
Wanalyzer-undefined-behavior-strtok	182
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	183
Wanalyzer-va-list-use-after-va-end	183
Wanalyzer-write-to-const	183
Wanalyzer-write-to-string-literal	183
Warith-conversion	138
Warray-bounds	138

Warray-compare	139	Wcpp	109
Warray-parameter	139	Wctad-maybe-unsupported	66
Wassign-intercept	85	Wctor-dtor-privacy	66
Wattribute-alias	140	Wdangling-else	153
Wattribute-warning	163	Wdangling-pointer	153
Wattributes	158	Wdangling-reference	66
Wauto-profile	137	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	141	Wdelete-incomplete	80
Wbool-operation	141	Wdelete-non-virtual-dtor	67
Wbuiltin-declaration-mismatch	159	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	151	Wdeprecated-enum-enum-conversion	68
Wc++14-compat	150	Wdeprecated-enum-float-conversion	68
Wc++14-extensions	151	Wdeprecated-literal-operator	68
Wc++17-compat	150	Wdeprecated-non-prototype	160
Wc++17-extensions	151	Wdeprecated-openmp	163
Wc++20-compat	150	Wdeprecated-variadic-comma-omission	68
Wc++20-extensions	151	Wdesignated-init	169
Wc++23-extensions	151	Wdisabled-optimization	168
Wc++26-compat	151	Wdiscarded-array-qualifiers	141
Wc++26-extensions	151	Wdiscarded-qualifiers	141
Wc11-c23-compat	150	Wdiv-by-zero	142
Wc11-c2x-compat	150	Wdouble-promotion	109
Wc23-c2y-compat	150	Wduplicate-decl-specifier	109
Wc90-c99-compat	149	Wduplicated-branches	141
Wc99-c11-compat	150	Wduplicated-cond	141
Wcalloc-transposed-args	136	weak_framework	385
Wcannot-profile	137	weak_reference_mismatches	385
Wcast-align	151	Weffc++	73
Wcast-align=strict	151	Welaborated-enum-base	68
Wcast-function-type	152	Wempty-body	155
Wcast-qual	151	Wendif-labels	149, 155
Wcast-user-defined	152	Wenum-compare	155
Wcatch-value	80	Wenum-conversion	155
Wchanges-meaning	108	Wenum-int-mismatch	155
Wchar-subscripts	108	Werror	101
Wclass-conversion	70	Werror=	101
Wclass-memaccess	70	Wexceptions	74
Wclobbered	152	Wexpansion-to-defined	149
Wco-re	392	Wexpose-global-module-tu-local	78
Wcomma-subscript	66	Wexternal-tu-local	78
Wcomment	148	Wextra	105, 161, 164
Wcomments	148	Wextra-semi	80
Wcompare-distinct-pointer-types	153	Wfatal-errors	101
Wcomplain-wrong-lang	152	Wflex-array-member-not-at-end	155
Wconditionally-supported	80	Wfloat-conversion	156
Wconstant-logical-operand	158	Wfloat-equal	144
Wconversion	153	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
Wcoverage-too-many-paths	109	Wformat-overflow	110, 111

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	161
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	147	Wmissing-parameter-name	160
Wglobal-module	81	Wmissing-parameter-type	160
Whardened	115	Wmissing-profile	119
whatsloaded	385	Wmissing-prototypes	161
Wheader-guard	159	Wmissing-requires	161
whyload	385	Wmissing-template-keyword	162
Wif-not-aligned	117	Wmissing-variable-declarations	161
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	142	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	142	Wno-address	157
Wint-in-bool-context	166	Wno-address-of-packed-member	158
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	136
Winvalid-offsetof	81	Wno-alloc-size-larger-than	136
Winvalid-pch	166	Wno-alloc-zero	136
Winvalid-utf8	166	Wno-alloca	136
Wjump-misses-init	155	Wno-alloca-larger-than	137, 138
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	173
Wleading-whitespace=	143	Wno-analyzer-double-free	173
Wliteral-suffix	69	Wno-analyzer-exposure- through-output-file	173
Wlogical-not-parentheses	158	Wno-analyzer-exposure- through-uninit-copy	173
Wlogical-op	158	Wno-analyzer-fd-access-mode-mismatch	173
Wlong-long	166	Wno-analyzer-fd-double-close	174
Wlto-type-mismatch	169	Wno-analyzer-fd-leak	174
Wmain	117	Wno-analyzer-fd-phase-mismatch	174
Wmaybe-musttail-local-addr	113	Wno-analyzer-fd-type-mismatch	174
Wmaybe-uninitialized	129	Wno-analyzer-fd-use-after-close	174
Wmemset-elt-size	157		
Wmemset-transposed-args	157		

Wno-analyzer-fd-use-without-check	174	Wno-bidi-chars	140
Wno-analyzer-file-leak	175	Wno-bool-compare	141
Wno-analyzer-free-of-non-heap	175	Wno-bool-operation	141
Wno-analyzer-imprecise-fp-arithmetic	175	Wno-builtin-declaration-mismatch	159
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	151
Wno-analyzer-mismatching-deallocation	177	Wno-c++14-compat	150
Wno-analyzer-mkostemp-redundant-flags	177	Wno-c++14-extensions	151
Wno-analyzer-mktemp- missing-placeholder	177	Wno-c++17-compat	150
Wno-analyzer-mktemp-of-string-literal	177	Wno-c++17-extensions	151
Wno-analyzer-null-argument	178	Wno-c++20-compat	150
Wno-analyzer-null-dereference	178	Wno-c++20-extensions	151
Wno-analyzer-out-of-bounds	177	Wno-c++23-extensions	151
Wno-analyzer-overlapping-buffers	178	Wno-c++26-compat	151
Wno-analyzer-possible-null-argument	178	Wno-c++26-extensions	151
Wno-analyzer-possible-null-dereference	178	Wno-c11-c23-compat	150
Wno-analyzer-putenv-of-auto-var	179	Wno-c11-c2x-compat	150
Wno-analyzer-shift-count-negative	179	Wno-c23-c2y-compat	150
Wno-analyzer-shift-count-overflow	179	Wno-c90-c99-compat	149
Wno-analyzer-stale-setjmp-buffer	179	Wno-c99-c11-compat	150
Wno-analyzer-symbol-too-complex	172	Wno-calloc-transposed-args	136
Wno-analyzer-tainted-allocation-size	179	Wno-cannot-profile	137
Wno-analyzer-tainted-array-index	180	Wno-cast-align	151
Wno-analyzer-tainted-assertion	180	Wno-cast-align=strict	151
Wno-analyzer-tainted-divisor	181	Wno-cast-function-type	152
Wno-analyzer-tainted-offset	181	Wno-cast-qual	151
Wno-analyzer-tainted-size	181	Wno-cast-user-defined	152
Wno-analyzer-throw-of-unexpected-type	181	Wno-catch-value	80
Wno-analyzer-too-complex	172	Wno-changes-meaning	108
Wno-analyzer-undefined- behavior-pttrdiff	181	Wno-char-subscripts	108
Wno-analyzer-undefined-behavior-strtok	182	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	153
Wno-analyzer-va-list-leak	183	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	158
Wno-analyzer-write-to-string-literal	183	Wno-conversion	153
Wno-arith-conversion	138	Wno-conversion-null	82
Wno-array-bounds	138	Wno-coverage-invalid-line-number	109
Wno-array-compare	139	Wno-coverage-mismatch	108
Wno-array-parameter	139	Wno-coverage-too-many-conditions	108
Wno-assign-intercept	85	Wno-coverage-too-many-paths	109
Wno-attribute-alias	140	Wno-cpp	109
Wno-attribute-warning	163	Wno-ctad-maybe-unsupported	66
Wno-attributes	158	Wno-ctor-dtor-privacy	66
Wno-auto-profile	137	Wno-dangling-else	153
Wno-bad-function-cast	149	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-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	142
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	142
Wno-discarded-qualifiers	141	Wno-int-in-bool-context	166
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-effc++	73	Wno-invalid-offsetof	81
Wno-elaborated-enum-base	68	Wno-invalid-pch	166
Wno-empty-body	155	Wno-invalid-utf8	166
Wno-endif-labels	149, 155	Wno-jump-misses-init	155
Wno-enum-compare	155	Wno-keyword-macro	159
Wno-enum-conversion	155	Wno-larger-than	146
Wno-enum-int-mismatch	155	Wno-literal-suffix	69
Wno-error	101	Wno-logical-not-parentheses	158
Wno-error=	101	Wno-logical-op	158
Wno-exceptions	74	Wno-long-long	166
Wno-expansion-to-defined	149	Wno-lto-type-mismatch	169
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, 164	Wno-maybe-uninitialized	129
Wno-extra-semi	80	Wno-memset-elt-size	157
Wno-fatal-errors	101	Wno-memset-transposed-args	157
Wno-flex-array-member-not-at-end	155	Wno-misleading-indentation	117
Wno-float-conversion	156	Wno-mismatched-dealloc	119
Wno-float-equal	144	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	161
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	161
Wno-frame-larger-than	146, 147	Wno-missing-requires	161
Wno-free-labels	159	Wno-missing-template-keyword	162
Wno-free-nonheap-object	147	Wno-missing-variable-declarations	161
Wno-global-module	81	Wno-misspelled-isr	364
Wno-hardened	115	Wno-multichar	162
Wno-header-guard	159	Wno-multiple-inheritance	76

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	156
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	146
Wno-non-virtual-dtor	71	Wno-shadow=compatible-local	146
Wno-nonnull	113	Wno-shadow=global	146
Wno-nonnull-compare	113	Wno-shadow=local	146
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	160	Wno-sized-deallocation	81
Wno-old-style-definition	160	Wno-sizeof-array-argument	156
Wno-openacc-dims	333	Wno-sizeof-array-div	156
Wno-openacc-parallelism	163	Wno-sizeof-pointer-div	156
Wno-openmp	164	Wno-sizeof-pointer-memaccess	156
Wno-openmp-simd	164	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	164	Wno-strict-null-sentinel	74
Wno-override-init-side-effects	164	Wno-strict-overflow	131
Wno-packed	164	Wno-strict-prototypes	160
Wno-packed-bitfield-compat	164	Wno-strict-selector-match	85
Wno-packed-not-aligned	164	Wno-string-compare	132
Wno-padded	165	Wno-stringop-overflow	132, 133
Wno-parentheses	120	Wno-stringop-overread	134
Wno-pedantic	102	Wno-stringop-truncation	134
Wno-pedantic-ms-format	148	Wno-subobject-linkage	73
Wno-pessimizing-move	71	Wno-suggest-attribute=	135
Wno-placement-new	79	Wno-suggest-attribute=cold	136
Wno-pmf-conversions	75, 1035	Wno-suggest-attribute=const	135
Wno-pointer-arith	148	Wno-suggest-attribute=format	135
Wno-pointer-compare	148	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	165	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-restrict	165	Wno-tautological-compare	143

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	160
Wno-trampolines	143	Wold-style-definition	160
Wno-trigraphs	149	Wopenacc-dims	333
Wno-trivial-auto-var-init	124	Wopenacc-parallelism	163
Wno-tsan	148	Wopenmp	164
Wno-type-limits	148	Wopenmp-simd	164
Wno-undeclared-selector	86	Woverflow	163
Wno-undef	149	Woverlength-strings	168
Wno-unicode	166	Woverloaded-virtual	75
Wno-uninitialized	128	Woverride-init	164
Wno-unknown-pragmas	130	Woverride-init-side-effects	164
Wno-unsafe-loop-optimizations	147	Wp	275
Wno-unsuffixed-float-constants	169	Wpacked	164
Wno-unterminated-string-initialization	139	Wpacked-bitfield-compat	164
Wno-unused	126	Wpacked-not-aligned	164
Wno-unused-but-set-parameter	124	Wpadded	165
Wno-unused-but-set-variable	124	Wparentheses	120
Wno-unused-const-variable	126	Wpedantic	102
Wno-unused-function	125	Wpedantic-ms-format	148
Wno-unused-label	125	Wpessimizing-move	71
Wno-unused-local-typedefs	125	Wplacement-new	79
Wno-unused-macros	149	Wpmf-conversions	75
Wno-unused-parameter	125	Wpointer-arith	148, 793
Wno-unused-result	126	Wpointer-compare	148
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	167	Wprio-ctor-dtor	130
Wno-variadic-macros	167	Wproperty-assign-default	85
Wno-vector-operation-performance	167	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	167	Wredundant-decls	165
Wno-vla-larger-than	167	Wredundant-move	72
Wno-vla-parameter	167	Wredundant-tags	73
Wno-volatile	79	Wregister	71
Wno-volatile-register-var	168	Wreorder	71
Wno-write-strings	152	Wrestrict	165
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	156
Wnon-template-friend	74	Wselector	85
Wnon-virtual-dtor	71	Wself-move	120
Wnonnull	113	Wsequence-point	121
Wnonnull-compare	113	Wsfinae-incomplete	74
Wnonportable-cfstrings	384	Wshadow	145

Wshadow-ivar	146
Wshadow=compatible-local	146
Wshadow=global	146
Wshadow=local	146
Wshift-count-negative	122
Wshift-count-overflow	122
Wshift-negative-value	122
Wshift-overflow	122
Wsign-compare	155
Wsign-conversion	155
Wsign-promo	75
Wsizeof-deallocation	81
Wsizeof-array-argument	156
Wsizeof-array-div	156
Wsizeof-pointer-div	156
Wsizeof-pointer-memaccess	156
Wstack-protector	168
Wstack-usage	147
Wstrict-aliasing	130
Wstrict-flex-arrays	134
Wstrict-null-sentinel	74
Wstrict-overflow	131
Wstrict-prototypes	160
Wstrict-selector-match	85
Wstring-compare	132
Wstringop-overflow	132, 133
Wstringop-overread	134
Wstringop-truncation	134
Wsubobject-linkage	73
Wsuggest-attribute=	135
Wsuggest-attribute=cold	136
Wsuggest-attribute=const	135
Wsuggest-attribute=format	135
Wsuggest-attribute=malloc	135
Wsuggest-attribute=noreturn	135
Wsuggest-attribute=pure	135
Wsuggest-attribute=returns_nonnull	135
Wsuggest-final-methods	82
Wsuggest-final-types	81
Wsuggest-override	82
Wswitch	123
Wswitch-bool	123
Wswitch-default	123
Wswitch-enum	123
Wswitch-outside-range	123
Wswitch-unreachable	123
Wsync-nand	124
Wsystem-headers	142
Wtautological-compare	143
Wtemplate-body	77
Wtemplate-id-ctor	77
Wtemplate-names-tu-local	77
Wtemplates	75
Wterminate	78
Wtraditional	144
Wtraditional-conversion	145
Wtrailing-whitespace	143
Wtrailing-whitespace=	143

Wtrampolines	143
Wtrigraphs	149
Wtrivial-auto-var-init	124
Wtsan	148
Wtype-limits	148
Wundeclared-selector	86
Wundef	149
Wunicode	166
Wuninitialized	128
Wunknown-pragmas	130
Wunsafe-loop-optimizations	147
Wunsuffixed-float-constants	169
Wunterminated-string-initialization	139
Wunused	126
Wunused-but-set-parameter	124
Wunused-but-set-parameter=	124
Wunused-but-set-variable	124
Wunused-but-set-variable=	124
Wunused-const-variable	126
Wunused-function	125
Wunused-label	125
Wunused-local-typedefs	125
Wunused-macros	149
Wunused-parameter	125
Wunused-result	126
Wunused-value	126
Wunused-variable	126
Wuse-after-free	127
Wuseless-cast	128
Wvarargs	167
Wvariadic-macros	167
Wvector-operation-performance	167
Wvexing-parse	78
Wvirtual-inheritance	77
Wvirtual-move-assign	77
Wvla	167
Wvla-larger-than=	167
Wvla-parameter	167
Wvolatile	79
Wvolatile-register-var	168
Wwrite-strings	152
Wxor-used-as-pow	168
Wzero-as-null-pointer-constant	142
Wzero-init-padding-bits=	169
Wzero-length-bounds	142

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	1035
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
btf_decl_tag	601
btf_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, 1037
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, 1037
hotpatch, S/390	685

I

ifunc	613
indirect_branch, x86	700
indirect_return, x86	701
init_priority	1035
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	1036
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("lzcnt"), x86.....	694
target("avxifma"), x86.....	697	target("max-vectorization"), AArch64	651
target("avxneonconvert"), x86.....	697	target("max-vectorization"), RISC-V.....	683
target("avxvnni"), x86.....	697	target("mfcfrf"), 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.....	1036
		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

__builtin_addc.....	809	__builtin_bswap16.....	804
__builtin_addcl.....	809	__builtin_bswap32.....	804
__builtin_addcll.....	809	__builtin_bswap64.....	804
__builtin_addf128_round_to_odd.....	926	__builtin_btf_type_id.....	854
__builtin_alloca.....	810	__builtin_call_with_static_chain.....	814
__builtin_alloca_with_align.....	810	__builtin_cfuged.....	928
__builtin_alloca_with_align_and_max.....	811	__builtin_choose_expr.....	833
__builtin_apply.....	812	__builtin_classify_type.....	840
__builtin_apply_args.....	812	__builtin_clear_padding.....	836
__builtin_arc_aligned.....	843	__builtin_clrslb.....	800
__builtin_arc_brk.....	843	__builtin_clrslbg.....	801
__builtin_arc_core_read.....	843	__builtin_clrslbl.....	800
__builtin_arc_core_write.....	843	__builtin_clrslbl1.....	801
__builtin_arc_divaw.....	843	__builtin_clz.....	800
__builtin_arc_flag.....	843	__builtin_clzg.....	801
__builtin_arc_lr.....	843	__builtin_clzl.....	800
__builtin_arc_mul64.....	843	__builtin_clzll.....	801
__builtin_arc_mulu64.....	844	__builtin_cntlzd.....	928
__builtin_arc_nop.....	844	__builtin_cnttazdm.....	929
__builtin_arc_norm.....	844	__builtin_complex.....	577
__builtin_arc_normw.....	844	__builtin_constant_p.....	834
__builtin_arc_rtie.....	844	__builtin_convertvector.....	819
__builtin_arc_sleep.....	844	__builtin_copysignq.....	1000
__builtin_arc_sr.....	844	__builtin_counted_by_ref.....	835
__builtin_arc_swap.....	844	__builtin_cpu_init.....	919, 1000
__builtin_arc_sw1.....	844	__builtin_cpu_is.....	919, 1001
__builtin_arc_sync.....	844	__builtin_cpu_supports.....	920, 1004
__builtin_arc_trap_s.....	845	__builtin_crc16_data16.....	806
__builtin_arc_unimp_s.....	845	__builtin_crc16_data8.....	806
__builtin_assoc_barrier.....	838	__builtin_crc32_data16.....	806
__builtin_assume_aligned.....	839	__builtin_crc32_data32.....	806
__builtin_avr_cli.....	850	__builtin_crc32_data8.....	806
__builtin_avr_delay_cycles.....	850	__builtin_crc64_data16.....	806
__builtin_avr_flash_segment.....	851	__builtin_crc64_data32.....	806
__builtin_avr_fmuls.....	850	__builtin_crc64_data64.....	806
__builtin_avr_fmuls.....	850	__builtin_crc64_data8.....	806
__builtin_avr_fmulsu.....	850	__builtin_crc8_data8.....	806
__builtin_avr_insert_bits.....	850	__builtin_ctz.....	800
__builtin_avr_mask1.....	851	__builtin_ctzg.....	801
__builtin_avr_nop.....	850	__builtin_ctzl.....	800
__builtin_avr_nops.....	851	__builtin_ctzll.....	801
__builtin_avr_sei.....	850	__builtin_darn.....	927
__builtin_avr_sleep.....	850	__builtin_darn_32.....	927
__builtin_avr_strlen_flash.....	852	__builtin_darn_raw.....	927
__builtin_avr_strlen_flashx.....	852	__builtin_divf128_round_to_odd.....	926
__builtin_avr_strlen_memx.....	852	__builtin_dynamic_object_size.....	828
__builtin_avr_swap.....	850	__builtin_expect.....	837
__builtin_avr_wdr.....	850	__builtin_expect_with_probability.....	837
__builtin_bit_cast.....	836	__builtin_extend_pointer.....	841
__builtin_bitreverse128.....	804	__builtin_extract_return_addr.....	814
__builtin_bitreverse16.....	804	__builtin_fabsq.....	1000
__builtin_bitreverse32.....	804	__builtin_ffs.....	800
__builtin_bitreverse64.....	804	__builtin_ffsg.....	801
__builtin_bitreverse8.....	804	__builtin_ffsl.....	800
__builtin_bpf_load_byte.....	852	__builtin_ffsll.....	800
__builtin_bpf_load_half.....	852	__builtin_FILE.....	839
__builtin_bpf_load_word.....	852	__builtin_fmaf128_round_to_odd.....	927
__builtin_bswap128.....	804	__builtin_fpclassify.....	797

__builtin_frame_address.....	815	__builtin_nand128.....	799
__builtin_frob_return_addr.....	815	__builtin_nand32.....	798
__builtin_FUNCTION.....	839	__builtin_nand64.....	798
__builtin_goacc_parlevel_id.....	841	__builtin_nanf.....	799
__builtin_goacc_parlevel_size.....	841	__builtin_nanfn.....	799
__builtin_has_attribute.....	831	__builtin_nanfnx.....	799
__builtin_huge_val.....	797	__builtin_nanl.....	799
__builtin_huge_valf.....	797	__builtin_nanq.....	1000
__builtin_huge_valfn.....	797	__builtin_nans.....	799
__builtin_huge_valfnx.....	797	__builtin_nansd128.....	799
__builtin_huge_vall.....	797	__builtin_nansd32.....	799
__builtin_huge_valq.....	1000	__builtin_nansd64.....	799
__builtin_ia32_crc32di.....	1014	__builtin_nansf.....	799
__builtin_ia32_crc32hi.....	1014	__builtin_nansfn.....	799
__builtin_ia32_crc32qi.....	1014	__builtin_nansfnx.....	799
__builtin_ia32_crc32si.....	1014	__builtin_nansl.....	799
__builtin_ia32_loadhps.....	1008	__builtin_nansq.....	1000
__builtin_ia32_loadlps.....	1008	__builtin_nds32_isb.....	918
__builtin_ia32_loadss.....	1008	__builtin_nds32_isync.....	918
__builtin_ia32_loadups.....	1008	__builtin_nds32_mfsr.....	918
__builtin_ia32_pause.....	1000	__builtin_nds32_mfusr.....	919
__builtin_ia32_pclmulqdq128.....	1020	__builtin_nds32_mtsr.....	919
__builtin_ia32_storehps.....	1008	__builtin_nds32_mtusr.....	919
__builtin_ia32_storelps.....	1008	__builtin_nds32_setgie_dis.....	919
__builtin_ia32_storeups.....	1008	__builtin_nds32_setgie_en.....	919
__builtin_ia32_vec_ext_v16qi.....	1013	__builtin_non_tx_store.....	994
__builtin_ia32_vec_ext_v2di.....	1013	__builtin_nvptx_brev.....	919
__builtin_ia32_vec_ext_v4sf.....	1013	__builtin_nvptx_brevll.....	919
__builtin_ia32_vec_ext_v4si.....	1013	__builtin_object_size.....	828
__builtin_ia32_vec_set_v16qi.....	1013	__builtin_offsetof.....	786
__builtin_ia32_vec_set_v2di.....	1013	__builtin_operator_delete.....	830
__builtin_ia32_vec_set_v4sf.....	1013	__builtin_operator_new.....	830
__builtin_ia32_vec_set_v4si.....	1013	__builtin_parity.....	800
__builtin_inf.....	798	__builtin_parityg.....	801
__builtin_infd128.....	798	__builtin_parityl.....	800
__builtin_infd32.....	798	__builtin_parityll.....	801
__builtin_infd64.....	798	__builtin_pdepd.....	929
__builtin_inff.....	798	__builtin_pextd.....	929
__builtin_inffn.....	798	__builtin_popcount.....	800, 1014
__builtin_inffnx.....	798	__builtin_popcountg.....	801
__builtin_infl.....	798	__builtin_popcountl.....	800, 1014
__builtin_infq.....	1000	__builtin_popcountll.....	801, 1014
__builtin_is_constant_evaluated.....	835	__builtin_powi.....	800
__builtin_is_virtual_base_of.....	1041	__builtin_powif.....	800
__builtin_iseqsig.....	795	__builtin_powil.....	800
__builtin_isfinite.....	795	__builtin_prefetch.....	840
__builtin_isgreater.....	795	__builtin_preserve_access_index.....	853
__builtin_isgreaterequal.....	795	__builtin_preserve_enum_value.....	854
__builtin_isinf_sign.....	798	__builtin_preserve_field_info.....	853
__builtin_isnormal.....	795	__builtin_preserve_type_info.....	855
__builtin_issignaling.....	799	__builtin_return.....	812
__builtin_isunordered.....	795	__builtin_return_address.....	814
__builtin_LINE.....	839	__builtin_rev_crc16_data16.....	805
__builtin_longjmp.....	812	__builtin_rev_crc16_data8.....	805
__builtin_mul_overflow.....	808	__builtin_rev_crc32_data16.....	805
__builtin_mul_overflow_p.....	808	__builtin_rev_crc32_data32.....	805
__builtin_mulf128_round_to_odd.....	926	__builtin_rev_crc32_data8.....	805
__builtin_nan.....	798	__builtin_rev_crc64_data16.....	805

__builtin_rev_crc64_data32	805	__builtin_riscv_cv_simd_avgu_h	975
__builtin_rev_crc64_data64	805	__builtin_riscv_cv_simd_avgu_sc_b	976
__builtin_rev_crc64_data8	805	__builtin_riscv_cv_simd_avgu_sc_h	976
__builtin_rev_crc8_data8	805	__builtin_riscv_cv_simd_cmpeq_b	985
__builtin_riscv_cv_alu_addN	973	__builtin_riscv_cv_simd_cmpeq_h	985
__builtin_riscv_cv_alu_addRN	973	__builtin_riscv_cv_simd_cmpeq_sc_b	985
__builtin_riscv_cv_alu_adduN	973	__builtin_riscv_cv_simd_cmpeq_sc_h	985
__builtin_riscv_cv_alu_adduRN	973	__builtin_riscv_cv_simd_cmpge_b	986
__builtin_riscv_cv_alu_clip	973	__builtin_riscv_cv_simd_cmpge_h	986
__builtin_riscv_cv_alu_clipu	973	__builtin_riscv_cv_simd_cmpge_sc_b	987
__builtin_riscv_cv_alu_extbs	973	__builtin_riscv_cv_simd_cmpge_sc_h	986
__builtin_riscv_cv_alu_extbz	973	__builtin_riscv_cv_simd_cmpgeu_b	988
__builtin_riscv_cv_alu_exths	972	__builtin_riscv_cv_simd_cmpgeu_h	988
__builtin_riscv_cv_alu_exthz	973	__builtin_riscv_cv_simd_cmpgeu_sc_b	988
__builtin_riscv_cv_alu_max	972	__builtin_riscv_cv_simd_cmpgeu_sc_h	988
__builtin_riscv_cv_alu_maxu	972	__builtin_riscv_cv_simd_cmpgt_b	986
__builtin_riscv_cv_alu_min	972	__builtin_riscv_cv_simd_cmpgt_h	986
__builtin_riscv_cv_alu_minu	972	__builtin_riscv_cv_simd_cmpgt_sc_b	986
__builtin_riscv_cv_alu_slet	972	__builtin_riscv_cv_simd_cmpgt_sc_h	986
__builtin_riscv_cv_alu_sletu	972	__builtin_riscv_cv_simd_cmpgtu_b	988
__builtin_riscv_cv_alu_subN	973	__builtin_riscv_cv_simd_cmpgtu_h	988
__builtin_riscv_cv_alu_subRN	974	__builtin_riscv_cv_simd_cmpgtu_sc_b	988
__builtin_riscv_cv_alu_subuN	973	__builtin_riscv_cv_simd_cmpgtu_sc_h	988
__builtin_riscv_cv_alu_subuRN	974	__builtin_riscv_cv_simd_cmple_b	987
__builtin_riscv_cv_elw_elw	974	__builtin_riscv_cv_simd_cmple_h	987
__builtin_riscv_cv_mac_mac	971	__builtin_riscv_cv_simd_cmple_sc_b	987, 988
__builtin_riscv_cv_mac_machhsN	972	__builtin_riscv_cv_simd_cmple_sc_h	987
__builtin_riscv_cv_mac_machhsRN	972	__builtin_riscv_cv_simd_cmpleu_b	989
__builtin_riscv_cv_mac_machhuN	971	__builtin_riscv_cv_simd_cmpleu_h	989
__builtin_riscv_cv_mac_machhuRN	972	__builtin_riscv_cv_simd_cmpleu_sc_b	989
__builtin_riscv_cv_mac_macsN	971	__builtin_riscv_cv_simd_cmpleu_sc_h	989
__builtin_riscv_cv_mac_macsRN	972	__builtin_riscv_cv_simd_cmplt_b	987
__builtin_riscv_cv_mac_macuN	971	__builtin_riscv_cv_simd_cmplt_h	987
__builtin_riscv_cv_mac_macuRN	972	__builtin_riscv_cv_simd_cmplt_sc_b	987
__builtin_riscv_cv_mac_msu	971	__builtin_riscv_cv_simd_cmplt_sc_h	987
__builtin_riscv_cv_mac_mulhhsN	971	__builtin_riscv_cv_simd_cmpltu_b	989
__builtin_riscv_cv_mac_mulhhsRN	971	__builtin_riscv_cv_simd_cmpltu_h	989
__builtin_riscv_cv_mac_mulhhuN	971	__builtin_riscv_cv_simd_cmpltu_sc_b	989
__builtin_riscv_cv_mac_mulhhuRN	971	__builtin_riscv_cv_simd_cmpltu_sc_h	989
__builtin_riscv_cv_mac_mulsN	971	__builtin_riscv_cv_simd_cmpne_b	985
__builtin_riscv_cv_mac_mulsRN	971	__builtin_riscv_cv_simd_cmpne_h	985
__builtin_riscv_cv_mac_muluN	971	__builtin_riscv_cv_simd_cmpne_sc_b	986
__builtin_riscv_cv_mac_muluRN	971	__builtin_riscv_cv_simd_cmpne_sc_h	985, 986
__builtin_riscv_cv_simd_abs_b	981	__builtin_riscv_cv_simd_cplxconj	990
__builtin_riscv_cv_simd_abs_h	980	__builtin_riscv_cv_simd_cplxmul_i	990
__builtin_riscv_cv_simd_add_b	974	__builtin_riscv_cv_simd_cplxmul_r	989, 990
__builtin_riscv_cv_simd_add_h	974, 990, 991	__builtin_riscv_cv_simd_dotsp_b	982
__builtin_riscv_cv_simd_add_sc_b	974	__builtin_riscv_cv_simd_dotsp_h	982
__builtin_riscv_cv_simd_add_sc_h	974	__builtin_riscv_cv_simd_dotsp_sc_b	982
__builtin_riscv_cv_simd_and_b	980	__builtin_riscv_cv_simd_dotsp_sc_h	982
__builtin_riscv_cv_simd_and_h	980	__builtin_riscv_cv_simd_dotup_b	981
__builtin_riscv_cv_simd_and_sc_b	980	__builtin_riscv_cv_simd_dotup_h	981
__builtin_riscv_cv_simd_and_sc_h	980	__builtin_riscv_cv_simd_dotup_sc_b	981
__builtin_riscv_cv_simd_avg_b	975	__builtin_riscv_cv_simd_dotup_sc_h	981
__builtin_riscv_cv_simd_avg_h	975	__builtin_riscv_cv_simd_dotusp_b	981
__builtin_riscv_cv_simd_avg_sc_b	975	__builtin_riscv_cv_simd_dotusp_h	981
__builtin_riscv_cv_simd_avg_sc_h	975	__builtin_riscv_cv_simd_dotusp_sc_b	981
__builtin_riscv_cv_simd_avgu_b	975	__builtin_riscv_cv_simd_dotusp_sc_h	981

__builtin_riscv_cv_simd_extract_b.....	983	__builtin_riscv_cv_simd_sll_sc_h.....	979
__builtin_riscv_cv_simd_extract_h.....	983	__builtin_riscv_cv_simd_sra_b.....	978
__builtin_riscv_cv_simd_extractu_b.....	984	__builtin_riscv_cv_simd_sra_h.....	978
__builtin_riscv_cv_simd_extractu_h.....	984	__builtin_riscv_cv_simd_sra_sc_b.....	978, 979
__builtin_riscv_cv_simd_insert_b.....	984	__builtin_riscv_cv_simd_sra_sc_h.....	978
__builtin_riscv_cv_simd_insert_h.....	984	__builtin_riscv_cv_simd_srl_b.....	978
__builtin_riscv_cv_simd_max_b.....	977	__builtin_riscv_cv_simd_srl_h.....	978
__builtin_riscv_cv_simd_max_h.....	977	__builtin_riscv_cv_simd_srl_sc_b.....	978
__builtin_riscv_cv_simd_max_sc_b.....	977	__builtin_riscv_cv_simd_srl_sc_h.....	978
__builtin_riscv_cv_simd_max_sc_h.....	977	__builtin_riscv_cv_simd_sub_b.....	975
__builtin_riscv_cv_simd_maxu_b.....	977	__builtin_riscv_cv_simd_sub_h.....	974, 991
__builtin_riscv_cv_simd_maxu_h.....	977	__builtin_riscv_cv_simd_sub_sc_b.....	975
__builtin_riscv_cv_simd_maxu_sc_b.....	978	__builtin_riscv_cv_simd_sub_sc_h.....	975
__builtin_riscv_cv_simd_maxu_sc_h.....	977	__builtin_riscv_cv_simd_subrotmj.....	990
__builtin_riscv_cv_simd_min_b.....	976	__builtin_riscv_cv_simd_xor_b.....	980
__builtin_riscv_cv_simd_min_h.....	976	__builtin_riscv_cv_simd_xor_h.....	980
__builtin_riscv_cv_simd_min_sc_b.....	976	__builtin_riscv_cv_simd_xor_sc_b.....	980
__builtin_riscv_cv_simd_min_sc_h.....	976	__builtin_riscv_cv_simd_xor_sc_h.....	980
__builtin_riscv_cv_simd_minu_b.....	976	__builtin_riscv_pause.....	970
__builtin_riscv_cv_simd_minu_h.....	976	__builtin_rx_brk.....	991
__builtin_riscv_cv_simd_minu_sc_b.....	977	__builtin_rx_clrpsw.....	991
__builtin_riscv_cv_simd_minu_sc_h....	976, 977	__builtin_rx_int.....	991
__builtin_riscv_cv_simd_or_b.....	979	__builtin_rx_machi.....	991
__builtin_riscv_cv_simd_or_h.....	979	__builtin_rx_maclo.....	991
__builtin_riscv_cv_simd_or_sc_b.....	979	__builtin_rx_mulhi.....	991
__builtin_riscv_cv_simd_or_sc_h.....	979	__builtin_rx_mullo.....	991
__builtin_riscv_cv_simd_packhi_b.....	985	__builtin_rx_mvfacchi.....	992
__builtin_riscv_cv_simd_packhi_h.....	985	__builtin_rx_mvfacmi.....	992
__builtin_riscv_cv_simd_packlo_b.....	985	__builtin_rx_mvfc.....	992
__builtin_riscv_cv_simd_packlo_h.....	985	__builtin_rx_mvtachi.....	992
__builtin_riscv_cv_simd_sdotsp_b.....	983	__builtin_rx_mvtaclo.....	992
__builtin_riscv_cv_simd_sdotsp_h.....	983	__builtin_rx_mvtc.....	992
__builtin_riscv_cv_simd_sdotsp_sc_b.....	983	__builtin_rx_mvtipl.....	992
__builtin_riscv_cv_simd_sdotsp_sc_h.....	983	__builtin_rx_racw.....	992
__builtin_riscv_cv_simd_sdotup_b.....	982	__builtin_rx_revw.....	992
__builtin_riscv_cv_simd_sdotup_h.....	982	__builtin_rx_rmpa.....	992
__builtin_riscv_cv_simd_sdotup_sc_b.....	982	__builtin_rx_round.....	992
__builtin_riscv_cv_simd_sdotup_sc_h.....	982	__builtin_rx_sat.....	992
__builtin_riscv_cv_simd_sdotusp_b.....	983	__builtin_rx_setpsw.....	992
__builtin_riscv_cv_simd_sdotusp_h.....	982	__builtin_rx_wait.....	992
__builtin_riscv_cv_simd_sdotusp_sc_b.....	983	__builtin_sadd_overflow.....	807
__builtin_riscv_cv_simd_sdotusp_sc_h.....	983	__builtin_saddl_overflow.....	807
__builtin_riscv_cv_simd_shuffle_b.....	984	__builtin_saddll_overflow.....	807
__builtin_riscv_cv_simd_shuffle_h.....	984	__builtin_set_thread_pointer.....	994
__builtin_riscv_cv_simd_shuffle_sci_h....	984	__builtin_setjmp.....	812
__builtin_riscv_cv_simd_shuffle2_b.....	984	__builtin_sh_get_fpscr.....	995
__builtin_riscv_cv_simd_shuffle2_h.....	984	__builtin_sh_set_fpscr.....	995
__builtin_riscv_cv_simd_shufflei0_sci_b.....	984	__builtin_shuffle.....	818
__builtin_riscv_cv_simd_shufflei1_sci_b.....	984	__builtin_shufflevector.....	819
__builtin_riscv_cv_simd_shufflei2_sci_b.....	984	__builtin_smul_overflow.....	808
__builtin_riscv_cv_simd_shufflei3_sci_b.....	984	__builtin_smull_overflow.....	808
__builtin_riscv_cv_simd_sll_b.....	979	__builtin_smulll_overflow.....	808
__builtin_riscv_cv_simd_sll_h.....	979	__builtin_speculation_safe_value.....	795, 831
__builtin_riscv_cv_simd_sll_sc_b.....	979	__builtin_sqrtf128_round_to_odd.....	926
		__builtin_ssub_overflow.....	807
		__builtin_ssubl_overflow.....	807
		__builtin_ssubll_overflow.....	807
		__builtin_stack_address.....	815

__builtin_stdcall_bit_ceil.....	802	__flash4 AVR Named Address Spaces.....	590
__builtin_stdcall_bit_floor.....	802	__flash5 AVR Named Address Spaces.....	590
__builtin_stdcall_bit_width.....	802	__flashx AVR Named Address Spaces.....	590
__builtin_stdcall_count_ones.....	802	__float128 data type.....	577
__builtin_stdcall_count_zeros.....	802	__float80 data type.....	577
__builtin_stdcall_first_leading_one.....	802	__force_132 Xtensa Named Address Spaces...	592
__builtin_stdcall_first_leading_zero.....	802	__fp16 data type.....	578
__builtin_stdcall_first_trailing_one.....	803	__func__ identifier.....	791
__builtin_stdcall_first_trailing_zero.....	803	__FUNCTION__ identifier.....	791
__builtin_stdcall_has_single_bit.....	803	__halt.....	970
__builtin_stdcall_leading_ones.....	803	__has_nothrow_assign.....	1040
__builtin_stdcall_leading_zeros.....	803	__has_nothrow_constructor.....	1040
__builtin_stdcall_rotate_left.....	803	__has_nothrow_copy.....	1040
__builtin_stdcall_rotate_right.....	804	__has_trivial_assign.....	1040
__builtin_stdcall_trailing_ones.....	803	__has_trivial_constructor.....	1040
__builtin_stdcall_trailing_zeros.....	803	__has_trivial_copy.....	1040
__builtin_structured_binding_size.....	1042	__has_trivial_destructor.....	1041
__builtin_sub_overflow.....	807	__has_virtual_destructor.....	1041
__builtin_sub_overflow_p.....	808	__ibm128 data type.....	577
__builtin_subc.....	809	__imag__ keyword.....	576
__builtin_subcl.....	809	__int128 data types.....	575
__builtin_subcll.....	809	__integer_pack.....	1042
__builtin_subf128_round_to_odd.....	926	__is_abstract.....	1041
__builtin_tabort.....	994	__is_aggregate.....	1041
__builtin_tbegin.....	993	__is_base_of.....	1041
__builtin_tbegin_nofloat.....	993	__is_class.....	1041
__builtin_tbegin_retry.....	994	__is_empty.....	1041
__builtin_tbegin_retry_nofloat.....	994	__is_enum.....	1042
__builtin_tbegininc.....	994	__is_final.....	1042
__builtin_tend.....	994	__is_literal_type.....	1042
__builtin_tgmam.....	834	__is_pod.....	1042
__builtin_thread_pointer.....	970, 995	__is_polymorphic.....	1042
__builtin_trap.....	837	__is_same.....	1042
__builtin_truncf128_round_to_odd.....	927	__is_standard_layout.....	1042
__builtin_tx_assist.....	994	__is_trivial.....	1042
__builtin_tx_nesting_depth.....	994	__is_union.....	1042
__builtin_types_compatible_p.....	832	__lmbd.....	970
__builtin_uadd_overflow.....	807	__memx AVR Named Address Spaces.....	590
__builtin_uaddl_overflow.....	807	__PRETTY_FUNCTION__ identifier.....	791
__builtin_uaddll_overflow.....	807	__real__ keyword.....	576
__builtin_umul_overflow.....	808	__regio_symbol PRU Named	
__builtin_umull_overflow.....	808	Address Spaces.....	592
__builtin_umulll_overflow.....	808	__seg_fs x86 named address space.....	592
__builtin_unreachable.....	837	__seg_gs x86 named address space.....	592
__builtin_usub_overflow.....	807	__STDC_HOSTED__.....	3
__builtin_usubl_overflow.....	807	__sync builtins.....	825
__builtin_usubll_overflow.....	807	__sync_add_and_fetch.....	826
__builtin_va_arg_pack.....	813	__sync_and_and_fetch.....	826
__builtin_va_arg_pack_len.....	813	__sync_bool_compare_and_swap.....	827
__complex__ keyword.....	575	__sync_fetch_and_add.....	826
__declspec.....	672	__sync_fetch_and_and.....	826
__delay_cycles.....	970	__sync_fetch_and_nand.....	826
__extension__.....	791	__sync_fetch_and_or.....	826
__far RL78 Named Address Spaces.....	592	__sync_fetch_and_sub.....	826
__flash AVR Named Address Spaces.....	590	__sync_fetch_and_xor.....	826
__flash1 AVR Named Address Spaces.....	590	__sync_lock_release.....	827
__flash2 AVR Named Address Spaces.....	590	__sync_lock_test_and_set.....	827
__flash3 AVR Named Address Spaces.....	590	__sync_nand_and_fetch.....	827

<code>__sync_or_and_fetch</code>	826
<code>__sync_sub_and_fetch</code>	826
<code>__sync_synchronize</code>	827
<code>__sync_val_compare_and_swap</code>	827
<code>__sync_xor_and_fetch</code>	826
<code>__thread</code>	715
<code>__underlying_type</code>	1042
<code>_Accum</code> data type	580
<code>_Bool</code> keyword	788
<code>_Complex</code> keyword	575
<code>_Countof</code>	786
<code>_Decimal128</code> data type	579
<code>_Decimal32</code> data type	579
<code>_Decimal64</code> data type	579
<code>_Exit</code>	795
<code>_exit</code>	795
<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>	1027
<code>_HTM_FIRST_USER_ABORT_CODE</code>	993
<code>_HTM_TBEGIN_INDETERMINATE</code>	993
<code>_HTM_TBEGIN_PERSISTENT</code>	993
<code>_HTM_TBEGIN_STARTED</code>	993
<code>_HTM_TBEGIN_TRANSIENT</code>	993
<code>_inc_ssp</code>	1027
<code>_Maxof</code>	786
<code>_Minof</code>	786
<code>_Sat</code> data type	580
<code>_xabort</code>	1026
<code>_XABORT_CAPACITY</code>	1026
<code>_XABORT_CONFLICT</code>	1026
<code>_XABORT_DEBUG</code>	1026
<code>_XABORT_EXPLICIT</code>	1026
<code>_XABORT_NESTED</code>	1026
<code>_XABORT_RETRY</code>	1026
<code>_xbegin</code>	1025
<code>_xend</code>	1026
<code>_xtest</code>	1026

O

<code>'0'</code> in constraint	746
--------------------------------------	-----

A

AArch64 code generation attributes	649
AArch64 Options	317
ABI	1061
<code>abort</code>	795
<code>abs</code>	795
accessing volatiles	720, 1029
<code>acos</code>	795
<code>acosf</code>	795
<code>acosh</code>	795
<code>acoshf</code>	795
<code>acoshl</code>	795

<code>acosl</code>	795
<code>acospi</code>	795
<code>acospif</code>	795
<code>acospil</code>	795
Ada	1
additional floating types	577
address constraints	747
address of a label	782
<code>address_operand</code>	747
alignment	787
alignment of allocated data	599
alignment of data	601
alignment of functions and data	597
alignment of structure fields	627, 645
<code>alloca</code>	795
<code>alloca</code> vs variable-length arrays	581
Allow nesting in an interrupt handler on the Blackfin processor	662
alternate keywords	791
AMD GCN Options	331
AMD1	3
ANSI C	3
ANSI C standard	3
ANSI C89	3
ANSI support	45
ANSI X3.159-1989	3
apostrophes	1102
application binary interface	1061
ARC options	333
architecture-specific options	316
ARM [Annotated C++ Reference Manual]	1043
ARM options	341
arrays of length zero	582
arrays of variable length	581
arrays, non-lvalue	586
<code>asin</code>	795
<code>asinf</code>	795
<code>asinh</code>	795
<code>asinhf</code>	795
<code>asinh1</code>	795
<code>asinl</code>	795
<code>asinpi</code>	795
<code>asinpif</code>	795
<code>asinpil</code>	795
asm assembler template	727
asm clobbers	734
asm constraints	744
asm expressions	733
asm flag output operands	731
asm goto labels	737
asm <code>inline</code>	779
asm input operands	733
asm keyword	721
asm output operands	728
asm scratch registers	734
asm volatile	725
assembler names for identifiers	773
assembly code, invalid	1115

assembly language in C	721
assembly language in C, basic	721
assembly language in C, extended	723
atan	795
atan2	795
atan2f	795
atan2l	795
atan2pi	795
atan2pif	795
atan2pil	795
atanf	795
atanh	795
atanhf	795
atanhl	795
atanl	795
atanpi	795
atanpif	795
atanpil	795
atomic memory access builtins	820
attribute syntax	703
attributes	593
autoincrement/decrement addressing	745
automatic inline for C++ member fns	719
AVR Options	359

B

Backwards Compatibility	1043
base class members	1107
base type of an enum	787
basic asm	721
bcmp	795
binary compatibility	1061
Binary constants using the ‘0b’ prefix	790
bit operation builtins	800
Blackfin Options	372
boolean type	788
bound pointer to member function	1034
break handler functions	671
bug criteria	1115
bugs	1115
bugs, known	1099
built-in functions	48
Built-in Functions	795
built-in library functions	795
built-in numeric functions	797
builtins for arithmetic overflow checking	806
builtins for atomic memory access	820
builtins for C++ new and delete operators	830
builtins for stack allocation	810
byte order	630
byte-swapping builtins	804
bzero	795

C

c++	44
C compilation options	9
C intermediate output, nonexistent	1
C language extensions	575
C language, traditional	273
C library function builtins	795
C standard	3
C standard attributes	703
C standards	3
C++	1
C++ comments	790
C++ Compiled Module Interface	561
C++ interface and implementation headers	1031
C++ language extensions	1029
C++ member fns, automatically inline	719
C++ misunderstandings	1106
C++ Module Mapper	559
C++ Module Preprocessing	560
C++ options, command-line	52
C++ pragmas, effect on inlining	1032
C++ source file suffixes	44
C++ standard attributes	703
C++ static data, declaring and defining	1106
C++11 memory model	820
C-SKY Options	377
C_INCLUDE_PATH	554
C11	3
C17	3
C1X	3
C23	3
C2X	3
C2Y	3
C6X Options	375
C89	3
C90	3
C94	3
C95	3
C99	3
C9X	3
cabs	795
cabsf	795
cabsl	795
cacos	795
cacosf	795
cacosh	795
cacoshf	795
cacoshl	795
cacosl	795
calling functions through the function vector on SH2A	686
calloc	795
carg	795
cargf	795
cargl	795
case labels in initializers	588
case ranges	789
casin	795

<code>casinf</code>	795	compiler options, Objective-C and	
<code>casinh</code>	795	Objective-C++	82
<code>casinhf</code>	795	compiler version, specifying	9
<code>casinhl</code>	795	<code>COMPILER_PATH</code>	553
<code>casinl</code>	795	complex conjugation	576
cast to a union	585	complex numbers	575
<code>catan</code>	795	compound literals	587
<code>catanf</code>	795	computed gotos	782
<code>catanh</code>	795	conditional expressions, extensions	789
<code>catanhf</code>	795	conflicting types	1105
<code>catanhl</code>	795	<code>conj</code>	795
<code>catanl</code>	795	<code>conjf</code>	795
<code>cbrt</code>	795	<code>conjl</code>	795
<code>cbrtf</code>	795	<code>const</code> applied to function	794
<code>cbrtl</code>	795	<code>const</code> qualifier	793
<code>ccos</code>	795	constants in constraints	745
<code>ccosf</code>	795	constraint modifier characters	748
<code>ccosh</code>	795	constraint, matching	746
<code>ccoshf</code>	795	constraints, <code>asm</code>	744
<code>ccoshl</code>	795	constraints, machine specific	749
<code>ccosl</code>	795	constructing calls	812
<code>ceil</code>	795	constructor expressions	587
<code>ceilf</code>	795	contributors	1145
<code>ceill</code>	795	<code>copysign</code>	795
<code>cexp</code>	795	<code>copysignf</code>	795
<code>cexpf</code>	795	<code>copysignl</code>	795
<code>cexpl</code>	795	core dump	1115
char type signedness	51	<code>cos</code>	795
character arrays, non-null-terminated	623	<code>cosf</code>	795
character arrays, null-terminated	625	<code>cosh</code>	795
character set, execution	271	<code>coshf</code>	795
character set, input	272	<code>coshl</code>	795
character set, input normalization	162	<code>cosl</code>	795
character set, wide execution	272	<code>cospi</code>	795
<code>cimag</code>	795	<code>cospif</code>	795
<code>cimagf</code>	795	<code>cospil</code>	795
<code>cimagl</code>	795	<code>CPATH</code>	554
cleanup functions	602	<code>CPLUS_INCLUDE_PATH</code>	554
<code>clog</code>	795	<code>cpow</code>	795
<code>clog10</code>	795	<code>cpowf</code>	795
<code>clog10f</code>	795	<code>cpowl</code>	795
<code>clog10l</code>	795	<code>cproj</code>	795
<code>clogf</code>	795	<code>cprojf</code>	795
<code>clogl</code>	795	<code>cprojl</code>	795
COBOL	1, 7	CRC builtins	805
code generation conventions	286	<code>creal</code>	795
code, mixed with declarations	789	<code>crealf</code>	795
cold functions and code regions	602	<code>creall</code>	795
command options	9	CRIS Options	375
comments, C++ style	790	cross compiling	9
common storage	603	<code>csin</code>	795
comparison of signed and unsigned		<code>csinf</code>	795
values, warning	155	<code>csinh</code>	795
compilation statistics	297	<code>csinhf</code>	795
compiler bugs, reporting	1115	<code>csinhl</code>	795
compiler compared to C++ preprocessor	1	<code>csinl</code>	795
compiler options, C++	52	<code>csqrt</code>	795
		<code>csqrtf</code>	795

<code>csqrtl</code>	795
<code>ctan</code>	795
<code>ctanf</code>	795
<code>ctanh</code>	795
<code>ctanhf</code>	795
<code>ctanhl</code>	795
<code>ctanl</code>	795
<code>CXX_MODULE_MAPPER</code> environment variable	61
Cygwin and MinGW Options	380

D

<code>d</code> floating point suffix	579
<code>D</code>	1
<code>D</code> floating point suffix	579
Darwin options	381
<code>dcgettext</code>	795
<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	1102
declarations inside expressions	779
declarations, mixed with code	789
declaring attributes	593
declaring attributes of functions	593
declaring static data in C++	1106
defining static data in C++	1106
dependencies for make as output	554, 555
dependencies, <code>make</code>	268
<code>DEPENDENCIES_OUTPUT</code>	555
dependent name lookup	1107
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>	795
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	790
double-word arithmetic	575
downward funargs	783

<code>drem</code>	795
<code>dremf</code>	795
<code>dreml</code>	795
dump options	297

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	787
environment variables	552
<code>erf</code>	795
<code>erfc</code>	795
<code>erfcf</code>	795
<code>erfcl</code>	795
<code>erff</code>	795
<code>erfl</code>	795
error GCC_COLORS capability	89
error messages	1113
escaped newlines	790
exception handler functions, Blackfin	661
exception handler functions, NDS32	678
<code>exit</code>	795
<code>exp</code>	795
<code>exp10</code>	795
<code>exp10f</code>	795
<code>exp10l</code>	795
<code>exp2</code>	795
<code>exp2f</code>	795
<code>exp2l</code>	795
<code>EXPERIMENTAL_SARIF_SOCKET</code>	554
<code>expf</code>	795
<code>expl</code>	795
explicit register variables	774
<code>expm1</code>	795
<code>expm1f</code>	795
<code>expm1l</code>	795
expressions containing statements	779
expressions, constructor	587
extended <code>asm</code>	723
extensible constraints	747
extensions, <code>?:</code>	789
extensions, C language	575
extensions, C++ language	1029
external declaration scope	1102
extra NOP instructions at the function entry point	628

F

‘F’ in constraint 746
fabs 795
fabsf 795
fabsl 795
 fallthrough in switch statements 608
 fatal signal 1115
fdim 795
fdimf 795
fdiml 795
 FDL, GNU Free Documentation License 1137
ffs 795
 file name suffix 34
 file names 276
 fixed-point types 580
fixit-delete GCC_COLORS capability 89
fixit-insert GCC_COLORS capability 89
flag_enum type attribute 609
 flexible array members 582, 605, 632
 float as function value type 1103
 floating point precision 1105
 floating-point format builtins 797
floor 795
floorf 795
floorl 795
fma 795
fmaf 795
fmal 795
fmax 795
fmaxf 795
fmaxl 795
fmin 795
fminf 795
fminl 795
fmod 795
fmodf 795
fmodl 795
fnname GCC_COLORS capability 89
 Fortran 1
 forward declaration of an **enum** 787
 forwarding calls 812
fprintf 795
fprintf_unlocked 795
fputs 795
fputs_unlocked 795
 FR30 Options 393
free 795
 freestanding environment 3
 freestanding implementation 3
frexp 795
frexpf 795
frexpl 795
 FRV Options 393
fscanf 795
fscanf, and constant strings 1101
 FT32 Options 396
 function addressability on the M32R/D 670
 function attributes 593

function inlining 600, 610, 612
 function inlining, suppressing 621
 function instrumentation 620
 function multiversioning 639
 function pointers, arithmetic 793
 function prototype declarations 792
 function versions 1038
 function, size of pointer to 793
 functions in arbitrary sections 631
 functions that are called on
 program startup or exit 604
 functions that are compiled with
 specific optimizations 626
 functions that are dynamically resolved 613
 functions that are expected to
 throw exceptions 608
 functions that are passed arguments in
 registers on x86-32 689
 functions that are supposed to be
 optimized away 608
 functions that behave like **malloc** 616
 functions that do not have
 prologue/epilogue code 619
 functions that do not throw exceptions 625
 functions that handle interrupts 615
 functions that have a
 null-terminated argument list 631
 functions that have format
 string arguments 610, 611
 functions that have no side effects... 603, 628, 629,
 641
 functions that have non-null
 pointer arguments 622, 623
 functions that have patchable entries 628
 functions that have SIMD clones 632
 functions that have tainted arguments 638
 functions that never return 624
 functions that pop the argument
 stack on x86-32 688, 690
 functions that restrict access to
 pointer arguments 595
 functions that return more than once 630
 functions that return via tail call 618
 functions that should have stack protection... 632
 functions that should not be instrumented... 620
 functions that should not be sanitized 620, 621
 functions that should not have
 stack limit checking 621
 functions that should not have
 stack protection 621
 functions that take file descriptor arguments .. 609
 functions whose return value must be used 646
 functions with **printf**, **scanf**, **strftime** or
 strfmon style arguments 610

G

'g' in constraint	746
g++	44
'G' in constraint	746
G++	1
gamma	795
gamma_r	795
gammaf	795
gammaf_r	795
gammal	795
gammal_r	795
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	795
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	782
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	790
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	797
hypot	795
hypotf	795
hypotl	795

I

'i' in constraint	745
'I' in constraint	746
IA-64 Options	401
IA64 atomic memory access builtins	825
IBM RS/6000 and PowerPC Options	469
identifier names, dollar signs in	790
identifiers, names in assembler code	773
ilogb	795
ilogbf	795
ilogbl	795
imaxabs	795
implementation-defined	
behavior, C language	563
implementation-defined behavior,	
C++ language	573
implied <code>#pragma implementation</code>	1032
incompatibilities of GCC	1101
incomplete <code>enum</code> types	787
increment operators	1115
index	795
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	797
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	1032
inlining, suppressing	621
installation trouble	1099
instrumentation options	245
integer arithmetic overflow checking builtins	806
integrating function code	718
interface and implementation headers, C++	1031
intermediate C version, nonexistent	1
interrupt handlers	615
invalid assembly code	1115
invalid GCC_COLORS capability	90
invalid input	1115
invoking g++	44
isalnum	795
isalpha	795
isascii	795
isblank	795
isctrl	795
isdigit	795
isgraph	795
islower	795

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	795
ispunct	795
isspace	795
isupper	795
iswalnum	795
iswalpha	795
iswblank	795
iswcntrl	795
iswdigit	795
iswgraph	795
iswlower	795
iswprint	795
iswpunct	795
iswspace	795
iswupper	795
iswxdigit	795
isxdigit	795

J

j0	795
j0f	795
j0l	795
j1	795
j1f	795
j1l	795
jn	795
jnf	795
jnl	795

K

k fixed-suffix	580
K fixed-suffix	580
keywords, alternate	791
known causes of trouble	1099

L

labeled elements in initializers	588
labels as values	782
labs	795
language dialect options	45
LANG	552
LC_ALL	552
LC_CTYPE	552
LC_MESSAGES	552
ldexp	795
ldexpf	795
ldexpl	795
legacy builtins for atomic memory access	825
length-zero arrays	582
lgamma	795
lgamma_r	795
lgammaf	795
lgammaf_r	795
lgammal	795
lgammal_r	795
Libraries	277
library function builtins	795
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	795
llk fixed-suffix	580
LLK fixed-suffix	580
llr fixed-suffix	580
LLR fixed-suffix	580
llrint	795
llrintf	795
llrintl	795
llround	795
llroundf	795
llroundl	795
LM32 options	404
load address instruction	747
local labels	781
local variables in macros	785
local variables, specifying registers	775
locale	552
locus GCC_COLORS capability	89
log	795
log10	795
log10f	795
log10l	795
log1p	795
log1pf	795
log1pl	795
log2	795
log2f	795
log2l	795

<code>logb</code>	795
<code>logbf</code>	795
<code>logbl</code>	795
<code>logf</code>	795
<code>logl</code>	795
<code>long long</code> data types.....	575
<code>longjmp</code>	775
<code>longjmp</code> incompatibilities.....	1101
<code>longjmp</code> warnings.....	129
LoongArch Options.....	405
lr fixed-suffix.....	580
LR fixed-suffix.....	580
<code>lrint</code>	795
<code>lrintf</code>	795
<code>lrintl</code>	795
<code>lround</code>	795
<code>lroundf</code>	795
<code>lroundl</code>	795
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.....	788
macros, inline alternative.....	718
macros, local labels.....	781
macros, local variables in.....	785
macros, statements in expressions.....	779
macros, types of arguments.....	784
<code>make</code>	268
<code>malloc</code>	795
matching constraint.....	746
MCORE options.....	418
member fns, automatically <code>inline</code>	719
<code>memchr</code>	795
<code>memcmp</code>	795
<code>memcpy</code>	795
memory references in constraints.....	745
<code>mempcpy</code>	795
<code>memset</code>	795
Mercury.....	1
message formatting.....	88
messages, warning.....	101
messages, warning and error.....	1113
MicroBlaze Options.....	419
Microsoft struct layout.....	710
middle-operands, omitted.....	789
MIPS options.....	420
misunderstandings in C++.....	1106
mixed declarations and code.....	789
mixing assembly language and C.....	721
<code>mktemp</code> , and constant strings.....	1101
MMIX Options.....	435
MN10300 options.....	437

<code>modf</code>	795
<code>modff</code>	795
<code>modfl</code>	795
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..	1032
NaN builtins.....	797
NDS32 Options.....	441
<code>nearbyint</code>	795
<code>nearbyintf</code>	795
<code>nearbyintl</code>	795
nested functions.....	783
<code>new</code> and <code>delete</code> builtins.....	830
newlines (escaped).....	790
<code>nextafter</code>	795
<code>nextafterf</code>	795
<code>nextafterl</code>	795
<code>nexttoward</code>	795
<code>nexttowardf</code>	795
<code>nexttowardl</code>	795
NFC.....	162
NFKC.....	162
NMI handler functions on the Blackfin processor.....	662
<code>no_profile_instrument_function</code> function attribute.....	620
<code>no_sanitiz</code> function attribute.....	620
<code>no_sanitiz</code> _coverage function attribute.....	620
non-constant initializers.....	586
non-static inline function.....	719
nonlocal gotos.....	811
nonstring character arrays.....	623
note GCC_COLORS capability.....	89
<code>number of elements</code>	786
numeric builtins.....	797
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	792
omitted middle-operands	789
opaque <code>enum</code> types	787
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	197
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	197
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	1112
order of options	9
other register constraints	747
output file option	37
overflow checking builtins	806
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	1034
pointer aliasing	617
pointer arguments	604
pointer arguments in variadic functions	793
pointer to member function	1034
pointers to arrays	793
portions of temporary objects, pointers to	1108
pow	795
pow10	795
pow10f	795
pow10l	795
PowerPC options	448
powf	795
powl	795
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	1032
pragmas, interface and implementation	1031
pragmas, warning of unknown	130
precompiled headers	555

preprocessing numbers	1103
preprocessing tokens	1103
preprocessor options	267
printf	795
printf_unlocked	795
prof	245
profiling options	245
program instrumentation options	245
program sections	631
promotion of formal parameters	792
PRU Options	448
push address instruction	747
putchar	795
puts	795

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	789
raw string literals	791
read-only strings	1101
realloc	795
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	795
remainderf	795
remainderl	795
remquo	795
remquof	795
remquol	795
reordering, warning	71
reporting bugs	1115
reset handler functions	678
rest argument (in macro)	788
restricted pointers	1029
restricted references	1029
restricted this pointer	1029
rindex	795

rint	795
rintf	795
rintl	795
RISC-V Options	450
RL78 Options	467
round	795
roundf	795
roundl	795
RS/6000 and PowerPC Options	469
RTTI	1030
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	795
scalbf	795
scalbl	795
scalbln	795
scalblnf	795
scalbn	795
scalbnf	795
scanf , and constant strings	1101
scanfnl	795
scope of a variable length array	581
scope of declaration	1105
scope of external declarations	1102
search path	282
setjmp	775
setjmp incompatibilities	1101
shared strings	1101
side effect in ?:	789
side effects, macro argument	779
side effects, order of evaluation	1112
signbit	795
signbitd128	795
signbitd32	795
signbitd64	795
signbitf	795
signbitl	795
signed and unsigned values, comparison warning	155
signedness of char type	51
significand	795
significandf	795
significandl	795
SIMD	87
SIMD function cloning	632
simple constraints	745
sin	795
sincos	795

<code>sincosf</code>	795
<code>sincosl</code>	795
<code>sinf</code>	795
<code>sinh</code>	795
<code>sinhf</code>	795
<code>sinhl</code>	795
<code>sinl</code>	795
<code>sinpi</code>	795
<code>sinpif</code>	795
<code>sinpil</code>	795
size of objects	599
<code>sizeof</code>	784
smaller data references	412
smaller data references (PowerPC)	481
<code>snprintf</code>	795
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>	795
<code>sqrt</code>	795
<code>sqrtf</code>	795
<code>sqrtl</code>	795
<code>sscanf</code>	795
<code>sscanf</code> , and constant strings	1101
stack allocation builtins	810
stack protection	621, 632
stack scrubbing	633
standard attribute syntax	703
statements inside expressions	779
static data in C++, declaring and defining	1106
<code>stpcpy</code>	795
<code>stpncpy</code>	795
<code>strcasemp</code>	795
<code>strcat</code>	795
<code>strchr</code>	795
<code>strcmp</code>	795
<code>strcpy</code>	795
<code>strcspn</code>	795
<code>strdup</code>	795
<code>strfmon</code>	795
<code>strftime</code>	795
string constants	1101
string literals, raw	791
<code>strlen</code>	795
<code>strncasemp</code>	795
<code>strncat</code>	795
<code>strncmp</code>	795
<code>strncpy</code>	795
<code>strndup</code>	795
<code>strnlen</code>	795
<code>strpbrk</code>	795
<code>strrchr</code>	795
<code>strspn</code>	795

<code>strstr</code>	795
strub eligibility and viability	636
<code>struct</code>	584
<code>struct __htm_tdb</code>	993
structures	1103
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++	1106
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>	795
<code>tanf</code>	795
<code>tanh</code>	795
<code>tanhf</code>	795
<code>tanh1</code>	795
<code>tanl</code>	795
<code>tanpi</code>	795
<code>tanpif</code>	795
<code>tanpil</code>	795
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	1032
temporaries, lifetime of	1108
tentative definitions	289
<code>TERM_URLS</code> environment variable	90
<code>tgamma</code>	795
<code>tgammaf</code>	795
<code>tgamma1</code>	795
Thread-Local Storage	715
thunks	783
tiny data section on the H8/300H and H8S	665
TLS	715
<code>TMPDIR</code>	552
<code>toascii</code>	795

<code>tolower</code>	795
<code>toupper</code>	795
<code>towlower</code>	795
<code>toupper</code>	795
traditional C language	273
transactional memory extensions for x86	822
transparent unions	640
<code>trivial_abi</code> type attribute	1037
<code>trunc</code>	795
<code>truncf</code>	795
<code>trunc1</code>	795
two-stage name lookup	1107
type alignment	787
type-diff GCC_COLORS capability	90
<code>type_info</code>	1030
typedef names as function parameters	1102
<code>typeof</code>	784

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	1115
undefined function value	1115
underlying type of an <code>enum</code>	787
underscores in variables in macros	785
<code>union</code>	584
union, casting to a	585
unions	1103
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	1030
valid GCC_COLORS capability	90
value after <code>longjmp</code>	775
variable addressability on the M32R/D	670
variable alignment	787
variable number of arguments	788
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	785
variadic function sentinel	631
variadic functions, pointer arguments	793
variadic macros	788
VAX options	510
<code>vec_blendv</code>	956
<code>vec_cfuge</code>	951
<code>vec_clrl</code>	952
<code>vec_clrr</code>	952
<code>vec_cntlzm</code>	951
<code>vec_cnttzm</code>	952
<code>vec_extracth</code>	953
<code>vec_extractl</code>	952
<code>vec_genpcvm</code>	959
<code>vec_gnb</code>	952
<code>vec_inserth</code>	954
<code>vec_insertl</code>	953
<code>vec_pdep</code>	953
<code>vec_permx</code>	957
<code>vec_pext</code>	957
<code>vec_replace_element</code>	954
<code>vec_replace_unaligned</code>	955
<code>vec_sldb</code>	955
<code>vec_splati</code>	956
<code>vec_splati_ins</code>	956
<code>vec_splatid</code>	956
<code>vec_srdb</code>	956
<code>vec_stril</code>	957
<code>vec_stril_p</code>	957
<code>vec_strir</code>	957
<code>vec_strir_p</code>	958
<code>vec_ternarylogic</code>	958
<code>vec_test_lsbb_all_ones</code>	951
<code>vec_test_lsbb_all_zeros</code>	951
<code>vec_xst_trunc</code>	929, 930
vector types	642
vector types, using with x86 intrinsics	819
<code>vfprintf</code>	795
<code>vfscanf</code>	795
visibility of symbols	643
Visium options	511
VLAs	581
vold pointers, arithmetic	793

void, size of pointer to 793
 volatile access 720, 1029
 volatile applied to function 794
 volatile **asm** 725
 volatile read 720, 1029
 volatile write 720, 1029
vprintf 795
vscanf 795
vsnprintf 795
vsprintf 795
vsscanf 795
vsx_xl_sext 929
vsx_xl_zext 929
 vtable 1030
 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 1113
 weak symbols 646

whitespace 1102
 Windows Options for x86 549

X

x86 named address spaces 592
 x86 Options 512
 x86 transactional memory extensions 822
 x86 Windows Options 549
 'X' in constraint 746
 X3.159-1989 3
 Xstormy16 Options 549
 Xtensa Options 549

Y

y0 795
 y0f 795
 y0l 795
 y1 795
 y1f 795
 y1l 795
 yn 795
 ynf 795
 ynl 795

Z

zero-length arrays 582
 zero-size structures 584
 zSeries options 552