

It is desirable to exclude internal labels from the symbol table of the object file. Most assemblers have a naming convention for labels that should be excluded; on many systems, the letter ‘L’ at the beginning of a label has this effect. You should find out what convention your system uses, and follow it.

The default version of this function utilizes `ASM_GENERATE_INTERNAL_LABEL`.

ASM_OUTPUT_DEBUG_LABEL (*stream*, *prefix*, *num*) [Macro]

A C statement to output to the stdio stream *stream* a debug info label whose name is made from the string *prefix* and the number *num*. This is useful for VLIW targets, where debug info labels may need to be treated differently than branch target labels. On some systems, branch target labels must be at the beginning of instruction bundles, but debug info labels can occur in the middle of instruction bundles.

If this macro is not defined, then `(*targetm.asm_out.internal_label)` will be used.

ASM_GENERATE_INTERNAL_LABEL (*string*, *prefix*, *num*) [Macro]

A C statement to store into the string *string* a label whose name is made from the string *prefix* and the number *num*.

This string, when output subsequently by `assemble_name`, should produce the output that `(*targetm.asm_out.internal_label)` would produce with the same *prefix* and *num*.

If the string begins with ‘*’, then `assemble_name` will output the rest of the string unchanged. It is often convenient for `ASM_GENERATE_INTERNAL_LABEL` to use ‘*’ in this way. If the string doesn’t start with ‘*’, then `ASM_OUTPUT_LABELREF` gets to output the string, and may change it. (Of course, `ASM_OUTPUT_LABELREF` is also part of your machine description, so you should know what it does on your machine.)

ASM_FORMAT_PRIVATE_NAME (*outvar*, *name*, *number*) [Macro]

A C expression to assign to *outvar* (which is a variable of type `char *`) a newly allocated string made from the string *name* and the number *number*, with some suitable punctuation added. Use `alloca` to get space for the string.

The string will be used as an argument to `ASM_OUTPUT_LABELREF` to produce an assembler label for an internal static variable whose name is *name*. Therefore, the string must be such as to result in valid assembler code. The argument *number* is different each time this macro is executed; it prevents conflicts between similarly-named internal static variables in different scopes.

Ideally this string should not be a valid C identifier, to prevent any conflict with the user’s own symbols. Most assemblers allow periods or percent signs in assembler symbols; putting at least one of these between the name and the number will suffice.

If this macro is not defined, a default definition will be provided which is correct for most systems.

ASM_OUTPUT_DEF (*stream*, *name*, *value*) [Macro]

A C statement to output to the stdio stream *stream* assembler code which defines (equates) the symbol *name* to have the value *value*.

If `SET_ASM_OP` is defined, a default definition is provided which is correct for most systems.

ASM_OUTPUT_DEF_FROM_DECLS (*stream*, *decl_of_name*, *decl_of_value*) [Macro]

A C statement to output to the stdio stream *stream* assembler code which defines (equates) the symbol whose tree node is *decl_of_name* to have the value of the tree node *decl_of_value*. This macro will be used in preference to ‘ASM_OUTPUT_DEF’ if it is defined and if the tree nodes are available.

If SET_ASM_OP is defined, a default definition is provided which is correct for most systems.

TARGET_DEFERRED_OUTPUT_DEFS (*decl_of_name*, *decl_of_value*) [Macro]

A C statement that evaluates to true if the assembler code which defines (equates) the symbol whose tree node is *decl_of_name* to have the value of the tree node *decl_of_value* should be emitted near the end of the current compilation unit. The default is to not defer output of defines. This macro affects defines output by ‘ASM_OUTPUT_DEF’ and ‘ASM_OUTPUT_DEF_FROM_DECLS’.

ASM_OUTPUT_WEAK_ALIAS (*stream*, *name*, *value*) [Macro]

A C statement to output to the stdio stream *stream* assembler code which defines (equates) the weak symbol *name* to have the value *value*. If *value* is NULL, it defines *name* as an undefined weak symbol.

Define this macro if the target only supports weak aliases; define ASM_OUTPUT_DEF instead if possible.

OBJC_GEN_METHOD_LABEL (*buf*, *is_inst*, *class_name*, *cat_name*, *sel_name*) [Macro]

Define this macro to override the default assembler names used for Objective-C methods.

The default name is a unique method number followed by the name of the class (e.g. ‘_1_Foo’). For methods in categories, the name of the category is also included in the assembler name (e.g. ‘_1_Foo_Bar’).

These names are safe on most systems, but make debugging difficult since the method’s selector is not present in the name. Therefore, particular systems define other ways of computing names.

buf is an expression of type `char *` which gives you a buffer in which to store the name; its length is as long as *class_name*, *cat_name* and *sel_name* put together, plus 50 characters extra.

The argument *is_inst* specifies whether the method is an instance method or a class method; *class_name* is the name of the class; *cat_name* is the name of the category (or NULL if the method is not in a category); and *sel_name* is the name of the selector.

On systems where the assembler can handle quoted names, you can use this macro to provide more human-readable names.

17.22.5 How Initialization Functions Are Handled

The compiled code for certain languages includes *constructors* (also called *initialization routines*)—functions to initialize data in the program when the program is started. These

functions need to be called before the program is “started”—that is to say, before `main` is called.

Compiling some languages generates *destructors* (also called *termination routines*) that should be called when the program terminates.

To make the initialization and termination functions work, the compiler must output something in the assembler code to cause those functions to be called at the appropriate time. When you port the compiler to a new system, you need to specify how to do this.

There are two major ways that GCC currently supports the execution of initialization and termination functions. Each way has two variants. Much of the structure is common to all four variations.

The linker must build two lists of these functions—a list of initialization functions, called `__CTOR_LIST__`, and a list of termination functions, called `__DTOR_LIST__`.

Each list always begins with an ignored function pointer (which may hold 0, `-1`, or a count of the function pointers after it, depending on the environment). This is followed by a series of zero or more function pointers to constructors (or destructors), followed by a function pointer containing zero.

Depending on the operating system and its executable file format, either `crtstuff.c` or `libgcc2.c` traverses these lists at startup time and exit time. Constructors are called in reverse order of the list; destructors in forward order.

The best way to handle static constructors works only for object file formats which provide arbitrarily-named sections. A section is set aside for a list of constructors, and another for a list of destructors. Traditionally these are called `‘.ctors’` and `‘.dtors’`. Each object file that defines an initialization function also puts a word in the constructor section to point to that function. The linker accumulates all these words into one contiguous `‘.ctors’` section. Termination functions are handled similarly.

This method will be chosen as the default by `target-def.h` if `TARGET_ASM_NAMED_SECTION` is defined. A target that does not support arbitrary sections, but does support special designated constructor and destructor sections may define `CTORS_SECTION_ASM_OP` and `DTORS_SECTION_ASM_OP` to achieve the same effect.

When arbitrary sections are available, there are two variants, depending upon how the code in `crtstuff.c` is called. On systems that support a `.init` section which is executed at program startup, parts of `crtstuff.c` are compiled into that section. The program is linked by the `gcc` driver like this:

```
ld -o output_file crti.o crtbegin.o ... -lgcc crtend.o crtn.o
```

The prologue of a function (`__init`) appears in the `.init` section of `crti.o`; the epilogue appears in `crtn.o`. Likewise for the function `__fini` in the `.fini` section. Normally these files are provided by the operating system or by the GNU C library, but are provided by GCC for a few targets.

The objects `crtbegin.o` and `crtend.o` are (for most targets) compiled from `crtstuff.c`. They contain, among other things, code fragments within the `.init` and `.fini` sections that branch to routines in the `.text` section. The linker will pull all parts of a section together, which results in a complete `__init` function that invokes the routines we need at startup.

To use this variant, you must define the `INIT_SECTION_ASM_OP` macro properly.

If no init section is available, when GCC compiles any function called `main` (or more accurately, any function designated as a program entry point by the language front end calling `expand_main_function`), it inserts a procedure call to `__main` as the first executable code after the function prologue. The `__main` function is defined in `libgcc2.c` and runs the global constructors.

In file formats that don't support arbitrary sections, there are again two variants. In the simplest variant, the GNU linker (GNU `ld`) and an 'a.out' format must be used. In this case, `TARGET_ASM_CONSTRUCTOR` is defined to produce a `.stabs` entry of type 'N_SETT', referencing the name `__CTOR_LIST__`, and with the address of the void function containing the initialization code as its value. The GNU linker recognizes this as a request to add the value to a *set*; the values are accumulated, and are eventually placed in the executable as a vector in the format described above, with a leading (ignored) count and a trailing zero element. `TARGET_ASM_DESTRUCTOR` is handled similarly. Since no init section is available, the absence of `INIT_SECTION_ASM_OP` causes the compilation of `main` to call `__main` as above, starting the initialization process.

The last variant uses neither arbitrary sections nor the GNU linker. This is preferable when you want to do dynamic linking and when using file formats which the GNU linker does not support, such as 'ECOFF'. In this case, `TARGET_HAVE_CTORS_DTORS` is false, initialization and termination functions are recognized simply by their names. This requires an extra program in the linkage step, called `collect2`. This program pretends to be the linker, for use with GCC; it does its job by running the ordinary linker, but also arranges to include the vectors of initialization and termination functions. These functions are called via `__main` as described above. In order to use this method, `use_collect2` must be defined in the target in `config.gcc`.

17.22.6 Macros Controlling Initialization Routines

Here are the macros that control how the compiler handles initialization and termination functions:

`INIT_SECTION_ASM_OP` [Macro]

If defined, a C string constant, including spacing, for the assembler operation to identify the following data as initialization code. If not defined, GCC will assume such a section does not exist. When you are using special sections for initialization and termination functions, this macro also controls how `crtstuff.c` and `libgcc2.c` arrange to run the initialization functions.

`HAS_INIT_SECTION` [Macro]

If defined, `main` will not call `__main` as described above. This macro should be defined for systems that control start-up code on a symbol-by-symbol basis, such as OSF/1, and should not be defined explicitly for systems that support `INIT_SECTION_ASM_OP`.

`LD_INIT_SWITCH` [Macro]

If defined, a C string constant for a switch that tells the linker that the following symbol is an initialization routine.

`LD_FINI_SWITCH` [Macro]

If defined, a C string constant for a switch that tells the linker that the following symbol is a finalization routine.

COLLECT_SHARED_INIT_FUNC (*stream, func*) [Macro]

If defined, a C statement that will write a function that can be automatically called when a shared library is loaded. The function should call *func*, which takes no arguments. If not defined, and the object format requires an explicit initialization function, then a function called `_GLOBAL__DI` will be generated.

This function and the following one are used by `collect2` when linking a shared library that needs constructors or destructors, or has DWARF2 exception tables embedded in the code.

COLLECT_SHARED_FINI_FUNC (*stream, func*) [Macro]

If defined, a C statement that will write a function that can be automatically called when a shared library is unloaded. The function should call *func*, which takes no arguments. If not defined, and the object format requires an explicit finalization function, then a function called `_GLOBAL__DD` will be generated.

INVOKE__main [Macro]

If defined, `main` will call `__main` despite the presence of `INIT_SECTION_ASM_OP`. This macro should be defined for systems where the init section is not actually run automatically, but is still useful for collecting the lists of constructors and destructors.

SUPPORTS_INIT_PRIORITY [Macro]

If nonzero, the C++ `init_priority` attribute is supported and the compiler should emit instructions to control the order of initialization of objects. If zero, the compiler will issue an error message upon encountering an `init_priority` attribute.

bool TARGET_HAVE_CTORS_DTORS [Target Hook]

This value is true if the target supports some “native” method of collecting constructors and destructors to be run at startup and exit. It is false if we must use `collect2`.

bool TARGET_DTORS_FROM_CXA_ATEXIT [Target Hook]

This value is true if the target wants destructors to be queued to be run from `__cxa_atexit`. If this is the case then, for each priority level, a new constructor will be entered that registers the destructors for that level with `__cxa_atexit` (and there will be no destructors emitted). It is false the method implied by `have_ctors_dtors` is used.

void TARGET_ASM_CONSTRUCTOR (*rtx symbol, int priority*) [Target Hook]

If defined, a function that outputs assembler code to arrange to call the function referenced by *symbol* at initialization time.

Assume that *symbol* is a `SYMBOL_REF` for a function taking no arguments and with no return value. If the target supports initialization priorities, *priority* is a value between 0 and `MAX_INIT_PRIORITY`; otherwise it must be `DEFAULT_INIT_PRIORITY`.

If this macro is not defined by the target, a suitable default will be chosen if (1) the target supports arbitrary section names, (2) the target defines `CTORS_SECTION_ASM_OP`, or (3) `USE_COLLECT2` is not defined.

void TARGET_ASM_DESTRUCTOR (*rtx symbol, int priority*) [Target Hook]

This is like `TARGET_ASM_CONSTRUCTOR` but used for termination functions rather than initialization functions.

If `TARGET_HAVE_CTORS_DTORS` is true, the initialization routine generated for the generated object file will have static linkage.

If your system uses `collect2` as the means of processing constructors, then that program normally uses `nm` to scan an object file for constructor functions to be called.

On certain kinds of systems, you can define this macro to make `collect2` work faster (and, in some cases, make it work at all):

OBJECT_FORMAT_COFF [Macro]

Define this macro if the system uses COFF (Common Object File Format) object files, so that `collect2` can assume this format and scan object files directly for dynamic constructor/destructor functions.

This macro is effective only in a native compiler; `collect2` as part of a cross compiler always uses `nm` for the target machine.

REAL_NM_FILE_NAME [Macro]

Define this macro as a C string constant containing the file name to use to execute `nm`. The default is to search the path normally for `nm`.

NM_FLAGS [Macro]

`collect2` calls `nm` to scan object files for static constructors and destructors and LTO info. By default, `-n` is passed. Define `NM_FLAGS` to a C string constant if other options are needed to get the same output format as GNU `nm -n` produces.

If your system supports shared libraries and has a program to list the dynamic dependencies of a given library or executable, you can define these macros to enable support for running initialization and termination functions in shared libraries:

LDD_SUFFIX [Macro]

Define this macro to a C string constant containing the name of the program which lists dynamic dependencies, like `ldd` under SunOS 4.

PARSE_LDD_OUTPUT (*ptr*) [Macro]

Define this macro to be C code that extracts filenames from the output of the program denoted by `LDD_SUFFIX`. *ptr* is a variable of type `char *` that points to the beginning of a line of output from `LDD_SUFFIX`. If the line lists a dynamic dependency, the code must advance *ptr* to the beginning of the filename on that line. Otherwise, it must set *ptr* to `NULL`.

SHLIB_SUFFIX [Macro]

Define this macro to a C string constant containing the default shared library extension of the target (e.g., `".so"`). `collect2` strips version information after this suffix when generating global constructor and destructor names. This define is only needed on targets that use `collect2` to process constructors and destructors.

17.22.7 Output of Assembler Instructions

This describes assembler instruction output.

REGISTER_NAMES [Macro]

A C initializer containing the assembler's names for the machine registers, each one as a C string constant. This is what translates register numbers in the compiler into assembler language.

ADDITIONAL_REGISTER_NAMES [Macro]

If defined, a C initializer for an array of structures containing a name and a register number. This macro defines additional names for hard registers, thus allowing the `asm` option in declarations to refer to registers using alternate names.

OVERLAPPING_REGISTER_NAMES [Macro]

If defined, a C initializer for an array of structures containing a name, a register number and a count of the number of consecutive machine registers the name overlaps. This macro defines additional names for hard registers, thus allowing the `asm` option in declarations to refer to registers using alternate names. Unlike **ADDITIONAL_REGISTER_NAMES**, this macro should be used when the register name implies multiple underlying registers.

This macro should be used when it is important that a clobber in an `asm` statement clobbers all the underlying values implied by the register name. For example, on ARM, clobbering the double-precision VFP register “d0” implies clobbering both single-precision registers “s0” and “s1”.

ASM_OUTPUT_OPCODE (*stream*, *ptr*) [Macro]

Define this macro if you are using an unusual assembler that requires different names for the machine instructions.

The definition is a C statement or statements which output an assembler instruction opcode to the stdio stream *stream*. The macro-operand *ptr* is a variable of type `char *` which points to the opcode name in its “internal” form—the form that is written in the machine description. The definition should output the opcode name to *stream*, performing any translation you desire, and increment the variable *ptr* to point at the end of the opcode so that it will not be output twice.

In fact, your macro definition may process less than the entire opcode name, or more than the opcode name; but if you want to process text that includes ‘%’-sequences to substitute operands, you must take care of the substitution yourself. Just be sure to increment *ptr* over whatever text should not be output normally.

If you need to look at the operand values, they can be found as the elements of `recog_data.operand`.

If the macro definition does nothing, the instruction is output in the usual way.

FINAL_PRESCAN_INSN (*insn*, *opvec*, *noperands*) [Macro]

If defined, a C statement to be executed just prior to the output of assembler code for *insn*, to modify the extracted operands so they will be output differently.

Here the argument *opvec* is the vector containing the operands extracted from *insn*, and *noperands* is the number of elements of the vector which contain meaningful data for this insn. The contents of this vector are what will be used to convert the insn template into assembler code, so you can change the assembler output by changing the contents of the vector.

This macro is useful when various assembler syntaxes share a single file of instruction patterns; by defining this macro differently, you can cause a large class of instructions to be output differently (such as with rearranged operands). Naturally, variations in assembler syntax affecting individual insn patterns ought to be handled by writing conditional output routines in those patterns.

If this macro is not defined, it is equivalent to a null statement.

```
void TARGET_ASM_FINAL_POSTSCAN_INSN (FILE *file, [Target Hook]
    rtx_insn *insn, rtx *opvec, int noperands)
```

If defined, this target hook is a function which is executed just after the output of assembler code for *insn*, to change the mode of the assembler if necessary.

Here the argument *opvec* is the vector containing the operands extracted from *insn*, and *noperands* is the number of elements of the vector which contain meaningful data for this insn. The contents of this vector are what was used to convert the insn template into assembler code, so you can change the assembler mode by checking the contents of the vector.

```
PRINT_OPERAND (stream, x, code) [Macro]
```

A C compound statement to output to stdio stream *stream* the assembler syntax for an instruction operand *x*. *x* is an RTL expression.

code is a value that can be used to specify one of several ways of printing the operand. It is used when identical operands must be printed differently depending on the context. *code* comes from the ‘%’ specification that was used to request printing of the operand. If the specification was just ‘%*digit*’ then *code* is 0; if the specification was ‘%*ltr digit*’ then *code* is the ASCII code for *ltr*.

If *x* is a register, this macro should print the register’s name. The names can be found in an array `reg_names` whose type is `char *`. `reg_names` is initialized from `REGISTER_NAMES`.

When the machine description has a specification ‘%*punct*’ (a ‘%’ followed by a punctuation character), this macro is called with a null pointer for *x* and the punctuation character for *code*.

```
PRINT_OPERAND_PUNCT_VALID_P (code) [Macro]
```

A C expression which evaluates to true if *code* is a valid punctuation character for use in the `PRINT_OPERAND` macro. If `PRINT_OPERAND_PUNCT_VALID_P` is not defined, it means that no punctuation characters (except for the standard one, ‘%’) are used in this way.

```
PRINT_OPERAND_ADDRESS (stream, x) [Macro]
```

A C compound statement to output to stdio stream *stream* the assembler syntax for an instruction operand that is a memory reference whose address is *x*. *x* is an RTL expression.

On some machines, the syntax for a symbolic address depends on the section that the address refers to. On these machines, define the hook `TARGET_ENCODE_SECTION_INFO` to store the information into the `symbol_ref`, and then check for it here. See Section 17.22 [Assembler Format], page 653.

DBR_OUTPUT_SEQEND (*file*) [Macro]

A C statement, to be executed after all slot-filler instructions have been output. If necessary, call `dbr_sequence_length` to determine the number of slots filled in a sequence (zero if not currently outputting a sequence), to decide how many no-ops to output, or whatever.

Don't define this macro if it has nothing to do, but it is helpful in reading assembly output if the extent of the delay sequence is made explicit (e.g. with white space).

Note that output routines for instructions with delay slots must be prepared to deal with not being output as part of a sequence (i.e. when the scheduling pass is not run, or when no slot fillers could be found.) The variable `final_sequence` is null when not processing a sequence, otherwise it contains the `sequence` rtx being output.

REGISTER_PREFIX [Macro]

LOCAL_LABEL_PREFIX [Macro]

USER_LABEL_PREFIX [Macro]

IMMEDIATE_PREFIX [Macro]

If defined, C string expressions to be used for the '%R', '%L', '%U', and '%I' options of `asm_fprintf` (see `final.cc`). These are useful when a single `md` file must support multiple assembler formats. In that case, the various `tm.h` files can define these macros differently.

ASM_FPRINTF_EXTENSIONS (*file*, *argptr*, *format*) [Macro]

If defined this macro should expand to a series of `case` statements which will be parsed inside the `switch` statement of the `asm_fprintf` function. This allows targets to define extra `printf` formats which may be useful when generating their assembler statements. Note that uppercase letters are reserved for future generic extensions to `asm_fprintf`, and so are not available to target specific code. The output file is given by the parameter *file*. The varargs input pointer is *argptr* and the rest of the format string, starting the character after the one that is being switched upon, is pointed to by *format*.

ASSEMBLER_DIALECT [Macro]

If your target supports multiple dialects of assembler language (such as different opcodes), define this macro as a C expression that gives the numeric index of the assembler language dialect to use, with zero as the first variant.

If this macro is defined, you may use constructs of the form

```
{option0|option1|option2...}
```

in the output templates of patterns (see Section 16.5 [Output Template], page 375) or in the first argument of `asm_fprintf`. This construct outputs '`option0`', '`option1`', '`option2`', etc., if the value of `ASSEMBLER_DIALECT` is zero, one, two, etc. Any special characters within these strings retain their usual meaning. If there are fewer alternatives within the braces than the value of `ASSEMBLER_DIALECT`, the construct outputs nothing. If it's needed to print curly braces or '`|`' character in assembler output directly, '%{', '%}' and '%|' can be used.

If you do not define this macro, the characters '{', '|' and '}' do not have any special meaning when used in templates or operands to `asm_fprintf`.

Define the macros `REGISTER_PREFIX`, `LOCAL_LABEL_PREFIX`, `USER_LABEL_PREFIX` and `IMMEDIATE_PREFIX` if you can express the variations in assembler language syntax with that mechanism. Define `ASSEMBLER_DIALECT` and use the ‘`{option0|option1}`’ syntax if the syntax variant are larger and involve such things as different opcodes or operand order.

ASM_OUTPUT_REG_PUSH (*stream*, *regno*) [Macro]

A C expression to output to *stream* some assembler code which will push hard register number *regno* onto the stack. The code need not be optimal, since this macro is used only when profiling.

ASM_OUTPUT_REG_POP (*stream*, *regno*) [Macro]

A C expression to output to *stream* some assembler code which will pop hard register number *regno* off of the stack. The code need not be optimal, since this macro is used only when profiling.

17.22.8 Output of Dispatch Tables

This concerns dispatch tables.

ASM_OUTPUT_ADDR_DIFF_ELT (*stream*, *body*, *value*, *rel*) [Macro]

A C statement to output to the stdio stream *stream* an assembler pseudo-instruction to generate a difference between two labels. *value* and *rel* are the numbers of two internal labels. The definitions of these labels are output using `(*targetm.asm_out.internal_label)`, and they must be printed in the same way here. For example,

```
fprintf (stream, "\t.word L%d-L%d\n",
        value, rel)
```

You must provide this macro on machines where the addresses in a dispatch table are relative to the table’s own address. If defined, GCC will also use this macro on all machines when producing PIC. *body* is the body of the `ADDR_DIFF_VEC`; it is provided so that the mode and flags can be read.

ASM_OUTPUT_ADDR_VEC_ELT (*stream*, *value*) [Macro]

This macro should be provided on machines where the addresses in a dispatch table are absolute.

The definition should be a C statement to output to the stdio stream *stream* an assembler pseudo-instruction to generate a reference to a label. *value* is the number of an internal label whose definition is output using `(*targetm.asm_out.internal_label)`. For example,

```
fprintf (stream, "\t.word L%d\n", value)
```

ASM_OUTPUT_CASE_LABEL (*stream*, *prefix*, *num*, *table*) [Macro]

Define this if the label before a jump-table needs to be output specially. The first three arguments are the same as for `(*targetm.asm_out.internal_label)`; the fourth argument is the jump-table which follows (a `jump_table_data` containing an `addr_vec` or `addr_diff_vec`).

This feature is used on system V to output a `swbeg` statement for the table.

If this macro is not defined, these labels are output with `(*targetm.asm_out.internal_label)`.

ASM_OUTPUT_CASE_END (*stream*, *num*, *table*) [Macro]

Define this if something special must be output at the end of a jump-table. The definition should be a C statement to be executed after the assembler code for the table is written. It should write the appropriate code to stdio stream *stream*. The argument *table* is the jump-table insn, and *num* is the label-number of the preceding label.

If this macro is not defined, nothing special is output at the end of the jump-table.

void TARGET_ASM_POST_CFI_STARTPROC (FILE *, *tree*) [Target Hook]

This target hook is used to emit assembly strings required by the target after the .cfi_startproc directive. The first argument is the file stream to write the strings to and the second argument is the function's declaration. The expected use is to add more .cfi_* directives.

The default is to not output any assembly strings.

void TARGET_ASM_EMIT_UNWIND_LABEL (FILE **stream*, *tree decl*, int *for_eh*, int *empty*) [Target Hook]

This target hook emits a label at the beginning of each FDE. It should be defined on targets where FDEs need special labels, and it should write the appropriate label, for the FDE associated with the function declaration *decl*, to the stdio stream *stream*. The third argument, *for_eh*, is a boolean: true if this is for an exception table. The fourth argument, *empty*, is a boolean: true if this is a placeholder label for an omitted FDE.

The default is that FDEs are not given nonlocal labels.

void TARGET_ASM_EMIT_EXCEPT_TABLE_LABEL (FILE **stream*) [Target Hook]

This target hook emits a label at the beginning of the exception table. It should be defined on targets where it is desirable for the table to be broken up according to function.

The default is that no label is emitted.

void TARGET_ASM_EMIT_EXCEPT_PERSONALITY (rtx *personality*) [Target Hook]

If the target implements TARGET_ASM_UNWIND_EMIT, this hook may be used to emit a directive to install a personality hook into the unwind info. This hook should not be used if dwarf2 unwind info is used.

void TARGET_ASM_UNWIND_EMIT (FILE **stream*, rtx_insn **insn*) [Target Hook]

This target hook emits assembly directives required to unwind the given instruction. This is only used when TARGET_EXCEPT_UNWIND_INFO returns UI_TARGET.

rtx TARGET_ASM_MAKE_EH_SYMBOL_INDIRECT (rtx *origsymbol*, bool *pubvis*) [Target Hook]

If necessary, modify personality and LSDA references to handle indirection. The original symbol is in *origsymbol* and if *pubvis* is true the symbol is visible outside the TU.

- bool TARGET_ASM_UNWIND_EMIT_BEFORE_INSN** [Target Hook]
 True if the `TARGET_ASM_UNWIND_EMIT` hook should be called before the assembly for *insn* has been emitted, false if the hook should be called afterward.
- bool TARGET_ASM_SHOULD_RESTORE_CFA_STATE (void)** [Target Hook]
 For DWARF-based unwind frames, two CFI instructions provide for save and restore of register state. GCC maintains the current frame address (CFA) separately from the register bank but the unwinder in libgcc preserves this state along with the registers (and this is expected by the code that writes the unwind frames). This hook allows the target to specify that the CFA data is not saved/restored along with the registers by the target unwinder so that suitable additional instructions should be emitted to restore it.

17.22.9 Assembler Commands for Exception Regions

This describes commands marking the start and the end of an exception region.

EH_FRAME_SECTION_NAME [Macro]
 If defined, a C string constant for the name of the section containing exception handling frame unwind information. If not defined, GCC will provide a default definition if the target supports named sections. `crtstuff.c` uses this macro to switch to the appropriate section.

You should define this symbol if your target supports DWARF 2 frame unwind information and the default definition does not work.

EH_FRAME_THROUGH_COLLECT2 [Macro]
 If defined, DWARF 2 frame unwind information will be identified by specially named labels. The collect2 process will locate these labels and generate code to register the frames.

This might be necessary, for instance, if the system linker will not place the `eh_frames` in-between the sentinels from `crtstuff.c`, or if the system linker does garbage collection and sections cannot be marked as not to be collected.

EH_TABLES_CAN_BE_READ_ONLY [Macro]
 Define this macro to 1 if your target is such that no frame unwind information encoding used with non-PIC code will ever require a runtime relocation, but the linker may not support merging read-only and read-write sections into a single read-write section.

MASK_RETURN_ADDR [Macro]
 An rtx used to mask the return address found via `RETURN_ADDR_RTX`, so that it does not contain any extraneous set bits in it.

DWARF2_UNWIND_INFO [Macro]
 Define this macro to 0 if your target supports DWARF 2 frame unwind information, but it does not yet work with exception handling. Otherwise, if your target supports this information (if it defines `INCOMING_RETURN_ADDR_RTX` and `OBJECT_FORMAT_ELF`), GCC will provide a default definition of 1.

enum unwind_info_type TARGET_EXCEPT_UNWIND_INFO [Common Target Hook]
 (struct gcc_options *opts)

This hook defines the mechanism that will be used for exception handling by the target. If the target has ABI specified unwind tables, the hook should return `UI_TARGET`. If the target is to use the `setjmp/longjmp`-based exception handling scheme, the hook should return `UI_SJLJ`. If the target supports DWARF 2 frame unwind information, the hook should return `UI_DWARF2`.

A target may, if exceptions are disabled, choose to return `UI_NONE`. This may end up simplifying other parts of target-specific code. The default implementation of this hook never returns `UI_NONE`.

Note that the value returned by this hook should be constant. It should not depend on anything except the command-line switches described by *opts*. In particular, the setting `UI_SJLJ` must be fixed at compiler start-up as C pre-processor macros and builtin functions related to exception handling are set up depending on this setting.

The default implementation of the hook first honors the `--enable-sjlj-exceptions` configure option, then `DWARF2_UNWIND_INFO`, and finally defaults to `UI_SJLJ`. If `DWARF2_UNWIND_INFO` depends on command-line options, the target must define this hook so that *opts* is used correctly.

bool TARGET_UNWIND_TABLES_DEFAULT [Common Target Hook]

This variable should be set to `true` if the target ABI requires unwinding tables even when exceptions are not used. It must not be modified by command-line option processing.

DONT_USE_BUILTIN_SETJMP [Macro]

Define this macro to 1 if the `setjmp/longjmp`-based scheme should use the `setjmp/longjmp` functions from the C library instead of the `__builtin_setjmp/__builtin_longjmp` machinery.

JMP_BUF_SIZE [Macro]

This macro has no effect unless `DONT_USE_BUILTIN_SETJMP` is also defined. Define this macro if the default size of `jmp_buf` buffer for the `setjmp/longjmp`-based exception handling mechanism is not large enough, or if it is much too large. The default size is `FIRST_PSEUDO_REGISTER * sizeof(void *)`.

DWARF_CIE_DATA_ALIGNMENT [Macro]

This macro need only be defined if the target might save registers in the function prologue at an offset to the stack pointer that is not aligned to `UNITS_PER_WORD`. The definition should be the negative minimum alignment if `STACK_GROWS_DOWNWARD` is true, and the positive minimum alignment otherwise. See Section 17.23.2 [DWARF], page 683. Only applicable if the target supports DWARF 2 frame unwind information.

bool TARGET_TERMINATE_DW2_EH_FRAME_INFO [Target Hook]

Contains the value true if the target should add a zero word onto the end of a Dwarf-2 frame info section when used for exception handling. Default value is false if `EH_FRAME_SECTION_NAME` is defined, and true otherwise.

rtx TARGET_DWARF_REGISTER_SPAN (**rtx** *reg*) [Target Hook]

Given a register, this hook should return a parallel of registers to represent where to find the register pieces. Define this hook if the register and its mode are represented in Dwarf in non-contiguous locations, or if the register should be represented in more than one register in Dwarf. Otherwise, this hook should return `NULL_RTX`. If not defined, the default is to return `NULL_RTX`.

machine_mode TARGET_DWARF_FRAME_REG_MODE (**int** *regno*) [Target Hook]

Given a register, this hook should return the mode which the corresponding Dwarf frame register should have. This is normally used to return a smaller mode than the raw mode to prevent call clobbered parts of a register altering the frame register size

bool TARGET_OUTPUT_CFI_DIRECTIVE (**FILE** * *f*, **dw_cfi_ref** *cfi*) [Target Hook]

This hook handles architecture-specific CFI directives and prints them out to the assembly file *f*. Return true if a architecture-specific directive was found, false otherwise.

bool TARGET_DW_CFI_OPRND1_DESC (**dwarf_call_frame_info** *cfi_opc*, **dw_cfi_oprnd_type** & *oprnd_type*) [Target Hook]

This hook informs the caller what the architecture-specific directives takes as a first operand. Return true if a architecture-specific directive was found and *oprnd_type* is set, false otherwise and *oprnd_type* is not modified.

void TARGET_INIT_DWARF_REG_SIZES_EXTRA (**tree** *address*) [Target Hook]

If some registers are represented in Dwarf-2 unwind information in multiple pieces, define this hook to fill in information about the sizes of those pieces in the table used by the unwinder at runtime. It will be called by `expand_builtin_init_dwarf_reg_sizes` after filling in a single size corresponding to each hard register; *address* is the address of the table.

bool TARGET_ASM_TTYPE (**rtx** *sym*) [Target Hook]

This hook is used to output a reference from a frame unwinding table to the type_info object identified by *sym*. It should return `true` if the reference was output. Returning `false` will cause the reference to be output using the normal Dwarf2 routines.

bool TARGET_ARM_EABI_UNWINDER [Target Hook]

This flag should be set to `true` on targets that use an ARM EABI based unwinding library, and `false` on other targets. This effects the format of unwinding tables, and how the unwinder is entered after running a cleanup. The default is `false`.

17.22.10 Assembler Commands for Alignment

This describes commands for alignment.

JUMP_ALIGN (*label*) [Macro]

The alignment (log base 2) to put in front of *label*, which is a common destination of jumps and has no fallthru incoming edge.

This macro need not be defined if you don't want any special alignment to be done at such a time. Most machine descriptions do not currently define the macro.

Unless it's necessary to inspect the *label* parameter, it is better to set the variable *align_jumps* in the target's `TARGET_OPTION_OVERRIDE`. Otherwise, you should try to honor the user's selection in *align_jumps* in a `JUMP_ALIGN` implementation.

LABEL_ALIGN_AFTER_BARRIER (*label*) [Macro]

The alignment (log base 2) to put in front of *label*, which follows a `BARRIER`.

This macro need not be defined if you don't want any special alignment to be done at such a time. Most machine descriptions do not currently define the macro.

LOOP_ALIGN (*label*) [Macro]

The alignment (log base 2) to put in front of *label* that heads a frequently executed basic block (usually the header of a loop).

This macro need not be defined if you don't want any special alignment to be done at such a time. Most machine descriptions do not currently define the macro.

Unless it's necessary to inspect the *label* parameter, it is better to set the variable *align_loops* in the target's `TARGET_OPTION_OVERRIDE`. Otherwise, you should try to honor the user's selection in *align_loops* in a `LOOP_ALIGN` implementation.

LABEL_ALIGN (*label*) [Macro]

The alignment (log base 2) to put in front of *label*. If `LABEL_ALIGN_AFTER_BARRIER` / `LOOP_ALIGN` specify a different alignment, the maximum of the specified values is used.

Unless it's necessary to inspect the *label* parameter, it is better to set the variable *align_labels* in the target's `TARGET_OPTION_OVERRIDE`. Otherwise, you should try to honor the user's selection in *align_labels* in a `LABEL_ALIGN` implementation.

ASM_OUTPUT_SKIP (*stream*, *nbytes*) [Macro]

A C statement to output to the stdio stream *stream* an assembler instruction to advance the location counter by *nbytes* bytes. Those bytes should be zero when loaded. *nbytes* will be a C expression of type `unsigned HOST_WIDE_INT`.

ASM_NO_SKIP_IN_TEXT [Macro]

Define this macro if `ASM_OUTPUT_SKIP` should not be used in the text section because it fails to put zeros in the bytes that are skipped. This is true on many Unix systems, where the pseudo-op to skip bytes produces no-op instructions rather than zeros when used in the text section.

ASM_OUTPUT_ALIGN (*stream*, *power*) [Macro]

A C statement to output to the stdio stream *stream* an assembler command to advance the location counter to a multiple of 2 to the *power* bytes. *power* will be a C expression of type `int`.

ASM_OUTPUT_ALIGN_WITH_NOP (*stream*, *power*) [Macro]

Like `ASM_OUTPUT_ALIGN`, except that the "nop" instruction is used for padding, if necessary.

ASM_OUTPUT_MAX_SKIP_ALIGN (*stream*, *power*, *max_skip*) [Macro]

A C statement to output to the stdio stream *stream* an assembler command to advance the location counter to a multiple of 2 to the *power* bytes, but only if *max_skip* or

fewer bytes are needed to satisfy the alignment request. *power* and *max_skip* will be a C expression of type `int`.

17.23 Controlling Debugging Information Format

This describes how to specify debugging information.

17.23.1 Macros Affecting All Debugging Formats

These macros affect all debugging formats.

DEBUGGER_REGNO (*regno*) [Macro]

A C expression that returns the debugger register number for the compiler register number *regno*. In the default macro provided, the value of this expression will be *regno* itself. But sometimes there are some registers that the compiler knows about and debugger does not, or vice versa. In such cases, some register may need to have one number in the compiler and another for debugger.

If two registers have consecutive numbers inside GCC, and they can be used as a pair to hold a multiword value, then they *must* have consecutive numbers after renumbering with `DEBUGGER_REGNO`. Otherwise, debuggers will be unable to access such a pair, because they expect register pairs to be consecutive in their own numbering scheme.

If you find yourself defining `DEBUGGER_REGNO` in way that does not preserve register pairs, then what you must do instead is redefine the actual register numbering scheme.

DEBUGGER_AUTO_OFFSET (*x*) [Macro]

A C expression that returns the integer offset value for an automatic variable having address *x* (an RTL expression). The default computation assumes that *x* is based on the frame-pointer and gives the offset from the frame-pointer. This is required for targets that produce debugging output for debugger and allow the frame-pointer to be eliminated when the `-g` option is used.

DEBUGGER_ARG_OFFSET (*offset*, *x*) [Macro]

A C expression that returns the integer offset value for an argument having address *x* (an RTL expression). The nominal offset is *offset*.

PREFERRED_DEBUGGING_TYPE [Macro]

A C expression that returns the type of debugging output GCC should produce when the user specifies just `-g`. Define this if you have arranged for GCC to support more than one format of debugging output. Currently, the allowable values are `DWARF2_DEBUG`, `VMS_DEBUG`, and `VMS_AND_DWARF2_DEBUG`.

When the user specifies `-gdb`, GCC normally also uses the value of this macro to select the debugging output format, but with two exceptions. If `DWARF2_DEBUGGING_INFO` is defined, GCC uses the value `DWARF2_DEBUG`.

The value of this macro only affects the default debugging output; the user can always get a specific type of output by using `-gdwarf-2`, or `-gvms`.

DEFAULT_GDB_EXTENSIONS [Macro]

Define this macro to control whether GCC should by default generate GDB's extended version of debugging information. If you don't define the macro, the default is 1: always generate the extended information if there is any occasion to.

17.23.2 Macros for DWARF Output

Here are macros for DWARF output.

DWARF2_DEBUGGING_INFO [Macro]

Define this macro if GCC should produce dwarf version 2 format debugging output in response to the `-g` option.

To support optional call frame debugging information, you must also define `INCOMING_RETURN_ADDR_RTX` and either set `RTX_FRAME_RELATED_P` on the prologue insns if you use RTL for the prologue, or call `dwarf2out_def_cfa` and `dwarf2out_reg_save` as appropriate from `TARGET_ASM_FUNCTION_PROLOGUE` if you don't.

int TARGET_DWARF_CALLING_CONVENTION (*const_tree* *function*) [Target Hook]

Define this to enable the dwarf attribute `DW_AT_calling_convention` to be emitted for each function. Instead of an integer return the enum value for the `DW_CC_` tag.

DWARF2_FRAME_INFO [Macro]

Define this macro to a nonzero value if GCC should always output Dwarf 2 frame information. If `TARGET_EXCEPT_UNWIND_INFO` (see Section 17.22.9 [Exception Region Output], page 678) returns `UI_DWARF2`, and exceptions are enabled, GCC will output this information not matter how you define `DWARF2_FRAME_INFO`.

enum unwind_info_type TARGET_DEBUG_UNWIND_INFO (*void*) [Target Hook]

This hook defines the mechanism that will be used for describing frame unwind information to the debugger. Normally the hook will return `UI_DWARF2` if DWARF 2 debug information is enabled, and return `UI_NONE` otherwise.

A target may return `UI_DWARF2` even when DWARF 2 debug information is disabled in order to always output DWARF 2 frame information.

A target may return `UI_TARGET` if it has ABI specified unwind tables. This will suppress generation of the normal debug frame unwind information.

DWARF2_ASM_LINE_DEBUG_INFO [Macro]

Define this macro to be a nonzero value if the assembler can generate Dwarf 2 line debug info sections. This will result in much more compact line number tables, and hence is desirable if it works.

DWARF2_ASM_VIEW_DEBUG_INFO [Macro]

Define this macro to be a nonzero value if the assembler supports view assignment and verification in `.loc`. If it does not, but the user enables location views, the compiler may have to fallback to internal line number tables.

int TARGET_RESET_LOCATION_VIEW (*rtx_insn **) [Target Hook]

This hook, if defined, enables `-ginternal-reset-location-views`, and uses its result to override cases in which the estimated min insn length might be nonzero even when a PC advance (i.e., a view reset) cannot be taken for granted.

If the hook is defined, it must return a positive value to indicate the insn definitely advances the PC, and so the view number can be safely assumed to be reset; a negative

value to mean the insn definitely does not advance the PC, and os the view number must not be reset; or zero to decide based on the estimated insn length.

If `insn length` is to be regarded as reliable, set the hook to `hook_int_rtx_insn_0`.

bool TARGET_WANT_DEBUG_PUB_SECTIONS [Target Hook]
 True if the `.debug_pubtypes` and `.debug_pubnames` sections should be emitted. These sections are not used on most platforms, and in particular GDB does not use them.

bool TARGET_DELAY_SCHED2 [Target Hook]
 True if `sched2` is not to be run at its normal place. This usually means it will be run as part of machine-specific reorg.

bool TARGET_DELAY_VARTRACK [Target Hook]
 True if `vartrack` is not to be run at its normal place. This usually means it will be run as part of machine-specific reorg.

bool TARGET_NO_REGISTER_ALLOCATION [Target Hook]
 True if register allocation and the passes following it should not be run. Usually true only for virtual assembler targets.

ASM_OUTPUT_DWARF_DELTA (*stream*, *size*, *label1*, *label2*) [Macro]
 A C statement to issue assembly directives that create a difference *lab1* minus *lab2*, using an integer of the given *size*.

ASM_OUTPUT_DWARF_VMS_DELTA (*stream*, *size*, *label1*, *label2*) [Macro]
 A C statement to issue assembly directives that create a difference between the two given labels in system defined units, e.g. instruction slots on IA64 VMS, using an integer of the given size.

ASM_OUTPUT_DWARF_OFFSET (*stream*, *size*, *label*, *offset*, *section*) [Macro]
 A C statement to issue assembly directives that create a section-relative reference to the given *label* plus *offset*, using an integer of the given *size*. The label is known to be defined in the given *section*.

ASM_OUTPUT_DWARF_PCREL (*stream*, *size*, *label*) [Macro]
 A C statement to issue assembly directives that create a self-relative reference to the given *label*, using an integer of the given *size*.

ASM_OUTPUT_DWARF_DATAREL (*stream*, *size*, *label*) [Macro]
 A C statement to issue assembly directives that create a reference to the given *label* relative to the dbase, using an integer of the given *size*.

ASM_OUTPUT_DWARF_TABLE_REF (*label*) [Macro]
 A C statement to issue assembly directives that create a reference to the DWARF table identifier *label* from the current section. This is used on some systems to avoid garbage collecting a DWARF table which is referenced by a function.

void TARGET_ASM_OUTPUT_DWARF_DTPREL (FILE **file*, int *size*, rtx *x*) [Target Hook]
 If defined, this target hook is a function which outputs a DTP-relative reference to the given TLS symbol of the specified size.

17.23.3 Macros for VMS Debug Format

Here are macros for VMS debug format.

VMS_DEBUGGING_INFO [Macro]

Define this macro if GCC should produce debugging output for VMS in response to the `-g` option. The default behavior for VMS is to generate minimal debug info for a traceback in the absence of `-g` unless explicitly overridden with `-g0`. This behavior is controlled by `TARGET_OPTION_OPTIMIZATION` and `TARGET_OPTION_OVERRIDE`.

17.23.4 Macros for CTF Debug Format

Here are macros for CTF debug format.

CTF_DEBUGGING_INFO [Macro]

Define this macro if GCC should produce debugging output in CTF debug format in response to the `-gctf` option.

17.23.5 Macros for BTF Debug Format

Here are macros for BTF debug format.

BTF_DEBUGGING_INFO [Macro]

Define this macro if GCC should produce debugging output in BTF debug format in response to the `-gbtf` option.

17.24 Cross Compilation and Floating Point

While all modern machines use twos-complement representation for integers, there are a variety of representations for floating point numbers. This means that in a cross-compiler the representation of floating point numbers in the compiled program may be different from that used in the machine doing the compilation.

Because different representation systems may offer different amounts of range and precision, all floating point constants must be represented in the target machine's format. Therefore, the cross compiler cannot safely use the host machine's floating point arithmetic; it must emulate the target's arithmetic. To ensure consistency, GCC always uses emulation to work with floating point values, even when the host and target floating point formats are identical.

The following macros are provided by `real.h` for the compiler to use. All parts of the compiler which generate or optimize floating-point calculations must use these macros. They may evaluate their operands more than once, so operands must not have side effects.

REAL_VALUE_TYPE [Macro]

The C data type to be used to hold a floating point value in the target machine's format. Typically this is a `struct` containing an array of `HOST_WIDE_INT`, but all code should treat it as an opaque quantity.

HOST_WIDE_INT REAL_VALUE_FIX (REAL_VALUE_TYPE x) [Macro]

Truncates `x` to a signed integer, rounding toward zero.

`unsigned HOST_WIDE_INT REAL_VALUE_UNSIGNED_FIX` [Macro]
`(REAL_VALUE_TYPE x)`

Truncates *x* to an unsigned integer, rounding toward zero. If *x* is negative, returns zero.

`REAL_VALUE_TYPE REAL_VALUE_ATOF (const char *string,` [Macro]
`machine_mode mode)`

Converts *string* into a floating point number in the target machine's representation for mode *mode*. This routine can handle both decimal and hexadecimal floating point constants, using the syntax defined by the C language for both.

`int REAL_VALUE_NEGATIVE (REAL_VALUE_TYPE x)` [Macro]
 Returns 1 if *x* is negative (including negative zero), 0 otherwise.

`int REAL_VALUE_ISINF (REAL_VALUE_TYPE x)` [Macro]
 Determines whether *x* represents infinity (positive or negative).

`int REAL_VALUE_ISNAN (REAL_VALUE_TYPE x)` [Macro]
 Determines whether *x* represents a “NaN” (not-a-number).

`REAL_VALUE_TYPE REAL_VALUE_NEGATE (REAL_VALUE_TYPE x)` [Macro]
 Returns the negative of the floating point value *x*.

`REAL_VALUE_TYPE REAL_VALUE_ABS (REAL_VALUE_TYPE x)` [Macro]
 Returns the absolute value of *x*.

17.25 Mode Switching Instructions

The following macros control mode switching optimizations:

`OPTIMIZE_MODE_SWITCHING (entity)` [Macro]

Define this macro if the port needs extra instructions inserted for mode switching.

For an example, the SH4 can perform both single and double precision floating point operations, but to perform a single precision operation, the FPSCR PR bit has to be cleared, while for a double precision operation, this bit has to be set. Changing the PR bit requires a general purpose register as a scratch register, hence these FPSCR sets have to be inserted before reload, i.e. you cannot put this into instruction emitting or `TARGET_MACHINE_DEPENDENT_REORG`.

You can have multiple entities that are mode-switched, some of which might only be needed conditionally. The entities are identified by their index into the `NUM_MODES_FOR_MODE_SWITCHING` initializer, with the length of the initializer determining the number of entities.

`OPTIMIZE_MODE_SWITCHING` should return nonzero for any *entity* that needs mode-switching.

If you define this macro, you also have to define `NUM_MODES_FOR_MODE_SWITCHING`, `TARGET_MODE_NEEDED`, `TARGET_MODE_PRIORITY` and `TARGET_MODE_EMIT`. The other macros in this section are optional.

NUM_MODES_FOR_MODE_SWITCHING [Macro]

If you define `OPTIMIZE_MODE_SWITCHING`, you have to define this as initializer for an array of integers. Each initializer element *N* refers to an entity that needs mode switching, and specifies the number of different modes that are defined for that entity. The position of the element in the initializer—starting counting at zero—determines the integer that is used to refer to the mode-switched entity in question. Modes are represented as numbers $0 \dots N - 1$. In mode arguments and return values, *N* either represents an unknown mode or “no mode”, depending on context.

void TARGET_MODE_EMIT (int *entity*, int *mode*, int *prev_mode*, **HARD_REG_SET** *regs_live*) [Target Hook]

Generate one or more insns to set *entity* to *mode*. *hard_reg_live* is the set of hard registers live at the point where the insn(s) are to be inserted. *prev_mode* indicates the mode to switch from, or is the number of modes if the previous mode is not known. Sets of a lower numbered entity will be emitted before sets of a higher numbered entity to a mode of the same or lower priority.

int TARGET_MODE_NEEDED (int *entity*, **rtx_insn** **insn*, **HARD_REG_SET** *regs_live*) [Target Hook]

entity is an integer specifying a mode-switched entity. If `OPTIMIZE_MODE_SWITCHING` is defined, you must define this hook to return the mode that *entity* must be switched into prior to the execution of *insn*, or the number of modes if *insn* has no such requirement. *regs_live* contains the set of hard registers that are live before *insn*.

int TARGET_MODE_AFTER (int *entity*, int *mode*, **rtx_insn** **insn*, **HARD_REG_SET** *regs_live*) [Target Hook]

entity is an integer specifying a mode-switched entity. If this hook is defined, it is evaluated for every *insn* during mode switching. It returns the mode that *entity* is in after *insn* has been executed. *mode* is the mode that *entity* was in before *insn* was executed, taking account of `TARGET_MODE_NEEDED`. *regs_live* is the set of hard registers that are live after *insn* has been executed.

mode is equal to the number of modes defined for *entity* if the mode before *insn* is unknown. The hook should likewise return the number of modes if it does not know what mode *entity* has after *insn*.

Not defining the hook is equivalent to returning *mode*.

int TARGET_MODE_CONFLUENCE (int *entity*, int *mode1*, int *mode2*) [Target Hook]

By default, the mode-switching pass assumes that a given entity’s modes are mutually exclusive. This means that the pass can only tell `TARGET_MODE_EMIT` about an entity’s previous mode if all incoming paths of execution leave the entity in the same state.

However, some entities might have overlapping, non-exclusive modes, so that it is sometimes possible to represent “mode *mode1* or mode *mode2*” with something more specific than “mode not known”. If this is true for at least one entity, you should define this hook and make it return a mode that includes *mode1* and *mode2* as possibilities. (The mode can include other possibilities too.) The hook should return the number of modes if no suitable mode exists for the given arguments.

```
int TARGET_MODE_BACKPROP (int entity, int model, int mode2) [Target Hook]
```

If defined, the mode-switching pass uses this hook to back-propagate mode requirements through blocks that have no mode requirements of their own. Specifically, *model* is the mode that *entity* has on exit from a block B1 (say) and *mode2* is the mode that the next block requires *entity* to have. B1 does not have any mode requirements of its own.

The hook should return the mode that it prefers or requires *entity* to have in B1, or the number of modes if there is no such requirement. If the hook returns a required mode for more than one of B1's outgoing edges, those modes are combined as for TARGET_MODE_CONFLUENCE.

For example, suppose there is a “one-shot” entity that, for a given execution of a function, either stays off or makes exactly one transition from off to on. It is safe to make the transition at any time, but it is better not to do so unnecessarily. This hook allows the function to manage such an entity without having to track its state at runtime. Specifically, the entity would have two modes, 0 for off and 1 for on, with 2 representing “don't know”. The system is forbidden from transitioning from 2 to 1, since 2 represents the possibility that the entity is already on (and the aim is to avoid having to emit code to check for that case). This hook would therefore return 1 when *model* is 2 and *mode2* is 1, which would force the entity to be on in the source block. Applying this inductively would remove all transitions in which the previous state is unknown.

```
int TARGET_MODE_ENTRY (int entity) [Target Hook]
```

If this hook is defined, it is evaluated for every *entity* that needs mode switching. It should return the mode that *entity* is guaranteed to be in on entry to the function, or the number of modes if there is no such guarantee. If TARGET_MODE_ENTRY is defined then TARGET_MODE_EXIT must be defined.

```
int TARGET_MODE_EXIT (int entity) [Target Hook]
```

If this hook is defined, it is evaluated for every *entity* that needs mode switching. It should return the mode that *entity* must be in on return from the function, or the number of modes if there is no such requirement. If TARGET_MODE_EXIT is defined then TARGET_MODE_ENTRY must be defined.

```
int TARGET_MODE_EH_HANDLER (int entity) [Target Hook]
```

If this hook is defined, it should return the mode that *entity* is guaranteed to be in on entry to an exception handler, or the number of modes if there is no such guarantee.

```
int TARGET_MODE_PRIORITY (int entity, int n) [Target Hook]
```

This hook specifies the order in which modes for *entity* are processed. 0 is the highest priority, NUM_MODES_FOR_MODE_SWITCHING[*entity*] - 1 the lowest. The hook returns an integer designating a mode for *entity*. For any fixed *entity*, mode_priority(*entity*, *n*) shall be a bijection in 0 ... num_modes_for_mode_switching[*entity*] - 1.

17.26 Defining target-specific uses of __attribute__

Target-specific attributes may be defined for functions, data and types. These are described using the following target hooks; they also need to be documented in `extend.texi`.

`array_slice<const struct scoped_attribute_specs *const>` [Target Hook]
`TARGET_ATTRIBUTE_TABLE`

If defined, this target hook provides an array of ‘`scoped_attribute_spec`’s (defined in `attrs.h`) that specify the machine-specific attributes for this target. The information includes some of the restrictions on the entities to which these attributes are applied and the arguments that the attributes take.

In C and C++, these attributes are associated with two syntaxes: the traditional GNU `__attribute__` syntax and the standard ‘`[]`’ syntax. Attributes that support the GNU syntax must be placed in the `gnu` namespace. Such attributes can then also be written ‘`[[gnu:...]]`’. Attributes that use only the standard syntax should be placed in whichever namespace the attribute specification requires. For example, a target might choose to support vendor-specific ‘`[]`’ attributes that the vendor places in their own namespace.

Targets that only define attributes in the `gnu` namespace can use the following shorthand to define the table:

```
TARGET_GNU_ATTRIBUTES (cpu_attribute_table, {
    { "attribute1", ... },
    { "attribute2", ... },
    ...,
    { "attributen", ... },
});
```

`bool TARGET_ATTRIBUTE_TAKES_IDENTIFIER_P (const_tree` [Target Hook]
`name)`

If defined, this target hook is a function which returns true if the machine-specific attribute named *name* expects an identifier given as its first argument to be passed on as a plain identifier, not subjected to name lookup. If this is not defined, the default is false for all machine-specific attributes.

`int TARGET_COMP_TYPE_ATTRIBUTES (const_tree type1,` [Target Hook]
`const_tree type2)`

If defined, this target hook is a function which returns zero if the attributes on *type1* and *type2* are incompatible, one if they are compatible, and two if they are nearly compatible (which causes a warning to be generated). If this is not defined, machine-specific attributes are supposed always to be compatible.

`void TARGET_SET_DEFAULT_TYPE_ATTRIBUTES (tree type)` [Target Hook]

If defined, this target hook is a function which assigns default attributes to the newly defined *type*.

`tree TARGET_MERGE_TYPE_ATTRIBUTES (tree type1, tree` [Target Hook]
`type2)`

Define this target hook if the merging of type attributes needs special handling. If defined, the result is a list of the combined `TYPE_ATTRIBUTES` of *type1* and *type2*. It is assumed that `comptypes` has already been called and returned 1. This function may call `merge_attributes` to handle machine-independent merging.

```
tree TARGET_MERGE_DECL_ATTRIBUTES (tree olddecl, tree newdecl) [Target Hook]
```

Define this target hook if the merging of decl attributes needs special handling. If defined, the result is a list of the combined `DECL_ATTRIBUTES` of *olddecl* and *newdecl*. *newdecl* is a duplicate declaration of *olddecl*. Examples of when this is needed are when one attribute overrides another, or when an attribute is nullified by a subsequent definition. This function may call `merge_attributes` to handle machine-independent merging.

If the only target-specific handling you require is ‘`dllimport`’ for Microsoft Windows targets, you should define the macro `TARGET_DLLIMPORT_DECL_ATTRIBUTES` to 1. The compiler will then define a function called `merge_dllimport_decl_attributes` which can then be defined as the expansion of `TARGET_MERGE_DECL_ATTRIBUTES`. You can also add `handle_dll_attribute` in the attribute table for your port to perform initial processing of the ‘`dllimport`’ and ‘`dllexport`’ attributes. This is done in `i386/cygwin.h` and `i386/i386.cc`, for example.

```
bool TARGET_VALID_DLLIMPORT_ATTRIBUTE_P (const_tree decl) [Target Hook]
```

decl is a variable or function with `__attribute__((dllimport))` specified. Use this hook if the target needs to add extra validation checks to `handle_dll_attribute`.

```
TARGET_DECLSPEC [Macro]
```

Define this macro to a nonzero value if you want to treat `__declspec(X)` as equivalent to `__attribute__((X))`. By default, this behavior is enabled only for targets that define `TARGET_DLLIMPORT_DECL_ATTRIBUTES`. The current implementation of `__declspec` is via a built-in macro, but you should not rely on this implementation detail.

```
void TARGET_INSERT_ATTRIBUTES (tree node, tree attr_ptr) [Target Hook]
```

Define this target hook if you want to be able to add attributes to a decl when it is being created. This is normally useful for back ends which wish to implement a pragma by using the attributes which correspond to the pragma’s effect. The *node* argument is the decl which is being created. The *attr_ptr* argument is a pointer to the attribute list for this decl. The list itself should not be modified, since it may be shared with other decls, but attributes may be chained on the head of the list and **attr_ptr* modified to point to the new attributes, or a copy of the list may be made if further changes are needed.

```
tree TARGET_HANDLE_GENERIC_ATTRIBUTE (tree *node, tree name, tree args, int flags, bool *no_add_attrs) [Target Hook]
```

Define this target hook if you want to be able to perform additional target-specific processing of an attribute which is handled generically by a front end. The arguments are the same as those which are passed to attribute handlers. So far this only affects the *noinit* and *section* attribute.

bool TARGET_FUNCTION_ATTRIBUTE_INLINABLE_P (*const_tree fndecl*) [Target Hook]

This target hook returns **false** if the target-specific attributes on *fndecl* always block it getting inlined, **true** otherwise. By default, if a function has a target specific attribute attached to it, it will not be inlined.

bool TARGET_OPTION_VALID_ATTRIBUTE_P (*tree fndecl, tree name, tree args, int flags*) [Target Hook]

This hook is called to parse `attribute(target("..."))`, which allows setting target-specific options on individual functions. These function-specific options may differ from the options specified on the command line. The hook should return **true** if the options are valid.

The hook should set the `DECL_FUNCTION_SPECIFIC_TARGET` field in the function declaration to hold a pointer to a target-specific `struct cl_target_option` structure.

bool TARGET_OPTION_VALID_VERSION_ATTRIBUTE_P (*tree fndecl, tree name, tree args, int flags*) [Target Hook]

This hook is called to parse `attribute(target_version("..."))`, which allows setting target-specific options on individual function versions. These function-specific options may differ from the options specified on the command line. The hook should return **true** if the options are valid.

The hook should set the `DECL_FUNCTION_SPECIFIC_TARGET` field in the function declaration to hold a pointer to a target-specific `struct cl_target_option` structure.

TARGET_HAS_FMV_TARGET_ATTRIBUTE [Macro]

Define this macro to zero to use `target_version` attributes for function multiversioning (FMV) rather than `target` attributes.

Targets using `target_version` attributes will also have "target_version" FMV semantics, which allow for FMV sets defined across TU's and using a combination of `target_version` and `target_clones` attributed declarations in the definition of a FMV function set.

TARGET_CLONES_ATTR_SEPARATOR [Macro]

Define this char-typed macro to select a character that separates each target specific attributes from the `attribute(target_clones("..."))` attribute string. This macro should be carefully chosen to avoid conflicts with the target specific attributes. The default value is `'.'`.

void TARGET_OPTION_SAVE (*struct cl_target_option *ptr, struct gcc_options *opts, struct gcc_options *opts_set*) [Target Hook]

This hook is called to save any additional target-specific information in the `struct cl_target_option` structure for function-specific options from the `struct gcc_options` structure. See Section 7.1 [Option file format], page 135.

void TARGET_OPTION_RESTORE (*struct gcc_options *opts, struct gcc_options *opts_set, struct cl_target_option *ptr*) [Target Hook]

This hook is called to restore any additional target-specific information in the `struct cl_target_option` structure for function-specific options to the `struct gcc_options` structure.

void TARGET_OPTION_POST_STREAM_IN (**struct** [Target Hook]
 cl_target_option *ptr)

This hook is called to update target-specific information in the **struct cl_target_option** structure after it is streamed in from LTO bytecode.

void TARGET_OPTION_PRINT (**FILE *file**, **int indent**, [Target Hook]
 struct cl_target_option *ptr)

This hook is called to print any additional target-specific information in the **struct cl_target_option** structure for function-specific options.

bool TARGET_OPTION_PRAGMA_PARSE (**tree args**, **tree** [Target Hook]
 pop_target)

This target hook parses the options for **#pragma GCC target**, which sets the target-specific options for functions that occur later in the input stream. The options accepted should be the same as those handled by the **TARGET_OPTION_VALID_ATTRIBUTE_P** hook.

void TARGET_OPTION_OVERRIDE (**void**) [Target Hook]

Sometimes certain combinations of command options do not make sense on a particular target machine. You can override the hook **TARGET_OPTION_OVERRIDE** to take account of this. This hook is called once just after all the command options have been parsed.

Don't use this hook to turn on various extra optimizations for **-O**. That is what **TARGET_OPTION_OPTIMIZATION** is for.

If you need to do something whenever the optimization level is changed via the **optimize** attribute or **pragma**, see **TARGET_OVERRIDE_OPTIONS_AFTER_CHANGE**

bool TARGET_OPTION_SAME_FUNCTION_VERSIONS (**string_slice** [Target Hook]
 fn1, **const_tree decl1**, **string_slice fn2**, **const_tree decl2**)

This target hook returns **true** if the target/target-version strings **fn1** and **fn2** imply the same function version.

If **decl1** and **decl2** are non **NULL** then **fn1** and **fn2** are from an earlier and a later declaration respectively, and the hook diagnoses any version string incompatibility for these decls. If the version strings are incompatible and the declarations can not be merged, then hook emits an error.

bool TARGET_OPTION_FUNCTIONS_B_RESOLVABLE_FROM_A (**tree** [Target Hook]
 decl_a, **tree decl_v**, **tree base**)

decl_b is a function declaration with a function multi-versioning (FMV) attribute; this attribute is either **target** or **target_version**, depending on **TARGET_HAS_FMV_TARGET_ATTRIBUTE**. **decl_a** is a function declaration that may or may not have an FMV attribute.

Return **true** if we have enough information to determine that the requirements of **decl_b**'s FMV attribute are met whenever **decl_a** is executed, given that the target supports all features required by function declaration **base**.

The default implementation just checks whether **decl_a** has the same FMV attribute as **decl_b**. This is conservatively correct, but ports can do better by taking the

relationships between architecture features into account. For example, on AArch64, `sve` is present whenever `sve2` is present.

bool TARGET_CAN_INLINE_P (tree *caller*, tree *callee*) [Target Hook]

This target hook returns `false` if the *caller* function cannot inline *callee*, based on target specific information. By default, inlining is not allowed if the callee function has function specific target options and the caller does not use the same options.

bool TARGET_UPDATE_IPA_FN_TARGET_INFO (unsigned int& *info*, const gimple* *stmt*) [Target Hook]

Allow target to analyze all gimple statements for the given function to record and update some target specific information for inlining. A typical example is that a caller with one isa feature disabled is normally not allowed to inline a callee with that same isa feature enabled even which is attributed by `always_inline`, but with the conservative analysis on all statements of the callee if we are able to guarantee the callee does not exploit any instructions from the mismatch isa feature, it would be safe to allow the caller to inline the callee. *info* is one `unsigned int` value to record information in which one set bit indicates one corresponding feature is detected in the analysis, *stmt* is the statement being analyzed. Return true if target still need to analyze the subsequent statements, otherwise return false to stop subsequent analysis. The default version of this hook returns false.

bool TARGET_NEED_IPA_FN_TARGET_INFO (const_tree *decl*, unsigned int& *info*) [Target Hook]

Allow target to check early whether it is necessary to analyze all gimple statements in the given function to update target specific information for inlining. See hook `update_ipa_fn_target_info` for usage example of target specific information. This hook is expected to be invoked ahead of the iterating with hook `update_ipa_fn_target_info`. *decl* is the function being analyzed, *info* is the same as what in hook `update_ipa_fn_target_info`, target can do one time update into *info* without iterating for some case. Return true if target decides to analyze all gimple statements to collect information, otherwise return false. The default version of this hook returns false.

void TARGET_RELAYOUT_FUNCTION (tree *fndekl*) [Target Hook]

This target hook fixes function *fndekl* after attributes are processed. Default does nothing. On ARM, the default function's alignment is updated with the attribute `target`.

17.27 Emulating TLS

For targets whose psABI does not provide Thread Local Storage via specific relocations and instruction sequences, an emulation layer is used. A set of target hooks allows this emulation layer to be configured for the requirements of a particular target. For instance the psABI may in fact specify TLS support in terms of an emulation layer.

The emulation layer works by creating a control object for every TLS object. To access the TLS object, a lookup function is provided which, when given the address of the control object, will return the address of the current thread's instance of the TLS object.

- const char * TARGET_EMUTLS_GET_ADDRESS** [Target Hook]
 Contains the name of the helper function that uses a TLS control object to locate a TLS instance. The default causes libgcc's emulated TLS helper function to be used.
- const char * TARGET_EMUTLS_REGISTER_COMMON** [Target Hook]
 Contains the name of the helper function that should be used at program startup to register TLS objects that are implicitly initialized to zero. If this is NULL, all TLS objects will have explicit initializers. The default causes libgcc's emulated TLS registration function to be used.
- const char * TARGET_EMUTLS_VAR_SECTION** [Target Hook]
 Contains the name of the section in which TLS control variables should be placed. The default of NULL allows these to be placed in any section.
- const char * TARGET_EMUTLS_TMPL_SECTION** [Target Hook]
 Contains the name of the section in which TLS initializers should be placed. The default of NULL allows these to be placed in any section.
- const char * TARGET_EMUTLS_VAR_PREFIX** [Target Hook]
 Contains the prefix to be prepended to TLS control variable names. The default of NULL uses a target-specific prefix.
- const char * TARGET_EMUTLS_TMPL_PREFIX** [Target Hook]
 Contains the prefix to be prepended to TLS initializer objects. The default of NULL uses a target-specific prefix.
- tree TARGET_EMUTLS_VAR_FIELDS (tree type, tree *name)** [Target Hook]
 Specifies a function that generates the FIELD_DECLS for a TLS control object type. *type* is the RECORD_TYPE the fields are for and *name* should be filled with the structure tag, if the default of `__emutls_object` is unsuitable. The default creates a type suitable for libgcc's emulated TLS function.
- tree TARGET_EMUTLS_VAR_INIT (tree var, tree decl, tree *tmpl_addr*)** [Target Hook]
 Specifies a function that generates the CONSTRUCTOR to initialize a TLS control object. *var* is the TLS control object, *decl* is the TLS object and *tmpl_addr* is the address of the initializer. The default initializes libgcc's emulated TLS control object.
- bool TARGET_EMUTLS_VAR_ALIGN_FIXED** [Target Hook]
 Specifies whether the alignment of TLS control variable objects is fixed and should not be increased as some backends may do to optimize single objects. The default is false.
- bool TARGET_EMUTLS_DEBUG_FORM_TLS_ADDRESS** [Target Hook]
 Specifies whether a DWARF `DW_OP_form_tls_address` location descriptor may be used to describe emulated TLS control objects.

17.28 Defining coprocessor specifics for MIPS targets.

The MIPS specification allows MIPS implementations to have as many as 4 coprocessors, each with as many as 32 private registers. GCC supports accessing these registers and transferring values between the registers and memory using asm-sized variables. For example:

```
register unsigned int cp0count asm ("c0r1");
unsigned int d;

d = cp0count + 3;
```

(“c0r1” is the default name of register 1 in coprocessor 0; alternate names may be added as described below, or the default names may be overridden entirely in `SUBTARGET_CONDITIONAL_REGISTER_USAGE`.)

Coprocessor registers are assumed to be epilogue-used; sets to them will be preserved even if it does not appear that the register is used again later in the function.

Another note: according to the MIPS spec, coprocessor 1 (if present) is the FPU. One accesses COP1 registers through standard mips floating-point support; they are not included in this mechanism.

17.29 Parameters for Precompiled Header Validity Checking

`void * TARGET_GET_PCH_VALIDITY (size_t *sz)` [Target Hook]
This hook returns a pointer to the data needed by `TARGET_PCH_VALID_P` and sets ‘*sz’ to the size of the data in bytes.

`const char * TARGET_PCH_VALID_P (const void *data, size_t sz)` [Target Hook]
This hook checks whether the options used to create a PCH file are compatible with the current settings. It returns NULL if so and a suitable error message if not. Error messages will be presented to the user and must be localized using ‘_(msg)’.

data is the data that was returned by `TARGET_GET_PCH_VALIDITY` when the PCH file was created and *sz* is the size of that data in bytes. It’s safe to assume that the data was created by the same version of the compiler, so no format checking is needed.

The default definition of `default_pch_valid_p` should be suitable for most targets.

`const char * TARGET_CHECK_PCH_TARGET_FLAGS (int pch_flags)` [Target Hook]
If this hook is nonnull, the default implementation of `TARGET_PCH_VALID_P` will use it to check for compatible values of `target_flags`. *pch_flags* specifies the value that `target_flags` had when the PCH file was created. The return value is the same as for `TARGET_PCH_VALID_P`.

`void TARGET_PREPARE_PCH_SAVE (void)` [Target Hook]
Called before writing out a PCH file. If the target has some garbage-collected data that needs to be in a particular state on PCH loads, it can use this hook to enforce that state. Very few targets need to do anything here.

17.30 C++ ABI parameters

- tree** TARGET_CXX_GUARD_TYPE (void) [Target Hook]
 Define this hook to override the integer type used for guard variables. These are used to implement one-time construction of static objects. The default is `long_long_integer_type_node`.
- bool** TARGET_CXX_GUARD_MASK_BIT (void) [Target Hook]
 This hook determines how guard variables are used. It should return **false** (the default) if the first byte should be used. A return value of **true** indicates that only the least significant bit should be used.
- tree** TARGET_CXX_GET_COOKIE_SIZE (tree type) [Target Hook]
 This hook returns the size of the cookie to use when allocating an array whose elements have the indicated *type*. Assumes that it is already known that a cookie is needed. The default is `max(sizeof (size_t), alignof (type))`, as defined in section 2.7 of the IA64/Generic C++ ABI.
- bool** TARGET_CXX_COOKIE_HAS_SIZE (void) [Target Hook]
 This hook should return **true** if the element size should be stored in array cookies. The default is to return **false**.
- int** TARGET_CXX_IMPORT_EXPORT_CLASS (tree type, int import_export) [Target Hook]
 If defined by a backend this hook allows the decision made to export class *type* to be overruled. Upon entry *import_export* will contain 1 if the class is going to be exported, -1 if it is going to be imported and 0 otherwise. This function should return the modified value and perform any other actions necessary to support the backend's targeted operating system.
- bool** TARGET_CXX_CDTOR_RETURNS_THIS (void) [Target Hook]
 This hook should return **true** if constructors and destructors return the address of the object created/destroyed. The default is to return **false**.
- bool** TARGET_CXX_KEY_METHOD_MAY_BE_INLINE (void) [Target Hook]
 This hook returns true if the key method for a class (i.e., the method which, if defined in the current translation unit, causes the virtual table to be emitted) may be an inline function. Under the standard Itanium C++ ABI the key method may be an inline function so long as the function is not declared inline in the class definition. Under some variants of the ABI, an inline function can never be the key method. The default is to return **true**.
- void** TARGET_CXX_DETERMINE_CLASS_DATA_VISIBILITY (tree decl) [Target Hook]
decl is a virtual table, virtual table table, typeinfo object, or other similar implicit class data object that will be emitted with external linkage in this translation unit. No ELF visibility has been explicitly specified. If the target needs to specify a visibility other than that of the containing class, use this hook to set `DECL_VISIBILITY` and `DECL_VISIBILITY_SPECIFIED`.

- bool TARGET_CXX_CLASS_DATA_ALWAYS_COMDAT (void)** [Target Hook]
 This hook returns true (the default) if virtual tables and other similar implicit class data objects are always COMDAT if they have external linkage. If this hook returns false, then class data for classes whose virtual table will be emitted in only one translation unit will not be COMDAT.
- bool TARGET_CXX_LIBRARY_RTTI_COMDAT (void)** [Target Hook]
 This hook returns true (the default) if the RTTI information for the basic types which is defined in the C++ runtime should always be COMDAT, false if it should not be COMDAT.
- bool TARGET_CXX_USE_AEABI_ATEXIT (void)** [Target Hook]
 This hook returns true if `__aeabi_atexit` (as defined by the ARM EABI) should be used to register static destructors when `-fuse-cxa-atexit` is in effect. The default is to return false to use `__cxa_atexit`.
- bool TARGET_CXX_USE_ATEXIT_FOR_CXA_ATEXIT (void)** [Target Hook]
 This hook returns true if the target `atexit` function can be used in the same manner as `__cxa_atexit` to register C++ static destructors. This requires that `atexit`-registered functions in shared libraries are run in the correct order when the libraries are unloaded. The default is to return false.
- tree TARGET_CXX_ADJUST_CDTOR_CALLABI_FNTYPE (tree *fntype*)** [Target Hook]
 This hook returns a possibly modified `FUNCTION_TYPE` for arguments to `__cxa_atexit`, `__cxa_thread_atexit` or `__cxa_throw` function pointers. ABIs like mingw32 require special attributes to be added to function types pointed to by arguments of these functions. The default is to return the passed argument unmodified.
- void TARGET_CXX_ADJUST_CLASS_AT_DEFINITION (tree *type*)** [Target Hook]
type is a C++ class (i.e., `RECORD_TYPE` or `UNION_TYPE`) that has just been defined. Use this hook to make adjustments to the class (eg, tweak visibility or perform any other required target modifications).
- tree TARGET_CXX_DECL_MANGLING_CONTEXT (const_tree *decl*)** [Target Hook]
 Return target-specific mangling context of *decl* or `NULL_TREE`.

17.31 D ABI parameters

- void TARGET_D_CPU_VERSIONS (void)** [D Target Hook]
 Declare all environmental version identifiers relating to the target CPU using the function `builtin_version`, which takes a string representing the name of the version. Version identifiers predefined by this hook apply to all modules that are being compiled and imported.
- void TARGET_D_OS_VERSIONS (void)** [D Target Hook]
 Similarly to `TARGET_D_CPU_VERSIONS`, but is used for versions relating to the target operating system.

void TARGET_D_REGISTER_CPU_TARGET_INFO (void) [D Target Hook]
 Register all target information keys relating to the target CPU using the function `d_add_target_info_handlers`, which takes a ‘`struct d_target_info_spec`’ (defined in `d/d-target.h`). The keys added by this hook are made available at compile time by the `__traits(getTargetInfo)` extension, the result is an expression describing the requested target information.

void TARGET_D_REGISTER_OS_TARGET_INFO (void) [D Target Hook]
 Same as `TARGET_D_CPU_TARGET_INFO`, but is used for keys relating to the target operating system.

const char * TARGET_D_MINFO_SECTION [D Target Hook]
 Contains the name of the section in which module info references should be placed. By default, the compiler puts all module info symbols in the “minfo” section. Define this macro to override the string if a different section name should be used. This section is expected to be bracketed by two symbols `TARGET_D_MINFO_SECTION_START` and `TARGET_D_MINFO_SECTION_END` to indicate the start and end address of the section, so that the runtime library can collect all modules for each loaded shared library and executable. Setting the value to `NULL` disables the use of sections for storing module info altogether.

const char * TARGET_D_MINFO_SECTION_START [D Target Hook]
 If `TARGET_D_MINFO_SECTION` is defined, then this must also be defined as the name of the symbol indicating the start address of the module info section

const char * TARGET_D_MINFO_SECTION_END [D Target Hook]
 If `TARGET_D_MINFO_SECTION` is defined, then this must also be defined as the name of the symbol indicating the end address of the module info section

bool TARGET_D_HAS_STDCALL_CONVENTION (unsigned int *link_system, unsigned int *link_windows) [D Target Hook]
 Returns `true` if the target supports the `stdcall` calling convention. The hook should also set `link_system` to 1 if the `stdcall` attribute should be applied to functions with `extern(System)` linkage, and `link_windows` to 1 to apply `stdcall` to functions with `extern(Windows)` linkage.

bool TARGET_D_TEMPLATES_ALWAYS_COMDAT [D Target Hook]
 This flag is true if instantiated functions and variables are always `COMDAT` if they have external linkage. If this flag is false, then instantiated decls will be emitted as weak symbols. The default is `false`.

17.32 Rust ABI parameters

void TARGET_RUST_CPU_INFO (void) [Rust Target Hook]
 Declare all environmental CPU info and features relating to the target CPU using the function `rust_add_target_info`, which takes a string representing the feature key and a string representing the feature value. Configuration pairs predefined by this hook apply to all files that are being compiled.

void TARGET_RUST_OS_INFO (void) [Rust Target Hook]
 Similar to `TARGET_RUST_CPU_INFO`, but is used for configuration info relating to the target operating system.

17.33 JIT ABI parameters

void TARGET_JIT_REGISTER_CPU_TARGET_INFO (void) [JIT Target Hook]
 Register all target information keys relating to the target CPU using the function `jit_add_target_info`, which takes a key and a value. The keys added by this hook are made available at compile time by calling `get_target_info`.

17.34 Adding support for named address spaces

The draft technical report of the ISO/IEC JTC1 S22 WG14 N1275 standards committee, *Programming Languages - C - Extensions to support embedded processors*, specifies a syntax for embedded processors to specify alternate address spaces. You can configure a GCC port to support section 5.1 of the draft report to add support for address spaces other than the default address space. These address spaces are new keywords that are similar to the `volatile` and `const` type attributes.

Pointers to named address spaces can have a different size than pointers to the generic address space.

For example, the SPU port uses the `__ea` address space to refer to memory in the host processor, rather than memory local to the SPU processor. Access to memory in the `__ea` address space involves issuing DMA operations to move data between the host processor and the local processor memory address space. Pointers in the `__ea` address space are either 32 bits or 64 bits based on the `-mea32` or `-mea64` switches (native SPU pointers are always 32 bits).

Internally, address spaces are represented as a small integer in the range 0 to 15 with address space 0 being reserved for the generic address space.

To register a named address space qualifier keyword with the C front end, the target may call the `c_register_addr_space` routine. For example, the SPU port uses the following to declare `__ea` as the keyword for named address space #1:

```
#define ADDR_SPACE_EA 1
c_register_addr_space ("__ea", ADDR_SPACE_EA);
```

scalar_int_mode TARGET_ADDR_SPACE_POINTER_MODE [Target Hook]
 (`addr_space_t address_space`)
 Define this to return the machine mode to use for pointers to *address_space* if the target supports named address spaces. The default version of this hook returns `ptr_mode`.

scalar_int_mode TARGET_ADDR_SPACE_ADDRESS_MODE [Target Hook]
 (`addr_space_t address_space`)
 Define this to return the machine mode to use for addresses in *address_space* if the target supports named address spaces. The default version of this hook returns `Pmode`.

bool TARGET_ADDR_SPACE_VALID_POINTER_MODE [Target Hook]
 (*scalar_int_mode mode*, *addr_space_t as*)

Define this to return nonzero if the port can handle pointers with machine mode *mode* to address space *as*. This target hook is the same as the **TARGET_VALID_POINTER_MODE** target hook, except that it includes explicit named address space support. The default version of this hook returns true for the modes returned by either the **TARGET_ADDR_SPACE_POINTER_MODE** or **TARGET_ADDR_SPACE_ADDRESS_MODE** target hooks for the given address space.

bool TARGET_ADDR_SPACE_LEGITIMATE_ADDRESS_P [Target Hook]
 (*machine_mode mode*, *rtx exp*, *bool strict*, *addr_space_t as*,
 code_helper ch)

Define this to return true if *exp* is a valid address for mode *mode* in the named address space *as* with the use context *ch*. The *strict* parameter says whether strict addressing is in effect after reload has finished. The *ch* indicates what context *exp* will be used for. This target hook is the same as the **TARGET_LEGITIMATE_ADDRESS_P** target hook, except that it includes explicit named address space support.

rtx TARGET_ADDR_SPACE_LEGITIMIZE_ADDRESS (*rtx x*, *rtx oldx*, *machine_mode mode*, *addr_space_t as*) [Target Hook]

Define this to modify an invalid address *x* to be a valid address with mode *mode* in the named address space *as*. This target hook is the same as the **TARGET_LEGITIMIZE_ADDRESS** target hook, except that it includes explicit named address space support.

bool TARGET_ADDR_SPACE_SUBSET_P (*addr_space_t subset*, *addr_space_t superset*) [Target Hook]

Define this to return whether the *subset* named address space is contained within the *superset* named address space. Pointers to a named address space that is a subset of another named address space will be converted automatically without a cast if used together in arithmetic operations. Pointers to a superset address space can be converted to pointers to a subset address space via explicit casts.

bool TARGET_ADDR_SPACE_ZERO_ADDRESS_VALID (*addr_space_t as*) [Target Hook]

Define this to modify the default handling of address 0 for the address space. Return true if 0 should be considered a valid address.

rtx TARGET_ADDR_SPACE_CONVERT (*rtx op*, *tree from_type*, *tree to_type*) [Target Hook]

Define this to convert the pointer expression represented by the RTL *op* with type *from_type* that points to a named address space to a new pointer expression with type *to_type* that points to a different named address space. When this hook is called, it is guaranteed that one of the two address spaces is a subset of the other, as determined by the **TARGET_ADDR_SPACE_SUBSET_P** target hook.

int TARGET_ADDR_SPACE_DEBUG (*addr_space_t as*) [Target Hook]

Define this to define how the address space is encoded in dwarf. The result is the value to be used with **DW_AT_address_class**.

```
void TARGET_ADDR_SPACE_DIAGNOSE_USAGE (addr_space_t as,      [Target Hook]
    location_t loc)
```

Define this hook if the availability of an address space depends on command line options and some diagnostics should be printed when the address space is used. This hook is called during parsing and allows to emit a better diagnostic compared to the case where the address space was not registered with `c_register_addr_space`. `as` is the address space as registered with `c_register_addr_space`. `loc` is the location of the address space qualifier token. The default implementation does nothing.

```
addr_space_t TARGET_ADDR_SPACE_FOR_ARTIFICIAL_RODATA          [Target Hook]
    (tree type, enum artificial_rodata purpose)
```

Define this hook to return a named address space to be used for `type`, usually the type of an artificial lookup-table that would reside in `.rodata` and in the generic address space.

The hook can be used to put compiler-generated, artificial lookup tables in static storage into a non-generic address space when it is better suited than the generic address space. The compiler will generate all accesses to the respective data so that all associated accesses will also use the specified address space and pointer mode.

`type` is the type of the lookup table. `purpose` specifies the purpose of the lookup table. It is one of:

ARTIFICIAL_RODATA_CSWITCH

`tree-switch-conversion.cc` lowered a GIMPLE_SWITCH expressions to something more efficient than a jump table.

ARTIFICIAL_RODATA_CRC

`gimple-crc-optimization.cc` optimized a CRC computation by using a polynomial lookup table.

The default implementation of the hook returns `ADDR_SPACE_GENERIC`.

17.35 Miscellaneous Parameters

Here are several miscellaneous parameters.

HAS_LONG_COND_BRANCH [Macro]

Define this boolean macro to indicate whether or not your architecture has conditional branches that can span all of memory. It is used in conjunction with an optimization that partitions hot and cold basic blocks into separate sections of the executable. If this macro is set to false, gcc will convert any conditional branches that attempt to cross between sections into unconditional branches or indirect jumps.

HAS_LONG_UNCOND_BRANCH [Macro]

Define this boolean macro to indicate whether or not your architecture has unconditional branches that can span all of memory. It is used in conjunction with an optimization that partitions hot and cold basic blocks into separate sections of the executable. If this macro is set to false, gcc will convert any unconditional branches that attempt to cross between sections into indirect jumps.

CASE_VECTOR_MODE [Macro]

An alias for a machine mode name. This is the machine mode that elements of a jump-table should have.

CASE_VECTOR_SHORTEN_MODE (*min_offset*, *max_offset*, *body*) [Macro]

Optional: return the preferred mode for an `addr_diff_vec` when the minimum and maximum offset are known. If you define this, it enables extra code in branch shortening to deal with `addr_diff_vec`. To make this work, you also have to define `INSN_ALIGN` and make the alignment for `addr_diff_vec` explicit. The *body* argument is provided so that the `offset_unsigned` and `scale` flags can be updated.

CASE_VECTOR_PC_RELATIVE [Macro]

Define this macro to be a C expression to indicate when jump-tables should contain relative addresses. You need not define this macro if jump-tables never contain relative addresses, or jump-tables should contain relative addresses only when `-fPIC` or `-fPIC` is in effect.

unsigned int TARGET_CASE_VALUES_THRESHOLD (void) [Target Hook]

This function return the smallest number of different values for which it is best to use a jump-table instead of a tree of conditional branches. The default is four for machines with a `casesi` instruction and five otherwise. This is best for most machines.

WORD_REGISTER_OPERATIONS [Macro]

Define this macro to 1 if operations between registers with integral mode smaller than a word are always performed on the entire register. To be more explicit, if you start with a pair of `word_mode` registers with known values and you do a subword, for example `QImode`, addition on the low part of the registers, then the compiler may consider that the result has a known value in `word_mode` too if the macro is defined to 1. Most RISC machines have this property and most CISC machines do not.

unsigned int TARGET_MIN_ARITHMETIC_PRECISION (void) [Target Hook]

On some RISC architectures with 64-bit registers, the processor also maintains 32-bit condition codes that make it possible to do real 32-bit arithmetic, although the operations are performed on the full registers.

On such architectures, defining this hook to 32 tells the compiler to try using 32-bit arithmetical operations setting the condition codes instead of doing full 64-bit arithmetic.

More generally, define this hook on RISC architectures if you want the compiler to try using arithmetical operations setting the condition codes with a precision lower than the word precision.

You need not define this hook if `WORD_REGISTER_OPERATIONS` is not defined to 1.

LOAD_EXTEND_OP (*mem_mode*) [Macro]

Define this macro to be a C expression indicating when insns that read memory in *mem_mode*, an integral mode narrower than a word, set the bits outside of *mem_mode* to be either the sign-extension or the zero-extension of the data read. Return `SIGN_EXTEND` for values of *mem_mode* for which the insn sign-extends, `ZERO_EXTEND` for which it zero-extends, and `UNKNOWN` for other modes.

This macro is not called with *mem_mode* non-integral or with a width greater than or equal to `BITS_PER_WORD`, so you may return any value in this case. Do not define this macro if it would always return `UNKNOWN`. On machines where this macro is defined, you will normally define it as the constant `SIGN_EXTEND` or `ZERO_EXTEND`.

You may return a non-`UNKNOWN` value even if for some hard registers the sign extension is not performed, if for the `REGNO_REG_CLASS` of these hard registers `TARGET_CAN_CHANGE_MODE_CLASS` returns false when the *from* mode is *mem_mode* and the *to* mode is any integral mode larger than this but not larger than *word_mode*.

You must return `UNKNOWN` if for some hard registers that allow this mode, `TARGET_CAN_CHANGE_MODE_CLASS` says that they cannot change to *word_mode*, but that they can change to another integral mode that is larger than *mem_mode* but still smaller than *word_mode*.

SHORT_IMMEDIATES_SIGN_EXTEND [Macro]

Define this macro to 1 if loading short immediate values into registers sign extends.

unsigned int TARGET_MIN_DIVISIONS_FOR_RECIP_MUL [Target Hook]
(*machine_mode mode*)

When `-ffast-math` is in effect, GCC tries to optimize divisions by the same divisor, by turning them into multiplications by the reciprocal. This target hook specifies the minimum number of divisions that should be there for GCC to perform the optimization for a variable of mode *mode*. The default implementation returns 3 if the machine has an instruction for the division, and 2 if it does not.

MOVE_MAX [Macro]

The maximum number of bytes that a single instruction can move quickly between memory and registers or between two memory locations.

MAX_MOVE_MAX [Macro]

The maximum number of bytes that a single instruction can move quickly between memory and registers or between two memory locations. If this is undefined, the default is `MOVE_MAX`. Otherwise, it is the constant value that is the largest value that `MOVE_MAX` can have at run-time.

SHIFT_COUNT_TRUNCATED [Macro]

A C expression that is nonzero if on this machine the number of bits actually used for the count of a shift operation is equal to the number of bits needed to represent the size of the object being shifted. When this macro is nonzero, the compiler will assume that it is safe to omit a sign-extend, zero-extend, and certain bitwise ‘and’ instructions that truncates the count of a shift operation. On machines that have instructions that act on bit-fields at variable positions, which may include ‘bit test’ instructions, a nonzero `SHIFT_COUNT_TRUNCATED` also enables deletion of truncations of the values that serve as arguments to bit-field instructions.

If both types of instructions truncate the count (for shifts) and position (for bit-field operations), or if no variable-position bit-field instructions exist, you should define this macro.

However, on some machines, such as the 80386 and the 680x0, truncation only applies to shift operations and not the (real or pretended) bit-field operations. Define `SHIFT_COUNT_TRUNCATED` to be zero on such machines. Instead, add patterns to the `md` file that include the implied truncation of the shift instructions.

You need not define this macro if it would always have the value of zero.

```
unsigned HOST_WIDE_INT TARGET_SHIFT_TRUNCATION_MASK      [Target Hook]
      (machine_mode mode)
```

This function describes how the standard shift patterns for *mode* deal with shifts by negative amounts or by more than the width of the mode. See [shift patterns], page 446.

On many machines, the shift patterns will apply a mask *m* to the shift count, meaning that a fixed-width shift of *x* by *y* is equivalent to an arbitrary-width shift of *x* by *y* & *m*. If this is true for mode *mode*, the function should return *m*, otherwise it should return 0. A return value of 0 indicates that no particular behavior is guaranteed.

Note that, unlike `SHIFT_COUNT_TRUNCATED`, this function does *not* apply to general shift rtxes; it applies only to instructions that are generated by the named shift patterns.

The default implementation of this function returns `GET_MODE_BITSIZE (mode) - 1` if `SHIFT_COUNT_TRUNCATED` and 0 otherwise. This definition is always safe, but if `SHIFT_COUNT_TRUNCATED` is false, and some shift patterns nevertheless truncate the shift count, you may get better code by overriding it.

```
bool TARGET_TRULY_NOOP_TRUNCATION (poly_uint64 outprec,    [Target Hook]
      poly_uint64 inprec)
```

This hook returns true if it is safe to “convert” a value of *inprec* bits to one of *outprec* bits (where *outprec* is smaller than *inprec*) by merely operating on it as if it had only *outprec* bits. The default returns true unconditionally, which is correct for most machines. When `TARGET_TRULY_NOOP_TRUNCATION` returns false, the machine description should provide a `trunc` optab to specify the RTL that performs the required truncation.

If `TARGET_MODES_TIEABLE_P` returns false for a pair of modes, suboptimal code can result if this hook returns true for the corresponding mode sizes. Making this hook return false in such cases may improve things.

```
int TARGET_MODE_REP_EXTENDED (scalar_int_mode mode,        [Target Hook]
      scalar_int_mode rep_mode)
```

The representation of an integral mode can be such that the values are always extended to a wider integral mode. Return `SIGN_EXTEND` if values of *mode* are represented in sign-extended form to *rep_mode*. Return `UNKNOWN` otherwise. (Currently, none of the targets use zero-extended representation this way so unlike `LOAD_EXTEND_OP`, `TARGET_MODE_REP_EXTENDED` is expected to return either `SIGN_EXTEND` or `UNKNOWN`. Also no target extends *mode* to *rep_mode* so that *rep_mode* is not the next widest integral mode and currently we take advantage of this fact.)

Similarly to `LOAD_EXTEND_OP` you may return a non-`UNKNOWN` value even if the extension is not performed on certain hard registers as long as for the `REGNO_REG_CLASS` of these hard registers `TARGET_CAN_CHANGE_MODE_CLASS` returns false.

Note that `TARGET_MODE_REP_EXTENDED` and `LOAD_EXTEND_OP` describe two related properties. If you define `TARGET_MODE_REP_EXTENDED (mode, word_mode)` you probably also want to define `LOAD_EXTEND_OP (mode)` to return the same type of extension. In order to enforce the representation of `mode`, `TARGET_TRULY_NOOP_TRUNCATION` should return false when truncating to `mode`.

bool TARGET_SETJMP_PRESERVES_NONVOLATILE_REGS_P (void) [Target Hook]

On some targets, it is assumed that the compiler will spill all pseudos that are live across a call to `setjmp`, while other targets treat `setjmp` calls as normal function calls.

This hook returns false if `setjmp` calls do not preserve all non-volatile registers so that gcc that must spill all pseudos that are live across `setjmp` calls. Define this to return true if the target does not need to spill all pseudos live across `setjmp` calls. The default implementation conservatively assumes all pseudos must be spilled across `setjmp` calls.

STORE_FLAG_VALUE [Macro]

A C expression describing the value returned by a comparison operator with an integral mode and stored by a store-flag instruction (`'cstoremode4'`) when the condition is true. This description must apply to *all* the `'cstoremode4'` patterns and all the comparison operators whose results have a `MODE_INT` mode.

A value of 1 or -1 means that the instruction implementing the comparison operator returns exactly 1 or -1 when the comparison is true and 0 when the comparison is false. Otherwise, the value indicates which bits of the result are guaranteed to be 1 when the comparison is true. This value is interpreted in the mode of the comparison operation, which is given by the mode of the first operand in the `'cstoremode4'` pattern. Either the low bit or the sign bit of `STORE_FLAG_VALUE` be on. Presently, only those bits are used by the compiler.

If `STORE_FLAG_VALUE` is neither 1 or -1 , the compiler will generate code that depends only on the specified bits. It can also replace comparison operators with equivalent operations if they cause the required bits to be set, even if the remaining bits are undefined. For example, on a machine whose comparison operators return an `SI` mode value and where `STORE_FLAG_VALUE` is defined as `'0x80000000'`, saying that just the sign bit is relevant, the expression

```
(ne:SI (and:SI x (const_int power-of-2)) (const_int 0))
```

can be converted to

```
(ashift:SI x (const_int n))
```

where n is the appropriate shift count to move the bit being tested into the sign bit.

There is no way to describe a machine that always sets the low-order bit for a true value, but does not guarantee the value of any other bits, but we do not know of any machine that has such an instruction. If you are trying to port GCC to such a machine, include an instruction to perform a logical-and of the result with 1 in the pattern for the comparison operators and let us know at gcc@gcc.gnu.org.

Often, a machine will have multiple instructions that obtain a value from a comparison (or the condition codes). Here are rules to guide the choice of value for `STORE_FLAG_VALUE`, and hence the instructions to be used:

- Use the shortest sequence that yields a valid definition for `STORE_FLAG_VALUE`. It is more efficient for the compiler to “normalize” the value (convert it to, e.g., 1 or 0) than for the comparison operators to do so because there may be opportunities to combine the normalization with other operations.
- For equal-length sequences, use a value of 1 or -1 , with -1 being slightly preferred on machines with expensive jumps and 1 preferred on other machines.
- As a second choice, choose a value of ‘0x80000001’ if instructions exist that set both the sign and low-order bits but do not define the others.
- Otherwise, use a value of ‘0x80000000’.

Many machines can produce both the value chosen for `STORE_FLAG_VALUE` and its negation in the same number of instructions. On those machines, you should also define a pattern for those cases, e.g., one matching

```
(set A (neg:m (ne:m B C)))
```

Some machines can also perform `and` or `plus` operations on condition code values with less instructions than the corresponding ‘`cstoremode4`’ insn followed by `and` or `plus`. On those machines, define the appropriate patterns. Use the names `incsc` and `decsc`, respectively, for the patterns which perform `plus` or `minus` operations on condition code values. See `rs6000.md` for some examples. The GNU Superoptimizer can be used to find such instruction sequences on other machines.

If this macro is not defined, the default value, 1, is used. You need not define `STORE_FLAG_VALUE` if the machine has no store-flag instructions, or if the value generated by these instructions is 1.

FLOAT_STORE_FLAG_VALUE (*mode*) [Macro]

A C expression that gives a nonzero `REAL_VALUE_TYPE` value that is returned when comparison operators with floating-point results are true. Define this macro on machines that have comparison operations that return floating-point values. If there are no such operations, do not define this macro.

VECTOR_STORE_FLAG_VALUE (*mode*) [Macro]

A C expression that gives an rtx representing the nonzero true element for vector comparisons. The returned rtx should be valid for the inner mode of *mode* which is guaranteed to be a vector mode. Define this macro on machines that have vector comparison operations that return a vector result. If there are no such operations, do not define this macro. Typically, this macro is defined as `const1_rtx` or `constm1_rtx`. This macro may return `NULL_RTX` to prevent the compiler optimizing such vector comparison operations for the given mode.

CLZ_DEFINED_VALUE_AT_ZERO (*mode*, *value*) [Macro]

CTZ_DEFINED_VALUE_AT_ZERO (*mode*, *value*) [Macro]

A C expression that indicates whether the architecture defines a value for `clz` or `ctz` with a zero operand. A result of 0 indicates the value is undefined. If the value is defined for only the RTL expression, the macro should evaluate to 1; if the value applies also to the corresponding optab entry (which is normally the case if it expands directly into the corresponding RTL), then the macro should evaluate to 2. In the cases where the value is defined, *value* should be set to this value.

If this macro is not defined, the value of `clz` or `ctz` at zero is assumed to be undefined. This macro must be defined if the target's expansion for `ffs` relies on a particular value to get correct results. Otherwise it is not necessary, though it may be used to optimize some corner cases, and to provide a default expansion for the `ffs` optab.

Note that regardless of this macro the “definedness” of `clz` and `ctz` at zero do *not* extend to the builtin functions visible to the user. Thus one may be free to adjust the value at will to match the target expansion of these operations without fear of breaking the API.

Pmode [Macro]

An alias for the machine mode for pointers. On most machines, define this to be the integer mode corresponding to the width of a hardware pointer; `SImode` on 32-bit machine or `DImode` on 64-bit machines. On some machines you must define this to be one of the partial integer modes, such as `PSImode`.

The width of `Pmode` must be at least as large as the value of `POINTER_SIZE`. If it is not equal, you must define the macro `POINTERS_EXTEND_UNSIGNED` to specify how pointers are extended to `Pmode`.

FUNCTION_MODE [Macro]

An alias for the machine mode used for memory references to functions being called, in `call` RTL expressions. On most CISC machines, where an instruction can begin at any byte address, this should be `QImode`. On most RISC machines, where all instructions have fixed size and alignment, this should be a mode with the same size and alignment as the machine instruction words - typically `SImode` or `HImode`.

STDC_0_IN_SYSTEM_HEADERS [Macro]

In normal operation, the preprocessor expands `__STDC__` to the constant 1, to signify that GCC conforms to ISO Standard C. On some hosts, like Solaris, the system compiler uses a different convention, where `__STDC__` is normally 0, but is 1 if the user specifies strict conformance to the C Standard.

Defining `STDC_0_IN_SYSTEM_HEADERS` makes GNU CPP follow the host convention when processing system header files, but when processing user files `__STDC__` will always expand to 1.

const char * TARGET_C_PREINCLUDE (void) [C Target Hook]

Define this hook to return the name of a header file to be included at the start of all compilations, as if it had been included with `#include <file>`. If this hook returns `NULL`, or is not defined, or the header is not found, or if the user specifies `-ffreestanding` or `-nostdinc`, no header is included.

This hook can be used together with a header provided by the system C library to implement ISO C requirements for certain macros to be predefined that describe properties of the whole implementation rather than just the compiler.

bool TARGET_CXX_IMPLICIT_EXTERN_C (const char*) [C Target Hook]

Define this hook to add target-specific C++ implicit extern C functions. If this function returns true for the name of a file-scope function, that function implicitly gets extern "C" linkage rather than whatever language linkage the declaration would normally have. An example of such function is `WinMain` on Win32 targets.

SYSTEM_IMPLICIT_EXTERN_C [Macro]

Define this macro if the system header files do not support C++. This macro handles system header files by pretending that system header files are enclosed in ‘extern "C" {...}’.

REGISTER_TARGET_PRAGMAS () [Macro]

Define this macro if you want to implement any target-specific pragmas. If defined, it is a C expression which makes a series of calls to `c_register_pragma` or `c_register_pragma_with_expansion` for each pragma. The macro may also do any setup required for the pragmas.

The primary reason to define this macro is to provide compatibility with other compilers for the same target. In general, we discourage definition of target-specific pragmas for GCC.

If the pragma can be implemented by attributes then you should consider defining the target hook ‘`TARGET_INSERT_ATTRIBUTES`’ as well.

Preprocessor macros that appear on pragma lines are not expanded. All ‘`#pragma`’ directives that do not match any registered pragma are silently ignored, unless the user specifies `-Wunknown-pragmas`.

void c_register_pragma (const char *space, const char [Function]

*name, void (*callback) (struct cpp_reader *))

void c_register_pragma_with_expansion (const char *space, [Function]

const char *name, void (*callback) (struct cpp_reader *))

Each call to `c_register_pragma` or `c_register_pragma_with_expansion` establishes one pragma. The *callback* routine will be called when the preprocessor encounters a pragma of the form

```
#pragma [space] name ...
```

space is the case-sensitive namespace of the pragma, or NULL to put the pragma in the global namespace. The callback routine receives *pfile* as its first argument, which can be passed on to `cpplib`’s functions if necessary. You can lex tokens after the *name* by calling `pragma_lex`. Tokens that are not read by the callback will be silently ignored. The end of the line is indicated by a token of type `CPP_EOF`. Macro expansion occurs on the arguments of pragmas registered with `c_register_pragma_with_expansion` but not on the arguments of pragmas registered with `c_register_pragma`.

Note that the use of `pragma_lex` is specific to the C and C++ compilers. It will not work in the Java or Fortran compilers, or any other language compilers for that matter. Thus if `pragma_lex` is going to be called from target-specific code, it must only be done so when building the C and C++ compilers. This can be done by defining the variables `c_target_objs` and `cxx_target_objs` in the target entry in the `config.gcc` file. These variables should name the target-specific, language-specific object file which contains the code that uses `pragma_lex`. Note it will also be necessary to add a rule to the makefile fragment pointed to by `tmake_file` that shows how to build this object file.

HANDLE_PRAGMA_PACK_WITH_EXPANSION [Macro]

Define this macro if macros should be expanded in the arguments of ‘`#pragma pack`’.

TARGET_DEFAULT_PACK_STRUCT [Macro]

If your target requires a structure packing default other than 0 (meaning the machine default), define this macro to the necessary value (in bytes). This must be a value that would also be valid to use with `#pragma pack()` (that is, a small power of two).

DOLLARS_IN_IDENTIFIERS [Macro]

Define this macro to control use of the character `'$'` in identifier names for the C family of languages. 0 means `'$'` is not allowed by default; 1 means it is allowed. 1 is the default; there is no need to define this macro in that case.

INSN_SETS_ARE_DELAYED (*insn*) [Macro]

Define this macro as a C expression that is nonzero if it is safe for the delay slot scheduler to place instructions in the delay slot of *insn*, even if they appear to use a resource set or clobbered in *insn*. *insn* is always a `jump_insn` or an `insn`; GCC knows that every `call_insn` has this behavior. On machines where some `insn` or `jump_insn` is really a function call and hence has this behavior, you should define this macro.

You need not define this macro if it would always return zero.

INSN_REFERENCES_ARE_DELAYED (*insn*) [Macro]

Define this macro as a C expression that is nonzero if it is safe for the delay slot scheduler to place instructions in the delay slot of *insn*, even if they appear to set or clobber a resource referenced in *insn*. *insn* is always a `jump_insn` or an `insn`. On machines where some `insn` or `jump_insn` is really a function call and its operands are registers whose use is actually in the subroutine it calls, you should define this macro. Doing so allows the delay slot scheduler to move instructions which copy arguments into the argument registers into the delay slot of *insn*.

You need not define this macro if it would always return zero.

MULTIPLE_SYMBOL_SPACES [Macro]

Define this macro as a C expression that is nonzero if, in some cases, global symbols from one translation unit may not be bound to undefined symbols in another translation unit without user intervention. For instance, under Microsoft Windows symbols must be explicitly imported from shared libraries (DLLs).

You need not define this macro if it would always evaluate to zero.

rtx_insn * TARGET_MD_ASM_ADJUST (vec<rtx>& *outputs*, [Target Hook]
 vec<rtx>& *inputs*, vec<machine_mode>& *input_modes*, vec<const
 char *>& *constraints*, vec<rtx>& *usess*, vec<rtx>& *clobbers*,
 HARD_REG_SET& *clobbered_regs*, location_t *loc*)

This target hook may add *clobbers* to *clobbers* and *clobbered_regs* for any hard regs the port wishes to automatically clobber for an asm. It can also add hard registers that are used by the asm to *usess*. The *outputs* and *inputs* may be inspected to avoid clobbering a register that is already used by the asm. *loc* is the source location of the asm.

It may modify the *outputs*, *inputs*, *input_modes*, and *constraints* as necessary for other pre-processing. In this case the return value is a sequence of insns to emit after the asm. Note that changes to *inputs* must be accompanied by the corresponding changes to *input_modes*.

MATH_LIBRARY [Macro]

Define this macro as a C string constant for the linker argument to link in the system math library, minus the initial `"-l"`, or `""` if the target does not have a separate math library.

You need only define this macro if the default of `"m"` is wrong.

LIBRARY_PATH_ENV [Macro]

Define this macro as a C string constant for the environment variable that specifies where the linker should look for libraries.

You need only define this macro if the default of `"LIBRARY_PATH"` is wrong.

TARGET_POSIX_IO [Macro]

Define this macro if the target supports the following POSIX file functions, access, mkdir and file locking with fcntl / F_SETLK. Defining **TARGET_POSIX_IO** will enable the test coverage code to use file locking when exiting a program, which avoids race conditions if the program has forked. It will also create directories at run-time for cross-profiling.

MAX_CONDITIONAL_EXECUTE [Macro]

A C expression for the maximum number of instructions to execute via conditional execution instructions instead of a branch. A value of **BRANCH_COST**+1 is the default.

IFCVT_MODIFY_TESTS (*ce_info*, *true_expr*, *false_expr*) [Macro]

Used if the target needs to perform machine-dependent modifications on the conditionals used for turning basic blocks into conditionally executed code. *ce_info* points to a data structure, `struct ce_if_block`, which contains information about the currently processed blocks. *true_expr* and *false_expr* are the tests that are used for converting the then-block and the else-block, respectively. Set either *true_expr* or *false_expr* to a null pointer if the tests cannot be converted.

IFCVT_MODIFY_MULTIPLE_TESTS (*ce_info*, *bb*, *true_expr*, *false_expr*) [Macro]

Like **IFCVT_MODIFY_TESTS**, but used when converting more complicated if-statements into conditions combined by **and** and **or** operations. *bb* contains the basic block that contains the test that is currently being processed and about to be turned into a condition.

IFCVT_MODIFY_INSN (*ce_info*, *pattern*, *insn*) [Macro]

A C expression to modify the *PATTERN* of an *INSN* that is to be converted to conditional execution format. *ce_info* points to a data structure, `struct ce_if_block`, which contains information about the currently processed blocks.

IFCVT_MODIFY_FINAL (*ce_info*) [Macro]

A C expression to perform any final machine dependent modifications in converting code to conditional execution. The involved basic blocks can be found in the `struct ce_if_block` structure that is pointed to by *ce_info*.

IFCVT_MODIFY_CANCEL (*ce_info*) [Macro]

A C expression to cancel any machine dependent modifications in converting code to conditional execution. The involved basic blocks can be found in the `struct ce_if_block` structure that is pointed to by *ce_info*.

IFCVT_MACHDEP_INIT (*ce_info*) [Macro]

A C expression to initialize any machine specific data for if-conversion of the if-block in the `struct ce_if_block` structure that is pointed to by *ce_info*.

bool TARGET_USE_LATE_PROLOGUE_EPILOGUE () [Target Hook]

Return true if the current function's prologue and epilogue should be emitted late in the pass pipeline, instead of at the usual point.

Normally, the prologue and epilogue sequences are introduced soon after register allocation is complete. The advantage of this approach is that it allows the prologue and epilogue instructions to be optimized and scheduled with other code in the function. However, some targets require the prologue and epilogue to be the first and last sequences executed by the function, with no variation allowed. This hook should return true on such targets.

The default implementation returns false, which is correct for most targets. The hook should only return true if there is a specific target limitation that cannot be described in RTL. For example, the hook might return true if the prologue and epilogue need to switch between instruction sets.

void TARGET_EMIT_EPILOGUE_FOR_SIBCALL (*rtx_call_insn* **call*) [Target Hook]

If defined, this hook emits an epilogue sequence for sibling (tail) call instruction *call*. Another way of providing epilogues for sibling calls is to define the `sibcall_epilogue` instruction pattern; the main advantage of this hook over the pattern is that it has access to the call instruction.

void TARGET_MACHINE_DEPENDENT_REORG (*void*) [Target Hook]

If non-null, this hook performs a target-specific pass over the instruction stream. The compiler will run it at all optimization levels, just before the point at which it normally does delayed-branch scheduling.

The exact purpose of the hook varies from target to target. Some use it to do transformations that are necessary for correctness, such as laying out in-function constant pools or avoiding hardware hazards. Others use it as an opportunity to do some machine-dependent optimizations.

You need not implement the hook if it has nothing to do. The default definition is null.

void TARGET_INIT_BUILTINS (*void*) [Target Hook]

Define this hook if you have any machine-specific built-in functions that need to be defined. It should be a function that performs the necessary setup.

Machine specific built-in functions can be useful to expand special machine instructions that would otherwise not normally be generated because they have no equivalent in the source language (for example, SIMD vector instructions or prefetch instructions).

To create a built-in function, call the function `lang_hooks.builtin_function` which is defined by the language front end. You can use any type nodes set up by `build_common_tree_nodes`; only language front ends that use those two functions will call 'TARGET_INIT_BUILTINS'.

tree TARGET_BUILTIN_DECL (unsigned code, bool initialize_p) [Target Hook]

Define this hook if you have any machine-specific built-in functions that need to be defined. It should be a function that returns the builtin function declaration for the builtin function code *code*. If there is no such builtin and it cannot be initialized at this time if *initialize_p* is true the function should return `NULL_TREE`. If *code* is out of range the function should return `error_mark_node`.

rtx TARGET_EXPAND_BUILTIN (tree exp, rtx target, rtx subtarget, machine_mode mode, int ignore) [Target Hook]

Expand a call to a machine specific built-in function that was set up by ‘TARGET_INIT_BUILTINS’. *exp* is the expression for the function call; the result should go to *target* if that is convenient, and have mode *mode* if that is convenient. *subtarget* may be used as the target for computing one of *exp*’s operands. *ignore* is nonzero if the value is to be ignored. This function should return the result of the call to the built-in function.

tree TARGET_RESOLVE_OVERLOADED_BUILTIN (location_t loc, tree fndecl, void *arglist, bool complain) [Target Hook]

Select a replacement for a machine specific built-in function that was set up by ‘TARGET_INIT_BUILTINS’. This is done *before* regular type checking, and so allows the target to implement a crude form of function overloading. *fndecl* is the declaration of the built-in function. *arglist* is the list of arguments passed to the built-in function. The result is a complete expression that implements the operation, usually another `CALL_EXPR`. *arglist* really has type ‘`VEC(tree,gc)*`’. *complain* is a boolean indicating whether invalid operations should emit errors. This is set to `false` when the C++ templating context expects that errors should not be emitted (i.e. SFINAE).

bool TARGET_CHECK_BUILTIN_CALL (location_t loc, vec<location_t> arg_loc, tree fndecl, tree orig_fndecl, unsigned int nargs, tree *args, bool complain) [Target Hook]

Perform semantic checking on a call to a machine-specific built-in function after its arguments have been constrained to the function signature. Return true if the call is valid, otherwise report an error and return false.

This hook is called after `TARGET_RESOLVE_OVERLOADED_BUILTIN`. The call was originally to built-in function *orig_fndecl*, but after the optional `TARGET_RESOLVE_OVERLOADED_BUILTIN` step is now to built-in function *fndecl*. *loc* is the location of the call and *args* is an array of function arguments, of which there are *nargs*. *arg_loc* specifies the location of each argument. *complain* is a boolean indicating whether invalid arguments should emit errors. This is set to `false` when the C++ templating context expects that errors should not be emitted (i.e. SFINAE).

tree TARGET_FOLD_BUILTIN (tree fndecl, int n_args, tree *argp, bool ignore) [Target Hook]

Fold a call to a machine specific built-in function that was set up by ‘TARGET_INIT_BUILTINS’. *fndecl* is the declaration of the built-in function. *n_args* is the number of arguments passed to the function; the arguments themselves are pointed to by *argp*. The result is another tree, valid for both GIMPLE and

support for decrement and branch, it may have to move IV value from hardware count register to general purpose register while doloop IV candidate is used for generic IV uses. It probably takes expensive penalty. This hook allows target owners to define the cost for this especially for generic IV uses. The default value is zero.

`int64_t TARGET_DOLOOP_COST_FOR_ADDRESS` [Target Hook]

One IV candidate dedicated for doloop is introduced in IVOPTs, we can calculate the computation cost of adopting it to any address IV use by function `get_computation_cost` as before. But for targets which have hardware count register support for decrement and branch, it may have to move IV value from hardware count register to general purpose register while doloop IV candidate is used for address IV uses. It probably takes expensive penalty. This hook allows target owners to define the cost for this especially for address IV uses. The default value is zero.

`bool TARGET_CAN_USE_DOLOOP_P (const widest_int
 &iterations, const widest_int &iterations_max, unsigned int
 loop_depth, bool entered_at_top)` [Target Hook]

Return true if it is possible to use low-overhead loops (`doloop_end` and `doloop_begin`) for a particular loop. *iterations* gives the exact number of iterations, or 0 if not known. *iterations_max* gives the maximum number of iterations, or 0 if not known. *loop_depth* is the nesting depth of the loop, with 1 for innermost loops, 2 for loops that contain innermost loops, and so on. *entered_at_top* is true if the loop is only entered from the top.

This hook is only used if `doloop_end` is available. The default implementation returns true. You can use `can_use_doloop_if_innermost` if the loop must be the innermost, and if there are no other restrictions.

`const char * TARGET_INVALID_WITHIN_DOLOOP (const
 rtx_insn *insn)` [Target Hook]

Take an instruction in *insn* and return NULL if it is valid within a low-overhead loop, otherwise return a string explaining why doloop could not be applied.

Many targets use special registers for low-overhead looping. For any instruction that clobbers these this function should return a string indicating the reason why the doloop could not be applied. By default, the RTL loop optimizer does not use a present doloop pattern for loops containing function calls or branch on table instructions.

`machine_mode TARGET_PREFERRED_DOLOOP_MODE (machine_mode
 mode)` [Target Hook]

This hook takes a *mode* for a doloop IV, where *mode* is the original mode for the operation. If the target prefers an alternate *mode* for the operation, then this hook should return that *mode*; otherwise the original *mode* should be returned. For example, on a 64-bit target, `DI`mode might be preferred over `SI`mode. Both the original and the returned modes should be `MODE_INT`.

`bool TARGET_LEGITIMATE_COMBINED_INSN (rtx_insn *insn)` [Target Hook]

Take an instruction in *insn* and return `false` if the instruction is not appropriate as a combination of two or more instructions. The default is to accept all instructions.

bool TARGET_CAN_FOLLOW_JUMP (const rtx_insn **follower*, [Target Hook]
const rtx_insn **followee*)

FOLLOWER and FOLLOWEE are JUMP_INSN instructions; return true if FOLLOWER may be modified to follow FOLLOWEE; false, if it can't. For example, on some targets, certain kinds of branches can't be made to follow through a hot/cold partitioning.

bool TARGET_COMMUTATIVE_P (const_rtx *x*, int *outer_code*) [Target Hook]

This target hook returns true if *x* is considered to be commutative. Usually, this is just COMMUTATIVE_P (*x*), but the HP PA doesn't consider PLUS to be commutative inside a MEM. *outer_code* is the rtx code of the enclosing rtl, if known, otherwise it is UNKNOWN.

rtx TARGET_ALLOCATE_INITIAL_VALUE (rtx *hard_reg*) [Target Hook]

When the initial value of a hard register has been copied in a pseudo register, it is often not necessary to actually allocate another register to this pseudo register, because the original hard register or a stack slot it has been saved into can be used. TARGET_ALLOCATE_INITIAL_VALUE is called at the start of register allocation once for each hard register that had its initial value copied by using `get_func_hard_reg_initial_val` or `get_hard_reg_initial_val`. Possible values are NULL_RTX, if you don't want to do any special allocation, a REG rtx—that would typically be the hard register itself, if it is known not to be clobbered—or a MEM. If you are returning a MEM, this is only a hint for the allocator; it might decide to use another register anyways. You may use `current_function_is_leaf` or `REG_N_SETS` in the hook to determine if the hard register in question will not be clobbered. The default value of this hook is NULL, which disables any special allocation.

int TARGET_UNSPEC_MAY_TRAP_P (const_rtx *x*, unsigned [Target Hook]
flags)

This target hook returns nonzero if *x*, an `unspec` might cause a trap. Targets can use this hook to enhance precision of analysis for `unspec` operations. You may call `may_trap_p_1` to analyze inner elements of *x* in which case *flags* should be passed along.

void TARGET_SET_CURRENT_FUNCTION (tree *decl*) [Target Hook]

The compiler invokes this hook whenever it changes its current function context (`cfun`). You can define this function if the back end needs to perform any initialization or reset actions on a per-function basis. For example, it may be used to implement function attributes that affect register usage or code generation patterns. The argument *decl* is the declaration for the new function context, and may be null to indicate that the compiler has left a function context and is returning to processing at the top level. The default hook function does nothing.

GCC sets `cfun` to a dummy function context during initialization of some parts of the back end. The hook function is not invoked in this situation; you need not worry about the hook being invoked recursively, or when the back end is in a partially-initialized state. `cfun` might be NULL to indicate processing at top level, outside of any function scope.

TARGET_OBJECT_SUFFIX [Macro]

Define this macro to be a C string representing the suffix for object files on your target machine. If you do not define this macro, GCC will use `‘.o’` as the suffix for object files.

TARGET_EXECUTABLE_SUFFIX [Macro]

Define this macro to be a C string representing the suffix to be automatically added to executable files on your target machine. If you do not define this macro, GCC will use the null string as the suffix for executable files.

COLLECT_EXPORT_LIST [Macro]

If defined, `collect2` will scan the individual object files specified on its command line and create an export list for the linker. Define this macro for systems like AIX, where the linker discards object files that are not referenced from `main` and uses export lists.

bool TARGET_CANNOT_MODIFY_JUMPS_P (void) [Target Hook]

This target hook returns `true` past the point in which new jump instructions could be created. On machines that require a register for every jump such as the SHmedia ISA of SH5, this point would typically be `reload`, so this target hook should be defined to a function such as:

```
static bool
cannot_modify_jumps_past_reload_p ()
{
    return (reload_completed || reload_in_progress);
}
```

bool TARGET_HAVE_CONDITIONAL_EXECUTION (void) [Target Hook]

This target hook returns `true` if the target supports conditional execution. This target hook is required only when the target has several different modes and they have different conditional execution capability, such as ARM.

rtx TARGET_GEN_CCMP_FIRST (rtx_insn **prep_seq, rtx_insn **gen_seq, rtx_code code, tree op0, tree op1) [Target Hook]

This function prepares to emit a comparison insn for the first compare in a sequence of conditional comparisons. It returns an appropriate comparison with `CC` for passing to `gen_ccmp_next` or `cbranch_optab`. The insns to prepare the compare are saved in `prep_seq` and the compare insns are saved in `gen_seq`. They will be emitted when all the compares in the conditional comparison are generated without error. `code` is the `rtx_code` of the compare for `op0` and `op1`.

rtx TARGET_GEN_CCMP_NEXT (rtx_insn **prep_seq, rtx_insn **gen_seq, rtx prev, rtx_code cmp_code, tree op0, tree op1, rtx_code bit_code) [Target Hook]

This function prepares to emit a conditional comparison within a sequence of conditional comparisons. It returns an appropriate comparison with `CC` for passing to `gen_ccmp_next` or `cbranch_optab`. The insns to prepare the compare are saved in `prep_seq` and the compare insns are saved in `gen_seq`. They will be emitted when all the compares in the conditional comparison are generated without error. The `prev` expression is the result of a prior call to `gen_ccmp_first` or `gen_ccmp_next`. It may

return `NULL` if the combination of *prev* and this comparison is not supported, otherwise the result must be appropriate for passing to `gen_ccmp_next` or `cbranch_optab`. *code* is the `rtx_code` of the compare for *op0* and *op1*. *bit_code* is `AND` or `IOR`, which is the op on the compares.

bool TARGET_HAVE_CCMP (void) [Target Hook]

This target hook returns true if the target supports conditional compare. This target hook is required only when the `ccmp` support is conditionally enabled, such as in response to command-line flags. The default implementation returns true iff `TARGET_GEN_CCMP_FIRST` is defined.

**unsigned TARGET_LOOP_UNROLL_ADJUST (unsigned *nunroll*,
class loop **loop*)** [Target Hook]

This target hook returns a new value for the number of times *loop* should be unrolled. The parameter *nunroll* is the number of times the loop is to be unrolled. The parameter *loop* is a pointer to the loop, which is going to be checked for unrolling. This target hook is required only when the target has special constraints like maximum number of memory accesses.

POWI_MAX_MULTS [Macro]

If defined, this macro is interpreted as a signed integer C expression that specifies the maximum number of floating point multiplications that should be emitted when expanding exponentiation by an integer constant inline. When this value is defined, exponentiation requiring more than this number of multiplications is implemented by calling the system library's `pow`, `powf` or `powl` routines. The default value places no upper bound on the multiplication count.

**void TARGET_EXTRA_INCLUDES (const char **sysroot*, const char
iprefix*, int *stdinc*) [Macro]

This target hook should register any extra include files for the target. The parameter *stdinc* indicates if normal include files are present. The parameter *sysroot* is the system root directory. The parameter *iprefix* is the prefix for the gcc directory.

**void TARGET_EXTRA_PRE_INCLUDES (const char **sysroot*, const
char **iprefix*, int *stdinc*)** [Macro]

This target hook should register any extra include files for the target before any standard headers. The parameter *stdinc* indicates if normal include files are present. The parameter *sysroot* is the system root directory. The parameter *iprefix* is the prefix for the gcc directory.

void TARGET_OPTF (char **path*) [Macro]

This target hook should register special include paths for the target. The parameter *path* is the include to register. On Darwin systems, this is used for Framework includes, which have semantics that are different from `-I`.

bool TARGET_USE_LOCAL_THUNK_ALIAS_P (tree *fndekl*) [Macro]

This target macro returns `true` if it is safe to use a local alias for a virtual function *fndekl* when constructing thunks, `false` otherwise. By default, the macro returns `true` for all functions, if a target supports aliases (i.e. defines `ASM_OUTPUT_DEF`), `false` otherwise,

TARGET_FORMAT_TYPES [Macro]

If defined, this macro is the name of a global variable containing target-specific format checking information for the `-Wformat` option. The default is to have no target-specific format checks.

TARGET_N_FORMAT_TYPES [Macro]

If defined, this macro is the number of entries in `TARGET_FORMAT_TYPES`.

TARGET_OVERRIDES_FORMAT_ATTRIBUTES [Macro]

If defined, this macro is the name of a global variable containing target-specific format overrides for the `-Wformat` option. The default is to have no target-specific format overrides. If defined, `TARGET_FORMAT_TYPES` and `TARGET_OVERRIDES_FORMAT_ATTRIBUTES_COUNT` must be defined, too.

TARGET_OVERRIDES_FORMAT_ATTRIBUTES_COUNT [Macro]

If defined, this macro specifies the number of entries in `TARGET_OVERRIDES_FORMAT_ATTRIBUTES`.

TARGET_OVERRIDES_FORMAT_INIT [Macro]

If defined, this macro specifies the optional initialization routine for target specific customizations of the system `printf` and `scanf` formatter settings.

`const char * TARGET_INVALID_ARG_FOR_UNPROTOTYPED_FN` [Target Hook]

(`const_tree typelist`, `const_tree funcdecl`, `const_tree val`)

If defined, this macro returns the diagnostic message when it is illegal to pass argument `val` to function `funcdecl` with prototype `typelist`.

`const char * TARGET_INVALID_CONVERSION (const_tree` [Target Hook]

`fromtype`, `const_tree totype`)

If defined, this macro returns the diagnostic message when it is invalid to convert from `fromtype` to `totype`, or NULL if validity should be determined by the front end.

`const char * TARGET_INVALID_UNARY_OP (int op,` [Target Hook]

`const_tree type`)

If defined, this macro returns the diagnostic message when it is invalid to apply operation `op` (where unary plus is denoted by `CONVERT_EXPR`) to an operand of type `type`, or NULL if validity should be determined by the front end.

`const char * TARGET_INVALID_BINARY_OP (int op,` [Target Hook]

`const_tree type1`, `const_tree type2`)

If defined, this macro returns the diagnostic message when it is invalid to apply operation `op` to operands of types `type1` and `type2`, or NULL if validity should be determined by the front end.

`tree TARGET_PROMOTED_TYPE (const_tree type)` [Target Hook]

If defined, this target hook returns the type to which values of `type` should be promoted when they appear in expressions, analogous to the integer promotions, or `NULL_TREE` to use the front end's normal promotion rules. This hook is useful when there are target-specific types with special promotion rules. This is currently used only by the C and C++ front ends.

tree TARGET_CONVERT_TO_TYPE (tree *type*, tree *expr*) [Target Hook]

If defined, this hook returns the result of converting *expr* to *type*. It should return the converted expression, or `NULL_TREE` to apply the front end's normal conversion rules. This hook is useful when there are target-specific types with special conversion rules. This is currently used only by the C and C++ front ends.

bool TARGET_VERIFY_TYPE_CONTEXT (location_t *loc*, [Target Hook]
type_context_kind *context*, const_tree *type*, bool *silent_p*)

If defined, this hook returns false if there is a target-specific reason why type *type* cannot be used in the source language context described by *context*. When *silent_p* is false, the hook also reports an error against *loc* for invalid uses of *type*.

Calls to this hook should be made through the global function `verify_type_context`, which makes the *silent_p* parameter default to false and also handles `error_mark_node`.

The default implementation always returns true.

OBJC_JBLEN [Macro]

This macro determines the size of the objective C jump buffer for the NeXT runtime. By default, `OBJC_JBLEN` is defined to an innocuous value.

LIBGCC2_UNWIND_ATTRIBUTE [Macro]

Define this macro if any target-specific attributes need to be attached to the functions in `libgcc` that provide low-level support for call stack unwinding. It is used in declarations in `unwind-generic.h` and the associated definitions of those functions.

void TARGET_UPDATE_STACK_BOUNDARY (void) [Target Hook]

Define this macro to update the current function stack boundary if necessary.

rtx TARGET_GET_DRAP_RTX (void) [Target Hook]

This hook should return an rtx for Dynamic Realign Argument Pointer (DRAP) if a different argument pointer register is needed to access the function's argument list due to stack realignment. Return `NULL` if no DRAP is needed.

HARD_REG_SET TARGET_ZERO_CALL_USED_REGS (HARD_REG_SET [Target Hook]
selected_regs)

This target hook emits instructions to zero the subset of *selected_regs* that could conceivably contain values that are useful to an attacker. Return the set of registers that were actually cleared.

For most targets, the returned set of registers is a subset of *selected_regs*, however, for some of the targets (for example MIPS), clearing some registers that are in the *selected_regs* requires clearing other call used registers that are not in the *selected_regs*, under such situation, the returned set of registers must be a subset of all call used registers.

The default implementation uses normal move instructions to zero all the registers in *selected_regs*. Define this hook if the target has more efficient ways of zeroing certain registers, or if you believe that certain registers would never contain values that are useful to an attacker.

bool TARGET_ALLOCATE_STACK_SLOTS_FOR_ARGS (void) [Target Hook]

When optimization is disabled, this hook indicates whether or not arguments should be allocated to stack slots. Normally, GCC allocates stack slots for arguments when not optimizing in order to make debugging easier. However, when a function is declared with `__attribute__((naked))`, there is no stack frame, and the compiler cannot safely move arguments from the registers in which they are passed to the stack. Therefore, this hook should return true in general, but false for naked functions. The default implementation always returns true.

unsigned HOST_WIDE_INT TARGET_CONST_ANCHOR [Target Hook]

On some architectures it can take multiple instructions to synthesize a constant. If there is another constant already in a register that is close enough in value then it is preferable that the new constant is computed from this register using immediate addition or subtraction. We accomplish this through CSE. Besides the value of the constant we also add a lower and an upper constant anchor to the available expressions. These are then queried when encountering new constants. The anchors are computed by rounding the constant up and down to a multiple of the value of `TARGET_CONST_ANCHOR`. `TARGET_CONST_ANCHOR` should be the maximum positive value accepted by immediate-add plus one. We currently assume that the value of `TARGET_CONST_ANCHOR` is a power of 2. For example, on MIPS, where add-immediate takes a 16-bit signed value, `TARGET_CONST_ANCHOR` is set to '0x8000'. The default value is zero, which disables this optimization.

unsigned HOST_WIDE_INT TARGET_ASAN_SHADOW_OFFSET (void) [Target Hook]

Return the offset bitwise ored into shifted address to get corresponding Address Sanitizer shadow memory address. NULL if Address Sanitizer is not supported by the target. May return 0 if Address Sanitizer is not supported or using dynamic shadow offset by a subtarget.

bool TARGET_ASAN_DYNAMIC_SHADOW_OFFSET_P (void) [Target Hook]

Return true if asan should use dynamic shadow offset.

unsigned HOST_WIDE_INT TARGET_MEMMODEL_CHECK (unsigned HOST_WIDE_INT val) [Target Hook]

Validate target specific memory model mask bits. When NULL no target specific memory model bits are allowed.

unsigned char TARGET_ATOMIC_TEST_AND_SET_TRUEVAL [Target Hook]

This value should be set if the result written by `atomic_test_and_set` is not exactly 1, i.e. the `bool true`.

bool TARGET_HAS_IFUNC_P (void) [Target Hook]

It returns true if the target supports GNU indirect functions. The support includes the assembler, linker and dynamic linker. The default value of this hook is based on target's libc.

bool TARGET_IFUNC_REF_LOCAL_OK (void) [Target Hook]

Return true if it is OK to reference indirect function resolvers locally. The default is to return false.

unsigned int TARGET_ATOMIC_ALIGN_FOR_MODE (*machine_mode* *mode*) [Target Hook]

If defined, this function returns an appropriate alignment in bits for an atomic object of machine-mode *mode*. If 0 is returned then the default alignment for the specified mode is used.

void TARGET_ATOMIC_ASSIGN_EXPAND_FENV (*tree* **hold*, *tree* **clear*, *tree* **update*) [Target Hook]

ISO C11 requires atomic compound assignments that may raise floating-point exceptions to raise exceptions corresponding to the arithmetic operation whose result was successfully stored in a compare-and-exchange sequence. This requires code equivalent to calls to `feholdexcept`, `feclearexcept` and `feupdateenv` to be generated at appropriate points in the compare-and-exchange sequence. This hook should set **hold* to an expression equivalent to the call to `feholdexcept`, **clear* to an expression equivalent to the call to `feclearexcept` and **update* to an expression equivalent to the call to `feupdateenv`. The three expressions are `NULL_TREE` on entry to the hook and may be left as `NULL_TREE` if no code is required in a particular place. The default implementation leaves all three expressions as `NULL_TREE`. The `__atomic_feraiseexcept` function from `libatomic` may be of use as part of the code generated in **update*.

void TARGET_RECORD_OFFLOAD_SYMBOL (*tree*) [Target Hook]

Used when offloaded functions are seen in the compilation unit and no named sections are available. It is called once for each symbol that must be recorded in the offload function and variable table.

char * TARGET_OFFLOAD_OPTIONS (*void*) [Target Hook]

Used when writing out the list of options into an LTO file. It should translate any relevant target-specific options (such as the ABI in use) into one of the `-foffload` options that exist as a common interface to express such options. It should return a string containing these options, separated by spaces, which the caller will free.

TARGET_SUPPORTS_WIDE_INT [Macro]

On older ports, large integers are stored in `CONST_DOUBLE` rtl objects. Newer ports define `TARGET_SUPPORTS_WIDE_INT` to be nonzero to indicate that large integers are stored in `CONST_WIDE_INT` rtl objects. The `CONST_WIDE_INT` allows very large integer constants to be represented. `CONST_DOUBLE` is limited to twice the size of the host's `HOST_WIDE_INT` representation.

Converting a port mostly requires looking for the places where `CONST_DOUBLES` are used with `VOIDmode` and replacing that code with code that accesses `CONST_WIDE_INTs`. “`grep -i const_double`” at the port level gets you to 95% of the changes that need to be made. There are a few places that require a deeper look.

- There is no equivalent to `hval` and `lval` for `CONST_WIDE_INTs`. This would be difficult to express in the md language since there are a variable number of elements.

Most ports only check that `hval` is either 0 or -1 to see if the value is small. As mentioned above, this will no longer be necessary since small constants are always `CONST_INT`. Of course there are still a few exceptions, the alpha's constraint used

by the zap instruction certainly requires careful examination by C code. However, all the current code does is pass the hval and lval to C code, so evolving the c code to look at the `CONST_WIDE_INT` is not really a large change.

- Because there is no standard template that ports use to materialize constants, there is likely to be some futzing that is unique to each port in this code.
- The rtx costs may have to be adjusted to properly account for larger constants that are represented as `CONST_WIDE_INT`.

All and all it does not take long to convert ports that the maintainer is familiar with.

bool TARGET_HAVE_SPECULATION_SAFE_VALUE (bool *active*) [Target Hook]

This hook is used to determine the level of target support for `__builtin_speculation_safe_value`. If called with an argument of false, it returns true if the target has been modified to support this builtin. If called with an argument of true, it returns true if the target requires active mitigation execution might be speculative. The default implementation returns false if the target does not define a pattern named `speculation_barrier`. Else it returns true for the first case and whether the pattern is enabled for the current compilation for the second case.

For targets that have no processors that can execute instructions speculatively an alternative implementation of this hook is available: simply redefine this hook to `speculation_safe_value_not_needed` along with your other target hooks.

rtx TARGET_SPECULATION_SAFE_VALUE (machine_mode *mode*, rtx *result*, rtx *val*, rtx *failval*) [Target Hook]

This target hook can be used to generate a target-specific code sequence that implements the `__builtin_speculation_safe_value` built-in function. The function must always return *val* in *result* in mode *mode* when the cpu is not executing speculatively, but must never return that when speculating until it is known that the speculation will not be unwound. The hook supports two primary mechanisms for implementing the requirements. The first is to emit a speculation barrier which forces the processor to wait until all prior speculative operations have been resolved; the second is to use a target-specific mechanism that can track the speculation state and to return *failval* if it can determine that speculation must be unwound at a later time.

The default implementation simply copies *val* to *result* and emits a `speculation_barrier` instruction if that is defined.

void TARGET_RUN_TARGET_SELFTESTS (void) [Target Hook]

If selftests are enabled, run any selftests for this target.

bool TARGET_MEMTAG_CAN_TAG_ADDRESSES () [Target Hook]

True if the backend architecture naturally supports ignoring some region of pointers. This feature means that `-fsanitize=hwaddress` can work.

At preset, this feature does not support address spaces. It also requires `Pmode` to be the same as `ptr_mode`.

uint8_t TARGET_MEMTAG_TAG_BITSIZE () [Target Hook]

Return the size of a tag (in bits) for this platform.

The default returns 8.

`uint8_t TARGET_MEMTAG_GRANULE_SIZE ()` [Target Hook]

Return the size in real memory that each byte in shadow memory refers to. I.e. if a variable is X bytes long in memory, then this hook should return the value Y such that the tag in shadow memory spans X/Y bytes.

Most variables will need to be aligned to this amount since two variables that are neighbors in memory and share a tag granule would need to share the same tag.

The default returns 16.

`rtx TARGET_MEMTAG_INSERT_RANDOM_TAG (rtx untagged, rtx target)` [Target Hook]

Return an RTX representing the value of *untagged* but with a (possibly) random tag in it. Put that value into *target* if it is convenient to do so. This function is used to generate a tagged base for the current stack frame. It is also used by memtag-stack sanitizer to emit specific memory tagging instructions.

`rtx TARGET_MEMTAG_ADD_TAG (rtx base, poly_int64 addr_offset, uint8_t tag_offset)` [Target Hook]

Return an RTX that represents the result of adding *addr_offset* to the address in pointer *base* and *tag_offset* to the tag in pointer *base*. The resulting RTX must either be a valid memory address or be able to get put into an operand with `force_operand`.

Unlike other memtag hooks, this must return an expression and not emit any RTL. In the case of memtag-stack sanitizer, this constraint is not enforced.

`rtx TARGET_MEMTAG_SET_TAG (rtx untagged_base, rtx tag, rtx target)` [Target Hook]

Return an RTX representing *untagged_base* but with the tag *tag*. Try and store this in *target* if convenient. *untagged_base* is required to have a zero tag when this hook is called. The default of this hook is to set the top byte of *untagged_base* to *tag*.

`rtx TARGET_MEMTAG_EXTRACT_TAG (rtx tagged_pointer, rtx target)` [Target Hook]

Return an RTX representing the tag stored in *tagged_pointer*. Store the result in *target* if it is convenient. The default represents the top byte of the original pointer. In the case of memtag-stack sanitizer for targets that can process tagged pointers (i.e. AArch64), this hook can return a tagged pointer.

`rtx TARGET_MEMTAG_UNTAGGED_POINTER (rtx tagged_pointer, rtx target)` [Target Hook]

Return an RTX representing *tagged_pointer* with its tag set to zero. Store the result in *target* if convenient. The default clears the top byte of the original pointer.

`bool TARGET_HAVE_SHADOW_CALL_STACK` [Target Hook]

This value is true if the target platform supports `-fsanitize=shadow-call-stack`. The default value is false.

`bool TARGET_HAVE_LIBATOMIC` [Target Hook]

This value is true if the target platform supports libatomic. The default value is false.

`const char * TARGET_DOCUMENTATION_NAME` [Target Hook]

If non-NULL, this value is a string used for locating target-specific documentation for this target. The default value is NULL.

18 Host Configuration

Most details about the machine and system on which the compiler is actually running are detected by the `configure` script. Some things are impossible for `configure` to detect; these are described in two ways, either by macros defined in a file named `xm-machine.h` or by hook functions in the file specified by the `out.host-hook.obj` variable in `config.gcc`. (The intention is that very few hosts will need a header file but nearly every fully supported host will need to override some hooks.)

If you need to define only a few macros, and they have simple definitions, consider using the `xm_defines` variable in your `config.gcc` entry instead of creating a host configuration header. See Section 5.3.2.2 [System Config], page 67.

18.1 Host Common

Some things are just not portable, even between similar operating systems, and are too difficult for `autoconf` to detect. They get implemented using hook functions in the file specified by the `host-hook.obj` variable in `config.gcc`.

void HOST_HOOKS_EXTRA_SIGNALS (void) [Host Hook]
 This host hook is used to set up handling for extra signals. The most common thing to do in this hook is to detect stack overflow.

void * HOST_HOOKS_GT_PCH_GET_ADDRESS (size_t size, int fd) [Host Hook]
 This host hook returns the address of some space that is likely to be free in some subsequent invocation of the compiler. We intend to load the PCH data at this address such that the data need not be relocated. The area should be able to hold *size* bytes. If the host uses `mmap`, *fd* is an open file descriptor that can be used for probing.

int HOST_HOOKS_GT_PCH_USE_ADDRESS (void * address, size_t size, int fd, size_t offset) [Host Hook]
 This host hook is called when a PCH file is about to be loaded. We want to load *size* bytes from *fd* at *offset* into memory at *address*. The given address will be the result of a previous invocation of `HOST_HOOKS_GT_PCH_GET_ADDRESS`. Return `-1` if we couldn't allocate *size* bytes at *address*. Return `0` if the memory is allocated but the data is not loaded. Return `1` if the hook has performed everything.

If the implementation uses reserved address space, free any reserved space beyond *size*, regardless of the return value. If no PCH will be loaded, this hook may be called with *size* zero, in which case all reserved address space should be freed.

Do not try to handle values of *address* that could not have been returned by this executable; just return `-1`. Such values usually indicate an out-of-date PCH file (built by some other GCC executable), and such a PCH file won't work.

size_t HOST_HOOKS_GT_PCH_ALLOC_GRANULARITY (void); [Host Hook]
 This host hook returns the alignment required for allocating virtual memory. Usually this is the same as `getpagesize`, but on some hosts the alignment for reserving memory differs from the `pagesize` for committing memory.

18.2 Host Filesystem

GCC needs to know a number of things about the semantics of the host machine's filesystem. Filesystems with Unix and MS-DOS semantics are automatically detected. For other systems, you can define the following macros in `xm-machine.h`.

HAVE_DOS_BASED_FILE_SYSTEM

This macro is automatically defined by `system.h` if the host file system obeys the semantics defined by MS-DOS instead of Unix. DOS file systems are case insensitive, file specifications may begin with a drive letter, and both forward slash and backslash ('/' and '\') are directory separators.

DIR_SEPARATOR

DIR_SEPARATOR_2

If defined, these macros expand to character constants specifying separators for directory names within a file specification. `system.h` will automatically give them appropriate values on Unix and MS-DOS file systems. If your file system is neither of these, define one or both appropriately in `xm-machine.h`.

However, operating systems like VMS, where constructing a pathname is more complicated than just stringing together directory names separated by a special character, should not define either of these macros.

PATH_SEPARATOR

If defined, this macro should expand to a character constant specifying the separator for elements of search paths. The default value is a colon (':'). DOS-based systems usually, but not always, use semicolon (;').

VMS

Define this macro if the host system is VMS.

HOST_OBJECT_SUFFIX

Define this macro to be a C string representing the suffix for object files on your host machine. If you do not define this macro, GCC will use '.o' as the suffix for object files.

HOST_EXECUTABLE_SUFFIX

Define this macro to be a C string representing the suffix for executable files on your host machine. If you do not define this macro, GCC will use the null string as the suffix for executable files.

HOST_BIT_BUCKET

A pathname defined by the host operating system, which can be opened as a file and written to, but all the information written is discarded. This is commonly known as a *bit bucket* or *null device*. If you do not define this macro, GCC will use '/dev/null' as the bit bucket. If the host does not support a bit bucket, define this macro to an invalid filename.

UPDATE_PATH_HOST_CANONICALIZE (*path*)

If defined, a C statement (sans semicolon) that performs host-dependent canonicalization when a path used in a compilation driver or preprocessor is canonicalized. *path* is a malloc-ed path to be canonicalized. If the C statement does canonicalize *path* into a different buffer, the old path should be freed and the new buffer should have been allocated with malloc.

DUMPFIL_ _FORMAT

Define this macro to be a C string representing the format to use for constructing the index part of debugging dump file names. The resultant string must fit in fifteen bytes. The full filename will be the concatenation of: the prefix of the assembler file name, the string resulting from applying this format to an index number, and a string unique to each dump file kind, e.g. `'rtl'`.

If you do not define this macro, GCC will use `'%.02d.'`. You should define this macro if using the default will create an invalid file name.

DELETE_ _IF_ _ORDINARY

Define this macro to be a C statement (sans semicolon) that performs host-dependent removal of ordinary temp files in the compilation driver.

If you do not define this macro, GCC will use the default version. You should define this macro if the default version does not reliably remove the temp file as, for example, on VMS which allows multiple versions of a file.

HOST_ _LACKS_ _INODE_ _NUMBERS

Define this macro if the host filesystem does not report meaningful inode numbers in struct stat.

18.3 Host Misc

FATAL_ _EXIT_ _CODE

A C expression for the status code to be returned when the compiler exits after serious errors. The default is the system-provided macro `'EXIT_FAILURE'`, or `'1'` if the system doesn't define that macro. Define this macro only if these defaults are incorrect.

SUCCESS_ _EXIT_ _CODE

A C expression for the status code to be returned when the compiler exits without serious errors. (Warnings are not serious errors.) The default is the system-provided macro `'EXIT_SUCCESS'`, or `'0'` if the system doesn't define that macro. Define this macro only if these defaults are incorrect.

USE_ _C_ _ALLOCA

Define this macro if GCC should use the C implementation of `alloca` provided by `libiberty.a`. This only affects how some parts of the compiler itself allocate memory. It does not change code generation.

When GCC is built with a compiler other than itself, the C `alloca` is always used. This is because most other implementations have serious bugs. You should define this macro only on a system where no stack-based `alloca` can possibly work. For instance, if a system has a small limit on the size of the stack, GCC's builtin `alloca` will not work reliably.

COLLECT2_ _HOST_ _INITIALIZATION

If defined, a C statement (sans semicolon) that performs host-dependent initialization when `collect2` is being initialized.

GCC_ _DRIVER_ _HOST_ _INITIALIZATION

If defined, a C statement (sans semicolon) that performs host-dependent initialization when a compilation driver is being initialized.

HOST_LONG_LONG_FORMAT

If defined, the string used to indicate an argument of type `long long` to functions like `printf`. The default value is `"ll"`.

HOST_LONG_FORMAT

If defined, the string used to indicate an argument of type `long` to functions like `printf`. The default value is `"l"`.

HOST_PTR_PRINTF

If defined, the string used to indicate an argument of type `void *` to functions like `printf`. The default value is `"%p"`.

In addition, if `configure` generates an incorrect definition of any of the macros in `auto-host.h`, you can override that definition in a host configuration header. If you need to do this, first see if it is possible to fix `configure`.

19 Makefile Fragments

When you configure GCC using the `configure` script, it will construct the file `Makefile` from the template file `Makefile.in`. When it does this, it can incorporate makefile fragments from the `config` directory. These are used to set Makefile parameters that are not amenable to being calculated by `autoconf`. The list of fragments to incorporate is set by `config.gcc` (and occasionally `config.build` and `config.host`); See Section 5.3.2.2 [System Config], page 67.

Fragments are named either `t-target` or `x-host`, depending on whether they are relevant to configuring GCC to produce code for a particular target, or to configuring GCC to run on a particular host. Here *target* and *host* are mnemonics which usually have some relationship to the canonical system name, but no formal connection.

If these files do not exist, it means nothing needs to be added for a given target or host. Most targets need a few `t-target` fragments, but needing `x-host` fragments is rare.

19.1 Target Makefile Fragments

Target makefile fragments can set these Makefile variables.

`LIBGCC2_CFLAGS`

Compiler flags to use when compiling `libgcc2.c`.

`LIB2FUNCS_EXTRA`

A list of source file names to be compiled or assembled and inserted into `libgcc.a`.

`CRTSTUFF_T_CFLAGS`

Special flags used when compiling `crtstuff.c`. See Section 17.22.5 [Initialization], page 668.

`CRTSTUFF_T_CFLAGS_S`

Special flags used when compiling `crtstuff.c` for shared linking. Used if you use `crtbeginS.o` and `crtendS.o` in `EXTRA-PARTS`. See Section 17.22.5 [Initialization], page 668.

`MULTILIB_OPTIONS`

For some targets, invoking GCC in different ways produces objects that cannot be linked together. For example, for some targets GCC produces both big and little endian code. For these targets, you must arrange for multiple versions of `libgcc.a` to be compiled, one for each set of incompatible options. When GCC invokes the linker, it arranges to link in the right version of `libgcc.a`, based on the command line options used.

The `MULTILIB_OPTIONS` macro lists the set of options for which special versions of `libgcc.a` must be built. Write options that are mutually incompatible side by side, separated by a slash. Write options that may be used together separated by a space. The build procedure will build all combinations of compatible options.

For example, if you set `MULTILIB_OPTIONS` to `'m68000/m68020 msoft-float'`, `Makefile` will build special versions of `libgcc.a` using the following sets of

options: `-m68000`, `-m68020`, `-msoft-float`, `'-m68000 -msoft-float'`, and `'-m68020 -msoft-float'`.

MULTILIB_DIRNAMES

If `MULTILIB_OPTIONS` is used, this variable specifies the directory names that should be used to hold the various libraries. Write one element in `MULTILIB_DIRNAMES` for each element in `MULTILIB_OPTIONS`. If `MULTILIB_DIRNAMES` is not used, the default value will be `MULTILIB_OPTIONS`, with all slashes treated as spaces.

`MULTILIB_DIRNAMES` describes the multilib directories using GCC conventions and is applied to directories that are part of the GCC installation. When multilib-enabled, the compiler will add a subdirectory of the form *prefix/multilib* before each directory in the search path for libraries and crt files.

For example, if `MULTILIB_OPTIONS` is set to `'m68000/m68020 msoft-float'`, then the default value of `MULTILIB_DIRNAMES` is `'m68000 m68020 msoft-float'`. You may specify a different value if you desire a different set of directory names.

MULTILIB_MATCHES

Sometimes the same option may be written in two different ways. If an option is listed in `MULTILIB_OPTIONS`, GCC needs to know about any synonyms. In that case, set `MULTILIB_MATCHES` to a list of items of the form `'option=option'` to describe all relevant synonyms. For example, `'m68000=mc68000 m68020=mc68020'`.

MULTILIB_EXCEPTIONS

Sometimes when there are multiple sets of `MULTILIB_OPTIONS` being specified, there are combinations that should not be built. In that case, set `MULTILIB_EXCEPTIONS` to be all of the switch exceptions in shell case syntax that should not be built.

For example the ARM processor cannot execute both hardware floating point instructions and the reduced size THUMB instructions at the same time, so there is no need to build libraries with both of these options enabled. Therefore `MULTILIB_EXCEPTIONS` is set to:

```
*mthumb/*mhard-float*
```

MULTILIB_REQUIRED

Sometimes when there are only a few combinations are required, it would be a big effort to come up with a `MULTILIB_EXCEPTIONS` list to cover all undesired ones. In such a case, just listing all the required combinations in `MULTILIB_REQUIRED` would be more straightforward.

The way to specify the entries in `MULTILIB_REQUIRED` is same with the way used for `MULTILIB_EXCEPTIONS`, only this time what are required will be specified. Suppose there are multiple sets of `MULTILIB_OPTIONS` and only two combinations are required, one for ARMv7-M and one for ARMv7-R with hard floating-point ABI and FPU, the `MULTILIB_REQUIRED` can be set to:

```
MULTILIB_REQUIRED = mthumb/march=armv7-m
MULTILIB_REQUIRED += march=armv7-r/mfloat-abi=hard/mfpu=vfpv3-d16
```

The `MULTILIB_REQUIRED` can be used together with `MULTILIB_EXCEPTIONS`. The option combinations generated from `MULTILIB_OPTIONS` will be filtered by `MULTILIB_EXCEPTIONS` and then by `MULTILIB_REQUIRED`.

`MULTILIB_REUSE`

Sometimes it is desirable to reuse one existing multilib for different sets of options. Such kind of reuse can minimize the number of multilib variants. And for some targets it is better to reuse an existing multilib than to fall back to default multilib when there is no corresponding multilib. This can be done by adding reuse rules to `MULTILIB_REUSE`.

A reuse rule is comprised of two parts connected by equality sign. The left part is the option set used to build multilib and the right part is the option set that will reuse this multilib. Both parts should only use options specified in `MULTILIB_OPTIONS` and the equality signs found in options name should be replaced with periods. An explicit period in the rule can be escaped by preceding it with a backslash. The order of options in the left part matters and should be same with those specified in `MULTILIB_REQUIRED` or aligned with the order in `MULTILIB_OPTIONS`. There is no such limitation for options in the right part as we don't build multilib from them.

`MULTILIB_REUSE` is different from `MULTILIB_MATCHES` in that it sets up relations between two option sets rather than two options. Here is an example to demo how we reuse libraries built in Thumb mode for applications built in ARM mode:

```
MULTILIB_REUSE = mthumb/march.armv7-r=marm/march.armv7-r
```

Before the advent of `MULTILIB_REUSE`, GCC select multilib by comparing command line options with options used to build multilib. The `MULTILIB_REUSE` is complementary to that way. Only when the original comparison matches nothing it will work to see if it is OK to reuse some existing multilib.

`MULTILIB_EXTRA_OPTS`

Sometimes it is desirable that when building multiple versions of `libgcc.a` certain options should always be passed on to the compiler. In that case, set `MULTILIB_EXTRA_OPTS` to be the list of options to be used for all builds. If you set this, you should probably set `CRTSTUFF_T_CFLAGS` to a dash followed by it.

`MULTILIB_OSDIRNAMES`

If `MULTILIB_OPTIONS` is used, this variable specifies a list of subdirectory names, that are used to modify the search path depending on the chosen multilib. Unlike `MULTILIB_DIRNAMES`, `MULTILIB_OSDIRNAMES` describes the multilib directories using operating systems conventions, and is applied to the directories such as `lib` or those in the `LIBRARY_PATH` environment variable. The format is either the same as of `MULTILIB_DIRNAMES`, or a set of mappings. When it is the same as `MULTILIB_DIRNAMES`, it describes the multilib directories using operating system conventions, rather than GCC conventions. When it is a set of mappings of the form `gccdir=osdir`, the left side gives the GCC convention and the right gives the equivalent OS defined location. If the `osdir` part begins with a `!`, GCC will not search in the non-multilib directory and use exclusively the multilib directory. Otherwise, the compiler will examine the search path for

libraries and crt files twice; the first time it will add *multilib* to each directory in the search path, the second it will not.

For configurations that support both multilib and multiarch, `MULTILIB_OSDIRNAMES` also encodes the multiarch name, thus subsuming `MULTIARCH_DIRNAME`. The multiarch name is appended to each directory name, separated by a colon (e.g. `../lib32:i386-linux-gnu`).

Each multiarch subdirectory will be searched before the corresponding OS multilib directory, for example `/lib/i386-linux-gnu` before `/lib/./lib32`. The multiarch name will also be used to modify the system header search path, as explained for `MULTIARCH_DIRNAME`.

`MULTIARCH_DIRNAME`

This variable specifies the multiarch name for configurations that are multiarch-enabled but not multilibbed configurations.

The multiarch name is used to augment the search path for libraries, crt files and system header files with additional locations. The compiler will add a multiarch subdirectory of the form *prefix/multiarch* before each directory in the library and crt search path. It will also add two directories `LOCAL_INCLUDE_DIR/multiarch` and `NATIVE_SYSTEM_HEADER_DIR/multiarch` to the system header search path, respectively before `LOCAL_INCLUDE_DIR` and `NATIVE_SYSTEM_HEADER_DIR`.

`MULTIARCH_DIRNAME` is not used for configurations that support both multilib and multiarch. In that case, multiarch names are encoded in `MULTILIB_OSDIRNAMES` instead.

More documentation about multiarch can be found at <https://wiki.debian.org/Multiarch>.

`SPECS`

Unfortunately, setting `MULTILIB_EXTRA_OPTS` is not enough, since it does not affect the build of target libraries, at least not the build of the default multilib. One possible work-around is to use `DRIVER_SELF_SPECS` to bring options from the `specs` file as if they had been passed in the compiler driver command line. However, you don't want to be adding these options after the toolchain is installed, so you can instead tweak the `specs` file that will be used during the toolchain build, while you still install the original, built-in `specs`. The trick is to set `SPECS` to some other filename (say `specs.install`), that will then be created out of the built-in `specs`, and introduce a `Makefile` rule to generate the `specs` file that's going to be used at build time out of your `specs.install`.

`T_CFLAGS`

These are extra flags to pass to the C compiler. They are used both when building GCC, and when compiling things with the just-built GCC. This variable is deprecated and should not be used.

19.2 Host Makefile Fragments

The use of *x-host* fragments is discouraged. You should only use it for makefile dependencies.

20 collect2

GCC uses a utility called `collect2` on nearly all systems to arrange to call various initialization functions at start time.

The program `collect2` works by linking the program once and looking through the linker output file for symbols with particular names indicating they are constructor functions. If it finds any, it creates a new temporary ‘.c’ file containing a table of them, compiles it, and links the program a second time including that file.

The actual calls to the constructors are carried out by a subroutine called `__main`, which is called (automatically) at the beginning of the body of `main` (provided `main` was compiled with GNU CC). Calling `__main` is necessary, even when compiling C code, to allow linking C and C++ object code together. (If you use `-nostdlib`, you get an unresolved reference to `__main`, since it’s defined in the standard GCC library. Include `-lgcc` at the end of your compiler command line to resolve this reference.)

The program `collect2` is installed as `ld` in the directory where the passes of the compiler are installed. When `collect2` needs to find the *real* `ld`, it tries the following file names:

- a hard coded linker file name, if GCC was configured with the `--with-ld` option.
- `real-ld` in the directories listed in the compiler’s search directories.
- `real-ld` in the directories listed in the environment variable `PATH`.
- The file specified in the `REAL_LD_FILE_NAME` configuration macro, if specified.
- `ld` in the compiler’s search directories, except that `collect2` will not execute itself recursively.
- `ld` in `PATH`.

“The compiler’s search directories” means all the directories where `gcc` searches for passes of the compiler. This includes directories that you specify with `-B`.

Cross-compilers search a little differently:

- `real-ld` in the compiler’s search directories.
- `target-real-ld` in `PATH`.
- The file specified in the `REAL_LD_FILE_NAME` configuration macro, if specified.
- `ld` in the compiler’s search directories.
- `target-ld` in `PATH`.

`collect2` explicitly avoids running `ld` using the file name under which `collect2` itself was invoked. In fact, it remembers up a list of such names—in case one copy of `collect2` finds another copy (or version) of `collect2` installed as `ld` in a second place in the search path.

`collect2` searches for the utilities `nm` and `strip` using the same algorithm as above for `ld`.

21 Standard Header File Directories

`GCC_INCLUDE_DIR` means the same thing for native and cross. It is where GCC stores its private include files, and also where GCC stores the fixed include files. A cross compiled GCC runs `fixincludes` on the header files in `$(tooldir)/include`. (If the cross compilation header files need to be fixed, they must be installed before GCC is built. If the cross compilation header files are already suitable for GCC, nothing special need be done).

`GPLUSPLUS_INCLUDE_DIR` means the same thing for native and cross. It is where `g++` looks first for header files. The C++ library installs only target independent header files in that directory.

`LOCAL_INCLUDE_DIR` is used only by native compilers. GCC doesn't install anything there. It is normally `/usr/local/include`. This is where local additions to a packaged system should place header files.

`CROSS_INCLUDE_DIR` is used only by cross compilers. GCC doesn't install anything there.

`TOOL_INCLUDE_DIR` is used for both native and cross compilers. It is the place for other packages to install header files that GCC will use. For a cross-compiler, this is the equivalent of `/usr/include`. When you build a cross-compiler, `fixincludes` processes any header files in this directory.

22 Memory Management and Type Information

GCC uses some fairly sophisticated memory management techniques, which involve determining information about GCC's data structures from GCC's source code and using this information to perform garbage collection and implement precompiled headers.

A full C++ parser would be too complicated for this task, so a limited subset of C++ is interpreted and special markers are used to determine what parts of the source to look at. All `struct`, `union` and `template` structure declarations that define data structures that are allocated under control of the garbage collector must be marked. All global variables that hold pointers to garbage-collected memory must also be marked. Finally, all global variables that need to be saved and restored by a precompiled header must be marked. (The precompiled header mechanism can only save static variables if they're scalar. Complex data structures must be allocated in garbage-collected memory to be saved in a precompiled header.)

The full format of a marker is

```
GTY ([option] [(param)], [option] [(param)] ...)
```

but in most cases no options are needed. The outer double parentheses are still necessary, though: `GTY(())`. Markers can appear:

- In a structure definition, before the open brace;
- In a global variable declaration, after the keyword `static` or `extern`; and
- In a structure field definition, before the name of the field.

Here are some examples of marking simple data structures and globals.

```
struct GTY(()) tag
{
    fields...
};

typedef struct GTY(()) tag
{
    fields...
} *typename;

static GTY(()) struct tag *list; /* points to GC memory */
static GTY(()) int counter;      /* save counter in a PCH */
```

The parser understands simple typedefs such as `typedef struct tag *name`; and `typedef int name`;. These don't need to be marked.

However, in combination with `GTY`, avoid using typedefs such as `typedef int_hash<...> name`; for these generate infinite-recursion code. See PR103157 (<https://gcc.gnu.org/PR103157>). Instead, you may use `struct name : int_hash<...> {};`, for example.

Since `gengtype`'s understanding of C++ is limited, there are several constructs and declarations that are not supported inside classes/structures marked for automatic GC code generation. The following C++ constructs produce a `gengtype` error on structures/classes marked for automatic GC code generation:

- Type definitions inside classes/structures are not supported.
- Enumerations inside classes/structures are not supported.

If you have a class or structure using any of the above constructs, you need to mark that class as `GTY ((user))` and provide your own marking routines (see section Section 22.3 [User GC], page 743, for details).

It is always valid to include function definitions inside classes. Those are always ignored by `gengtype`, as it only cares about data members.

22.1 The Inside of a `GTY(())`

Sometimes the C code is not enough to fully describe the type structure. Extra information can be provided with `GTY` options and additional markers. Some options take a parameter, which may be either a string or a type name, depending on the parameter. If an option takes no parameter, it is acceptable either to omit the parameter entirely, or to provide an empty string as a parameter. For example, `GTY ((skip))` and `GTY ((skip ("")))` are equivalent.

When the parameter is a string, often it is a fragment of C code. Four special escapes may be used in these strings, to refer to pieces of the data structure being marked:

<code>%h</code>	The current structure.
<code>%1</code>	The structure that immediately contains the current structure.
<code>%0</code>	The outermost structure that contains the current structure.
<code>%a</code>	A partial expression of the form <code>[i1][i2]...</code> that indexes the array item currently being marked.

For instance, suppose that you have a structure of the form

```
struct A {
    ...
};
struct B {
    struct A foo[12];
};
```

and `b` is a variable of type `struct B`. When marking `'b.foo[11]'`, `%h` would expand to `'b.foo[11]'`, `%0` and `%1` would both expand to `'b'`, and `%a` would expand to `'[11]'`.

As in ordinary C, adjacent strings will be concatenated; this is helpful when you have a complicated expression.

```
GTY ((chain_next ("TREE_CODE (&%h.generic) == INTEGER_TYPE"
                  " ? TYPE_NEXT_VARIANT (&%h.generic)"
                  " : TREE_CHAIN (&%h.generic)")))
```

The available options are:

`length ("expression")`

There are two places the type machinery will need to be explicitly told the length of an array of non-atomic objects. The first case is when a structure ends in a variable-length array, like this:

```
struct GTY(()) rtvec_def {
    int num_elem;          /* number of elements */
    rtx GTY ((length ("%h.num_elem"))) elem[1];
};
```

In this case, the `length` option is used to override the specified array length (which should usually be 1). The parameter of the option is a fragment of C code that calculates the length.

The second case is when a structure or a global variable contains a pointer to an array, like this:

```
struct gimple_omp_for_iter * GTY((length ("%h.collapse"))) iter;
```

In this case, `iter` has been allocated by writing something like

```
x->iter = ggc_alloc_cleared_vec_gimple_omp_for_iter (collapse);
```

and the `collapse` provides the length of the field.

This second use of `length` also works on global variables, like:

```
static GTY((length("reg_known_value_size"))) rtx *reg_known_value;
```

Note that the `length` option is only meant for use with arrays of non-atomic objects, that is, objects that contain pointers pointing to other GTY-managed objects. For other GC-allocated arrays and strings you should use `atomic` or `string_length`.

`string_length ("expression")`

In order to simplify production of PCH, a structure member that is a plain array of bytes (an optionally `const` and/or `unsigned char *`) is treated specially by the infrastructure. Even if such an array has not been allocated in GC-controlled memory, it will still be written properly into a PCH. The machinery responsible for this needs to know the length of the data; by default, the length is determined by calling `strlen` on the pointer. The `string_length` option specifies an alternate way to determine the length, such as by inspecting another struct member:

```
struct GTY(()) non_terminated_string {
    size_t sz;
    const char * GTY((string_length ("%h.sz"))) data;
};
```

Similarly, this is useful for (regular NUL-terminated) strings with NUL characters embedded (that the default `strlen` use would run afoul of):

```
struct GTY(()) multi_string {
    const char * GTY((string_length ("%h.len + 1"))) str;
    size_t len;
};
```

The `string_length` option currently is not supported for (fields in) global variables.

`skip`

If `skip` is applied to a field, the type machinery will ignore it. This is somewhat dangerous; the only safe use is in a union when one field really isn't ever used.

`callback`

`callback` should be applied to fields with pointer to function type and causes the field to be ignored similarly to `skip`, except when writing PCH and the field is non-NULL it will remember the field's address for relocation purposes if the process writing PCH has different load base from a process reading PCH.

for_user

Use this to mark types that need to be marked by user gc routines, but are not referred to in a template argument. So if you have some user gc type T1 and a non user gc type T2 you can give T2 the `for_user` option so that the marking functions for T1 can call non mangled functions to mark T2.

```
desc ("expression")
tag ("constant")
default
```

The type machinery needs to be told which field of a **union** is currently active. This is done by giving each field a constant **tag** value, and then specifying a discriminator using **desc**. The value of the expression given by **desc** is compared against each **tag** value, each of which should be different. If no **tag** is matched, the field marked with **default** is used if there is one, otherwise no field in the union will be marked.

In the **desc** option, the “current structure” is the union that it discriminates. Use `%1` to mean the structure containing it. There are no escapes available to the **tag** option, since it is a constant.

For example,

```
struct GTY(()) tree_binding
{
    struct tree_common common;
    union tree_binding_u {
        tree GTY ((tag ("0"))) scope;
        struct cp_binding_level * GTY ((tag ("1"))) level;
    } GTY ((desc ("BINDING_HAS_LEVEL_P ((tree)&%0)"))) xscope;
    tree value;
};
```

In this example, the value of `BINDING_HAS_LEVEL_P` when applied to a `struct tree_binding *` is presumed to be 0 or 1. If 1, the type mechanism will treat the field `level` as being present and if 0, will treat the field `scope` as being present.

The **desc** and **tag** options can also be used for inheritance to denote which subclass an instance is. See Section 22.2 [Inheritance and GTY], page 743, for more information.

cache

When the **cache** option is applied to a global variable `gt_cleare_cache` is called on that variable between the mark and sweep phases of garbage collection. The `gt_clear_cache` function is free to mark blocks as used, or to clear pointers in the variable.

In a hash table, the ‘`gt_cleare_cache`’ function discards entries if the key is not marked, or marks the value if the key is marked.

Note that caches should generally use **deletable** instead; **cache** is only preferable if the value is impractical to recompute from the key when needed.

The **cache** option can have an optional argument, name of the function which should be called before ‘`gt_cleare_cache`’. This can be useful if the hash table

needs to be traversed and mark some pointers before `gt_cleare_cache` could clear slots in it.

`deletable`

`deletable`, when applied to a global variable, indicates that when garbage collection runs, there's no need to mark anything pointed to by this variable, it can just be set to `NULL` instead. This is used to keep a list of free structures around for re-use.

`maybe_undef`

When applied to a field, `maybe_undef` indicates that it's OK if the structure that this field points to is never defined, so long as this field is always `NULL`. This is used to avoid requiring backends to define certain optional structures. It doesn't work with language frontends.

`nested_ptr (type, "to expression", "from expression")`

The type machinery expects all pointers to point to the start of an object. Sometimes for abstraction purposes it's convenient to have a pointer which points inside an object. So long as it's possible to convert the original object to and from the pointer, such pointers can still be used. *type* is the type of the original object, the *to expression* returns the pointer given the original object, and the *from expression* returns the original object given the pointer. The pointer will be available using the `%h` escape.

`chain_next ("expression")`

`chain_prev ("expression")`

`chain_circular ("expression")`

It's helpful for the type machinery to know if objects are often chained together in long lists; this lets it generate code that uses less stack space by iterating along the list instead of recursing down it. `chain_next` is an expression for the next item in the list, `chain_prev` is an expression for the previous item. For singly linked lists, use only `chain_next`; for doubly linked lists, use both. The machinery requires that taking the next item of the previous item gives the original item. `chain_circular` is similar to `chain_next`, but can be used for circular single linked lists.

`reorder ("function name")`

Some data structures depend on the relative ordering of pointers. If the precompiled header machinery needs to change that ordering, it will call the function referenced by the `reorder` option, before changing the pointers in the object that's pointed to by the field the option applies to. The function must take four arguments, with the signature `'void *, void *, gt_pointer_operator, void *'`. The first parameter is a pointer to the structure that contains the object being updated, or the object itself if there is no containing structure. The second parameter is a cookie that should be ignored. The third parameter is a routine that, given a pointer, will update it to its correct new value. The fourth parameter is a cookie that must be passed to the second parameter.

PCH cannot handle data structures that depend on the absolute values of pointers. **reorder** functions can be expensive. When possible, it is better to depend on properties of the data, like an ID number or the hash of a string instead.

atomic

The **atomic** option can only be used with pointers. It informs the GC machinery that the memory that the pointer points to does not contain any pointers, and hence it should be treated by the GC and PCH machinery as an “atomic” block of memory that does not need to be examined when scanning memory for pointers. In particular, the machinery will not scan that memory for pointers to mark them as reachable (when marking pointers for GC) or to relocate them (when writing a PCH file).

The **atomic** option differs from the **skip** option. **atomic** keeps the memory under Garbage Collection, but makes the GC ignore the contents of the memory. **skip** is more drastic in that it causes the pointer and the memory to be completely ignored by the Garbage Collector. So, memory marked as **atomic** is automatically freed when no longer reachable, while memory marked as **skip** is not.

The **atomic** option must be used with great care, because all sorts of problem can occur if used incorrectly, that is, if the memory the pointer points to does actually contain a pointer.

Here is an example of how to use it:

```
struct GTY(()) my_struct {
    int number_of_elements;
    unsigned int * GTY ((atomic)) elements;
};
```

In this case, **elements** is a pointer under GC, and the memory it points to needs to be allocated using the Garbage Collector, and will be freed automatically by the Garbage Collector when it is no longer referenced. But the memory that the pointer points to is an array of **unsigned int** elements, and the GC must not try to scan it to find pointers to mark or relocate, which is why it is marked with the **atomic** option.

Note that, currently, global variables cannot be marked with **atomic**; only fields of a struct can. This is a known limitation. It would be useful to be able to mark global pointers with **atomic** to make the PCH machinery aware of them so that they are saved and restored correctly to PCH files.

special ("name")

The **special** option is used to mark types that have to be dealt with by special case machinery. The parameter is the name of the special case. See `gengtype.cc` for further details. Avoid adding new special cases unless there is no other alternative.

user

The **user** option indicates that the code to mark structure fields is completely handled by user-provided routines. See section Section 22.3 [User GC], page 743, for details on what functions need to be provided.

22.2 Support for inheritance

gengtype has some support for simple class hierarchies. You can use this to have gengtype autogenerate marking routines, provided:

- There must be a concrete base class, with a discriminator expression that can be used to identify which subclass an instance is.
- Only single inheritance is used.
- None of the classes within the hierarchy are templates.

If your class hierarchy does not fit in this pattern, you must use Section 22.3 [User GC], page 743, instead.

The base class and its discriminator must be identified using the “desc” option. Each concrete subclass must use the “tag” option to identify which value of the discriminator it corresponds to.

Every class in the hierarchy must have a `GTY(())` marker, as gengtype will only attempt to parse classes that have such a marker¹.

```
class GTY((desc("%h.kind"), tag("0"))) example_base
{
public:
    int kind;
    tree a;
};

class GTY((tag("1"))) some_subclass : public example_base
{
public:
    tree b;
};

class GTY((tag("2"))) some_other_subclass : public example_base
{
public:
    tree c;
};
```

The generated marking routines for the above will contain a “switch” on “kind”, visiting all appropriate fields. For example, if kind is 2, it will cast to “some_other_subclass” and visit fields a, b, and c.

22.3 Support for user-provided GC marking routines

The garbage collector supports types for which no automatic marking code is generated. For these types, the user is required to provide three functions: one to act as a marker for garbage collection, and two functions to act as marker and pointer walker for pre-compiled headers.

Given a structure `struct GTY((user)) my_struct`, the following functions should be defined to mark `my_struct`:

```
void gt_ggc_mx (my_struct *p)
```

¹ Classes lacking such a marker will not be identified as being part of the hierarchy, and so the marking routines will not handle them, leading to a assertion failure within the marking routines due to an unknown tag value (assuming that assertions are enabled).

```

{
    /* This marks field 'fld'. */
    gt_ggc_mx (p->fld);
}

void gt_pch_nx (my_struct *p)
{
    /* This marks field 'fld'. */
    gt_pch_nx (tp->fld);
}

void gt_pch_nx (my_struct *p, gt_pointer_operator op, void *cookie)
{
    /* For every field 'fld', call the given pointer operator. */
    op (&(tp->fld), NULL, cookie);
}

```

In general, each marker *M* should call *M* for every pointer field in the structure. Fields that are not allocated in GC or are not pointers must be ignored.

For embedded lists (e.g., structures with a `next` or `prev` pointer), the marker must follow the chain and mark every element in it.

Note that the rules for the pointer walker `gt_pch_nx (my_struct *, gt_pointer_operator, void *)` are slightly different. In this case, the operation `op` must be applied to the *address* of every pointer field.

22.3.1 User-provided marking routines for template types

When a template type *TP* is marked with *GTY*, all instances of that type are considered user-provided types. This means that the individual instances of *TP* do not need to be marked with *GTY*. The user needs to provide template functions to mark all the fields of the type.

The following code snippets represent all the functions that need to be provided. Note that type *TP* may reference to more than one type. In these snippets, there is only one type *T*, but there could be more.

```

template<typename T>
void gt_ggc_mx (TP<T> *tp)
{
    extern void gt_ggc_mx (T&);

    /* This marks field 'fld' of type 'T'. */
    gt_ggc_mx (tp->fld);
}

template<typename T>
void gt_pch_nx (TP<T> *tp)
{
    extern void gt_pch_nx (T&);

    /* This marks field 'fld' of type 'T'. */
    gt_pch_nx (tp->fld);
}

template<typename T>
void gt_pch_nx (TP<T *> *tp, gt_pointer_operator op, void *cookie)
{
    /* For every field 'fld' of 'tp' with type 'T *', call the given

```

```

        pointer operator. */
    op (&(tp->fld), NULL, cookie);
}

template<typename T>
void gt_pch_nx (TP<T> *tp, gt_pointer_operator, void *cookie)
{
    extern void gt_pch_nx (T *, gt_pointer_operator, void *);

    /* For every field 'fld' of 'tp' with type 'T', call the pointer
       walker for all the fields of T. */
    gt_pch_nx (&(tp->fld), op, cookie);
}

```

Support for user-defined types is currently limited. The following restrictions apply:

1. Type TP and all the argument types T must be marked with **GTy**.
2. Type TP can only have type names in its argument list.
3. The pointer walker functions are different for TP<T> and TP<T *>. In the case of TP<T>, references to T must be handled by calling `gt_pch_nx` (which will, in turn, walk all the pointers inside fields of T). In the case of TP<T *>, references to T * must be handled by calling the `op` function on the address of the pointer (see the code snippets above).

22.4 Marking Roots for the Garbage Collector

In addition to keeping track of types, the type machinery also locates the global variables (*roots*) that the garbage collector starts at. Roots must be declared using one of the following syntaxes:

- `extern GTy([options]) type name;`
- `static GTy([options]) type name;`

The syntax

- `GTy([options]) type name;`

is *not* accepted. There should be an `extern` declaration of such a variable in a header somewhere—mark that, not the definition. Or, if the variable is only used in one file, make it `static`.

22.5 Source Files Containing Type Information

Whenever you add **GTy** markers to a source file that previously had none, or create a new source file containing **GTy** markers, there are three things you need to do:

1. You need to add the file to the list of source files the type machinery scans. There are four cases:
 - a. For a back-end file, this is usually done automatically; if not, you should add it to `target_gtfiles` in the appropriate port's entries in `config.gcc`.
 - b. For files shared by all front ends, add the filename to the `GTFILES` variable in `Makefile.in`.
 - c. For files that are part of one front end, add the filename to the `gtfiles` variable defined in the appropriate `config-lang.in`. Headers should appear before non-headers in this list.

- d. For files that are part of some but not all front ends, add the filename to the `gtfiles` variable of *all* the front ends that use it.
2. If the file was a header file, you'll need to check that it's included in the right place to be visible to the generated files. For a back-end header file, this should be done automatically. For a front-end header file, it needs to be included by the same file that includes `gtype-lang.h`. For other header files, it needs to be included in `gtype-desc.cc`, which is a generated file, so add it to `ifiles` in `open_base_file` in `gengtype.cc`.

For source files that aren't header files, the machinery will generate a header file that should be included in the source file you just changed. The file will be called `gt-path.h` where *path* is the pathname relative to the `gcc` directory with slashes replaced by `-`, so for example the header file to be included in `cp/parser.cc` is called `gt-cp-parser.h`. The generated header file should be included after everything else in the source file.

For language frontends, there is another file that needs to be included somewhere. It will be called `gtype-lang.h`, where *lang* is the name of the subdirectory the language is contained in.

Plugins can add additional root tables. Run the `gengtype` utility in plugin mode as `gengtype -P pluginout.h source-dir file-list plugin*.c` with your plugin files *plugin*.c* using `GTy` to generate the *pluginout.h* file. The GCC build tree is needed to be present in that mode.

22.6 How to invoke the garbage collector

The GCC garbage collector GGC is only invoked explicitly. In contrast with many other garbage collectors, it is not implicitly invoked by allocation routines when a lot of memory has been consumed. So the only way to have GGC reclaim storage is to call the `ggc_collect` function explicitly. With *mode* `GGC_COLLECT_FORCE` or otherwise (default `GGC_COLLECT_HEURISTIC`) when the internal heuristic decides to collect, this call is potentially an expensive operation, as it may have to scan the entire heap. Beware that local variables (on the GCC call stack) are not followed by such an invocation (as many other garbage collectors do): you should reference all your data from static or external `GTy`-ed variables, and it is advised to call `ggc_collect` with a shallow call stack. The GGC is an exact mark and sweep garbage collector (so it does not scan the call stack for pointers). In practice GCC passes don't often call `ggc_collect` themselves, because it is called by the pass manager between passes.

At the time of the `ggc_collect` call all pointers in the GC-marked structures must be valid or `NULL`. In practice this means that there should not be uninitialized pointer fields in the structures even if your code never reads or writes those fields at a particular instance. One way to ensure this is to use cleared versions of allocators unless all the fields are initialized manually immediately after allocation.

22.7 Troubleshooting the garbage collector

With the current garbage collector implementation, most issues should show up as GCC compilation errors. Some of the most commonly encountered issues are described below.

- Gengtype does not produce allocators for a `GTy`-marked type. Gengtype checks if there is at least one possible path from GC roots to at least one instance of each type before

outputting allocators. If there is no such path, the **GTY** markers will be ignored and no allocators will be output. Solve this by making sure that there exists at least one such path. If creating it is unfeasible or raises a “code smell”, consider if you really must use GC for allocating such type.

- Link-time errors about undefined `gt_ggc_r_foo_bar` and similarly-named symbols. Check if your `foo_bar` source file has `#include "gt-foo_bar.h"` as its very last line.

23 Plugins

GCC plugins are loadable modules that provide extra features to the compiler. Like GCC itself they can be distributed in source and binary forms.

GCC plugins provide developers with a rich subset of the GCC API to allow them to extend GCC as they see fit. Whether it is writing an additional optimization pass, transforming code, or analyzing information, plugins can be quite useful.

23.1 Loading Plugins

Plugins are supported on platforms that support `-ldl -rdynamic` as well as Windows/MinGW. They are loaded by the compiler using `dlopen` or equivalent and invoked at pre-determined locations in the compilation process.

Plugins are loaded with

```
-fplugin=/path/to/name.ext -fplugin-arg-name-key1[=value1]
```

Where *name* is the plugin name and *ext* is the platform-specific dynamic library extension. It should be `dll` on Windows/MinGW, `dylib` on Darwin/macOS, and `so` on all other platforms. The plugin arguments are parsed by GCC and passed to respective plugins as key-value pairs. Multiple plugins can be invoked by specifying multiple `-fplugin` arguments.

A plugin can be simply given by its short name (no dots or slashes). When simply passing `-fplugin=name`, the plugin is loaded from the `plugin` directory, so `-fplugin=name` is the same as `-fplugin=`gcc -print-file-name=plugin`/name.ext`, using backquote shell syntax to query the `plugin` directory.

23.2 Plugin API

Plugins are activated by the compiler at specific events as defined in `gcc-plugin.h`. For each event of interest, the plugin should call `register_callback` specifying the name of the event and address of the callback function that will handle that event.

The header `gcc-plugin.h` must be the first gcc header to be included.

23.2.1 Plugin license check

Every plugin should define the global symbol `plugin_is_GPL_compatible` to assert that it has been licensed under a GPL-compatible license. If this symbol does not exist, the compiler will emit a fatal error and exit with the error message:

```
fatal error: plugin name is not licensed under a GPL-compatible license
name: undefined symbol: plugin_is_GPL_compatible
compilation terminated
```

The declared type of the symbol should be `int`, to match a forward declaration in `gcc-plugin.h` that suppresses C++ mangling. It does not need to be in any allocated section, though. The compiler merely asserts that the symbol exists in the global scope. Something like this is enough:

```
int plugin_is_GPL_compatible;
```

23.2.2 Plugin initialization

Every plugin should export a function called `plugin_init` that is called right after the plugin is loaded. This function is responsible for registering all the callbacks required by the plugin and do any other required initialization.

This function is called from `compile_file` right before invoking the parser. The arguments to `plugin_init` are:

- `plugin_info`: Plugin invocation information.
- `version`: GCC version.

The `plugin_info` struct is defined as follows:

```
struct plugin_name_args
{
    char *base_name;           /* Short name of the plugin
                               (filename without .so suffix). */
    const char *full_name;     /* Path to the plugin as specified with
                               -fplugin=. */
    int argc;                  /* Number of arguments specified with
                               -fplugin-arg-.... */
    struct plugin_argument *argv; /* Array of ARGV key-value pairs. */
    const char *version;       /* Version string provided by plugin. */
    const char *help;          /* Help string provided by plugin. */
}
```

If initialization fails, `plugin_init` must return a non-zero value. Otherwise, it should return 0.

The version of the GCC compiler loading the plugin is described by the following structure:

```
struct plugin_gcc_version
{
    const char *basever;
    const char *datestamp;
    const char *devphase;
    const char *revision;
    const char *configuration_arguments;
};
```

The function `plugin_default_version_check` takes two pointers to such structure and compare them field by field. It can be used by the plugin's `plugin_init` function.

The version of GCC used to compile the plugin can be found in the symbol `gcc_version` defined in the header `plugin-version.h`. The recommended version check to perform looks like

```
#include "plugin-version.h"
...

int
plugin_init (struct plugin_name_args *plugin_info,
             struct plugin_gcc_version *version)
{
    if (!plugin_default_version_check (version, &gcc_version))
        return 1;
}
```

but you can also check the individual fields if you want a less strict check.

23.2.3 Plugin callbacks

Callback functions have the following prototype:

```
/* The prototype for a plugin callback function.
   gcc_data - event-specific data provided by GCC
   user_data - plugin-specific data provided by the plug-in. */
typedef void (*plugin_callback_func)(void *gcc_data, void *user_data);
```

Callbacks can be invoked at the following pre-determined events:

```
enum plugin_event
{
    PLUGIN_START_PARSE_FUNCTION, /* Called before parsing the body of a function. */
    PLUGIN_FINISH_PARSE_FUNCTION, /* After finishing parsing a function. */
    PLUGIN_PASS_MANAGER_SETUP, /* To hook into pass manager. */
    PLUGIN_FINISH_TYPE, /* After finishing parsing a type. */
    PLUGIN_FINISH_DECL, /* After finishing parsing a declaration. */
    PLUGIN_FINISH_UNIT, /* Useful for summary processing. */
    PLUGIN_PRE_GENERICIZE, /* Allows to see low level AST in C and C++ frontends. */
    PLUGIN_FINISH, /* Called before GCC exits. */
    PLUGIN_INFO, /* Information about the plugin. */
    PLUGIN_GGC_START, /* Called at start of GCC Garbage Collection. */
    PLUGIN_GGC_MARKING, /* Extend the GGC marking. */
    PLUGIN_GGC_END, /* Called at end of GGC. */
    PLUGIN_REGISTER_GGC_ROOTS, /* Register an extra GGC root table. */
    PLUGIN_ATTRIBUTES, /* Called during attribute registration */
    PLUGIN_START_UNIT, /* Called before processing a translation unit. */
    PLUGIN_PRAGMAS, /* Called during pragma registration. */
    /* Called before first pass from all_passes. */
    PLUGIN_ALL_PASSES_START,
    /* Called after last pass from all_passes. */
    PLUGIN_ALL_PASSES_END,
    /* Called before first ipa pass. */
    PLUGIN_ALL_IPA_PASSES_START,
    /* Called after last ipa pass. */
    PLUGIN_ALL_IPA_PASSES_END,
    /* Allows to override pass gate decision for current_pass. */
    PLUGIN_OVERRIDE_GATE,
    /* Called before executing a pass. */
    PLUGIN_PASS_EXECUTION,
    /* Called before executing subpasses of a GIMPLE_PASS in
       execute_ipa_pass_list. */
    PLUGIN_EARLY_GIMPLE_PASSES_START,
    /* Called after executing subpasses of a GIMPLE_PASS in
       execute_ipa_pass_list. */
    PLUGIN_EARLY_GIMPLE_PASSES_END,
    /* Called when a pass is first instantiated. */
    PLUGIN_NEW_PASS,
    /* Called when a file is #include-d or given via the #line directive.
       This could happen many times. The event data is the included file path,
       as a const char* pointer. */
    PLUGIN_INCLUDE_FILE,

    PLUGIN_EVENT_FIRST_DYNAMIC /* Dummy event used for indexing callback
                               array. */
};
```

In addition, plugins can also look up the enumerator of a named event, and / or generate new events dynamically, by calling the function `get_named_event_id`.

To register a callback, the plugin calls `register_callback` with the arguments:

- `char *name`: Plugin name.
- `int event`: The event code.
- `plugin_callback_func callback`: The function that handles `event`.
- `void *user_data`: Pointer to plugin-specific data.

For the `PLUGIN_PASS_MANAGER_SETUP`, `PLUGIN_INFO`, and `PLUGIN_REGISTER_GGC_ROOTS` pseudo-events the `callback` should be null, and the `user_data` is specific.

When the `PLUGIN_PRAGMAS` event is triggered (with a null pointer as data from GCC), plugins may register their own pragmas. Notice that pragmas are not available from `lto1`, so plugins used with `-flto` option to GCC during link-time optimization cannot use pragmas and do not even see functions like `c_register_pragma` or `pragma_lex`.

The `PLUGIN_INCLUDE_FILE` event, with a `const char*` file path as GCC data, is triggered for processing of `#include` or `#line` directives.

The `PLUGIN_FINISH` event is the last time that plugins can call GCC functions, notably emit diagnostics with `warning`, `error` etc.

23.3 Interacting with the pass manager

There needs to be a way to add/reorder/remove passes dynamically. This is useful for both analysis plugins (plugging in after a certain pass such as CFG or an IPA pass) and optimization plugins.

Basic support for inserting new passes or replacing existing passes is provided. A plugin registers a new pass with GCC by calling `register_callback` with the `PLUGIN_PASS_MANAGER_SETUP` event and a pointer to a `struct register_pass_info` object defined as follows

```
enum pass_positioning_ops
{
    PASS_POS_INSERT_AFTER, // Insert after the reference pass.
    PASS_POS_INSERT_BEFORE, // Insert before the reference pass.
    PASS_POS_REPLACE       // Replace the reference pass.
};

struct register_pass_info
{
    struct opt_pass *pass;           /* New pass provided by the plugin. */
    const char *reference_pass_name; /* Name of the reference pass for hooking
                                     up the new pass. */
    int ref_pass_instance_number;    /* Insert the pass at the specified
                                     instance number of the reference pass. */
                                     /* Do it for every instance if it is 0. */
    enum pass_positioning_ops pos_op; /* how to insert the new pass. */
};

/* Sample plugin code that registers a new pass. */
int
plugin_init (struct plugin_name_args *plugin_info,
             struct plugin_gcc_version *version)
{
    struct register_pass_info pass_info;
```

```

...

/* Code to fill in the pass_info object with new pass information. */

...

/* Register the new pass. */
register_callback (plugin_info->base_name, PLUGIN_PASS_MANAGER_SETUP, NULL, &pass_info);
...
}

```

23.4 Interacting with the GCC Garbage Collector

Some plugins may want to be informed when GGC (the GCC Garbage Collector) is running. They can register callbacks for the `PLUGIN_GGC_START` and `PLUGIN_GGC_END` events (for which the callback is called with a null `gcc_data`) to be notified of the start or end of the GCC garbage collection.

Some plugins may need to have GGC mark additional data. This can be done by registering a callback (called with a null `gcc_data`) for the `PLUGIN_GGC_MARKING` event. Such callbacks can call the `ggc_set_mark` routine, preferably through the `ggc_mark` macro (and conversely, these routines should usually not be used in plugins outside of the `PLUGIN_GGC_MARKING` event). Plugins that wish to hold weak references to gc data may also use this event to drop weak references when the object is about to be collected. The `ggc_marked_p` function can be used to tell if an object is marked, or is about to be collected. The `gt_clear_cache` overloads which some types define may also be of use in managing weak references.

Some plugins may need to add extra GGC root tables, e.g. to handle their own GTY-ed data. This can be done with the `PLUGIN_REGISTER_GGC_ROOTS` pseudo-event with a null callback and the extra root table (of type `struct ggc_root_tab*`) as `user_data`. Running the `genctype -p source-dir file-list plugin*.c ...` utility generates these extra root tables.

You should understand the details of memory management inside GCC before using `PLUGIN_GGC_MARKING` or `PLUGIN_REGISTER_GGC_ROOTS`.

23.5 Giving information about a plugin

A plugin should give some information to the user about itself. This uses the following structure:

```

struct plugin_info
{
    const char *version;
    const char *help;
};

```

Such a structure is passed as the `user_data` by the plugin's init routine using `register_callback` with the `PLUGIN_INFO` pseudo-event and a null callback.

23.6 Registering custom attributes or pragmas

For analysis (or other) purposes it is useful to be able to add custom attributes or pragmas.

The `PLUGIN_ATTRIBUTES` callback is called during attribute registration. Use the `register_attribute` function to register custom attributes.

```
/* Attribute handler callback */
static tree
handle_user_attribute (tree *node, tree name, tree args,
                      int flags, bool *no_add_attrs)
{
    return NULL_TREE;
}

/* Attribute definition */
static struct attribute_spec user_attr =
{ "user", 1, 1, false, false, false, false, handle_user_attribute, NULL };

/* Plugin callback called during attribute registration.
Registered with register_callback (plugin_name, PLUGIN_ATTRIBUTES, register_attributes, NULL) */
static void
register_attributes (void *event_data, void *data)
{
    warning (0, G_("Callback to register attributes"));
    register_attribute (&user_attr);
}
```

The `PLUGIN_PRAGMAS` callback is called once during pragmas registration. Use the `c_register_pragma`, `c_register_pragma_with_data`, `c_register_pragma_with_expansion`, `c_register_pragma_with_expansion_and_data` functions to register custom pragmas and their handlers (which often want to call `pragma_lex`) from `c-family/c-pragma.h`.

```
/* Plugin callback called during pragmas registration. Registered with
register_callback (plugin_name, PLUGIN_PRAGMAS,
register_my_pragma, NULL);
*/
static void
register_my_pragma (void *event_data, void *data)
{
    warning (0, G_("Callback to register pragmas"));
    c_register_pragma ("GCCPLUGIN", "sayhello", handle_pragma_sayhello);
}
```

It is suggested to pass "GCCPLUGIN" (or a short name identifying your plugin) as the "space" argument of your pragma.

Pragmas registered with `c_register_pragma_with_expansion` or `c_register_pragma_with_expansion_and_data` support preprocessor expansions. For example:

```
#define NUMBER 10
#pragma GCCPLUGIN fofothreshold (NUMBER)
```

23.7 Recording information about pass execution

The event `PLUGIN_PASS_EXECUTION` passes the pointer to the executed pass (the same as `current_pass`) as `gcc_data` to the callback. You can also inspect `cfun` to find out about which function this pass is executed for. Note that this event will only be invoked if the gate check (if applicable, modified by `PLUGIN_OVERRIDE_GATE`) succeeds. You can use other hooks, like `PLUGIN_ALL_PASSES_START`, `PLUGIN_ALL_PASSES_END`,

PLUGIN_ALL_IPA_PASSES_START, PLUGIN_ALL_IPA_PASSES_END, PLUGIN_EARLY_GIMPLE_PASSES_START, and/or PLUGIN_EARLY_GIMPLE_PASSES_END to manipulate global state in your plugin(s) in order to get context for the pass execution.

23.8 Controlling which passes are being run

After the original gate function for a pass is called, its result - the gate status - is stored as an integer. Then the event `PLUGIN_OVERRIDE_GATE` is invoked, with a pointer to the gate status in the `gcc_data` parameter to the callback function. A nonzero value of the gate status means that the pass is to be executed. You can both read and write the gate status via the passed pointer.

23.9 Keeping track of available passes

When your plugin is loaded, you can inspect the various pass lists to determine what passes are available. However, other plugins might add new passes. Also, future changes to GCC might cause generic passes to be added after plugin loading. When a pass is first added to one of the pass lists, the event `PLUGIN_NEW_PASS` is invoked, with the callback parameter `gcc_data` pointing to the new pass.

23.10 Building GCC plugins

If plugins are enabled, GCC installs the headers needed to build a plugin (somewhere in the installation tree, e.g. under `/usr/local`). In particular a `plugin/include` directory is installed, containing all the header files needed to build plugins.

On most systems, you can query this `plugin` directory by invoking `gcc -print-file-name=plugin` (replace if needed `gcc` with the appropriate program path).

Inside plugins, this `plugin` directory name can be queried by calling `default_plugin_dir_name()`.

Plugins may know, when they are compiled, the GCC version for which `plugin-version.h` is provided. The constant macros `GCCPLUGIN_VERSION_MAJOR`, `GCCPLUGIN_VERSION_MINOR`, `GCCPLUGIN_VERSION_PATCHLEVEL`, `GCCPLUGIN_VERSION` are integer numbers, so a plugin could ensure it is built for GCC 4.7 with

```
#if GCCPLUGIN_VERSION != 4007
#error this GCC plugin is for GCC 4.7
#endif
```

The following GNU Makefile excerpt shows how to build a simple plugin:

```
HOST_GCC=g++
TARGET_GCC=gcc
PLUGIN_SOURCE_FILES= plugin1.c plugin2.cc
GCCPLUGINS_DIR:= $(shell $(TARGET_GCC) -print-file-name=plugin)
CXXFLAGS+= -I$(GCCPLUGINS_DIR)/include -fPIC -fno-rtti -O2

plugin.so: $(PLUGIN_SOURCE_FILES)
    $(HOST_GCC) -shared $(CXXFLAGS) $^ -o $@
```

A single source file plugin may be built with `g++ -I`gcc -print-file-name=plugin`/include -fPIC -shared -fno-rtti -O2 plugin.cc -o plugin.so`, using backquote shell syntax to query the `plugin` directory.

Plugin support on Windows/MinGW has a number of limitations and additional requirements. When building a plugin on Windows we have to link an import library for the corresponding backend executable, for example, `cc1.exe`, `cc1plus.exe`, etc., in order to gain access to the symbols provided by GCC. This means that on Windows a plugin is language-specific, for example, for C, C++, etc. If you wish to use your plugin with multiple languages, then you will need to build multiple plugin libraries and either instruct your users on how to load the correct version or provide a compiler wrapper that does this automatically.

Additionally, on Windows the plugin library has to export the `plugin_is_GPL_compatible` and `plugin_init` symbols. If you do not wish to modify the source code of your plugin, then you can use the `-Wl,--export-all-symbols` option or provide a suitable DEF file. Alternatively, you can export just these two symbols by decorating them with `__declspec(dllexport)`, for example:

```
#ifdef _WIN32
__declspec(dllexport)
#endif
int plugin_is_GPL_compatible;

#ifdef _WIN32
__declspec(dllexport)
#endif
int plugin_init (plugin_name_args *, plugin_gcc_version *)
```

The import libraries are installed into the `plugin` directory and their names are derived by appending the `.a` extension to the backend executable names, for example, `cc1.exe.a`, `cc1plus.exe.a`, etc. The following command line shows how to build the single source file plugin on Windows to be used with the C++ compiler:

```
g++ -I`gcc -print-file-name=plugin`/include -shared -Wl,--export-all-symbols \
-o plugin.dll plugin.cc `gcc -print-file-name=plugin`/cc1plus.exe.a
```

When a plugin needs to use `gengtype`, be sure that both `gengtype` and `gtype.state` have the same version as the GCC for which the plugin is built.

24 Link Time Optimization

Link Time Optimization (LTO) gives GCC the capability of dumping its internal representation (GIMPLE) to disk, so that all the different compilation units that make up a single executable can be optimized as a single module. This expands the scope of inter-procedural optimizations to encompass the whole program (or, rather, everything that is visible at link time).

24.1 Design Overview

Link time optimization is implemented as a GCC front end for a bytecode representation of GIMPLE that is emitted in special sections of `.o` files. Currently, LTO support is enabled in most ELF-based systems, as well as darwin, cygwin and mingw systems.

By default, object files generated with LTO support contain only GIMPLE bytecode. Such objects are called “slim”, and they require that tools like `ar` and `nm` understand symbol tables of LTO sections. For most targets these tools have been extended to use the plugin infrastructure, so GCC can support “slim” objects consisting of the intermediate code alone.

GIMPLE bytecode could also be saved alongside final object code if the `-ffat-lto-objects` option is passed, or if no plugin support is detected for `ar` and `nm` when GCC is configured. It makes the object files generated with LTO support larger than regular object files. This “fat” object format allows to ship one set of fat objects which could be used both for development and the production of optimized builds. A, perhaps surprising, side effect of this feature is that any mistake in the toolchain leads to LTO information not being used (e.g. an older `libtool` calling `ld` directly). This is both an advantage, as the system is more robust, and a disadvantage, as the user is not informed that the optimization has been disabled.

At the highest level, LTO splits the compiler in two. The first half (the “writer”) produces a streaming representation of all the internal data structures needed to optimize and generate code. This includes declarations, types, the callgraph and the GIMPLE representation of function bodies.

When `-flto` is given during compilation of a source file, the pass manager executes all the passes in `all_lto_gen_passes`. Currently, this phase is composed of two IPA passes:

- `pass_ipa_lto_gimple_out` This pass executes the function `lto_output` in `lto-streamer-out.cc`, which traverses the call graph encoding every reachable declaration, type and function. This generates a memory representation of all the file sections described below.
- `pass_ipa_lto_finish_out` This pass executes the function `produce_asm_for_decls` in `lto-streamer-out.cc`, which takes the memory image built in the previous pass and encodes it in the corresponding ELF file sections.

The second half of LTO support is the “reader”. This is implemented as the GCC front end `lto1` in `lto/lto.cc`. When `collect2` detects a link set of `.o/.a` files with LTO information and the `-flto` is enabled, it invokes `lto1` which reads the set of files and aggregates them into a single translation unit for optimization. The main entry point for the reader is `lto/lto.cc:lto_main`.

24.1.1 LTO modes of operation

One of the main goals of the GCC link-time infrastructure was to allow effective compilation of large programs. For this reason GCC implements two link-time compilation modes.

1. *LTO mode*, in which the whole program is read into the compiler at link-time and optimized in a similar way as if it were a single source-level compilation unit.
2. *WHOPR or partitioned mode*, designed to utilize multiple CPUs and/or a distributed compilation environment to quickly link large applications. WHOPR stands for WHOLE Program optimizer (not to be confused with the semantics of `-fwhole-program`). It partitions the aggregated callgraph from many different `.o` files and distributes the compilation of the sub-graphs to different CPUs.

Note that distributed compilation is not implemented yet, but since the parallelism is facilitated via generating a `Makefile`, it would be easy to implement.

WHOPR splits LTO into three main stages:

1. Local generation (LGEN) This stage executes in parallel. Every file in the program is compiled into the intermediate language and packaged together with the local call-graph and summary information. This stage is the same for both the LTO and WHOPR compilation mode.
2. Whole Program Analysis (WPA) WPA is performed sequentially. The global call-graph is generated, and a global analysis procedure makes transformation decisions. The global call-graph is partitioned to facilitate parallel optimization during phase 3. The results of the WPA stage are stored into new object files which contain the partitions of program expressed in the intermediate language and the optimization decisions.
3. Local transformations (LTRANS) This stage executes in parallel. All the decisions made during phase 2 are implemented locally in each partitioned object file, and the final object code is generated. Optimizations which cannot be decided efficiently during the phase 2 may be performed on the local call-graph partitions.

WHOPR can be seen as an extension of the usual LTO mode of compilation. In LTO, WPA and LTRANS are executed within a single execution of the compiler, after the whole program has been read into memory.

When compiling in WHOPR mode, the callgraph is partitioned during the WPA stage. The whole program is split into a given number of partitions of roughly the same size. The compiler tries to minimize the number of references which cross partition boundaries. The main advantage of WHOPR is to allow the parallel execution of LTRANS stages, which are the most time-consuming part of the compilation process. Additionally, it avoids the need to load the whole program into memory.

24.2 LTO file sections

LTO information is stored in several ELF sections inside object files. Data structures and enum codes for sections are defined in `lto-streamer.h`.

These sections are emitted from `lto-streamer-out.cc` and mapped in all at once from `lto/lto.cc:lto_file_read`. The individual functions dealing with the reading/writing of each section are described below.

- Command line options (`.gnu.lto_.opts`)

This section contains the command line options used to generate the object files. This is used at link time to determine the optimization level and other settings when they are not explicitly specified at the linker command line.

Most options are recorded at a per function level and their setting restored when processing the functions at link time. Global options are composed from options specified at compile time and link time. How exactly they are combined or mismatches diagnosed is implemented in `lto-wrapper.cc:find_and_merge_options`.

- Symbol table (`.gnu.lto_.symtab`)

This table replaces the ELF symbol table for functions and variables represented in the LTO IL. Symbols used and exported by the optimized assembly code of “fat” objects might not match the ones used and exported by the intermediate code. This table is necessary because the intermediate code is less optimized and thus requires a separate symbol table.

Additionally, the binary code in the “fat” object will lack a call to a function, since the call was optimized out at compilation time after the intermediate language was streamed out. In some special cases, the same optimization may not happen during link-time optimization. This would lead to an undefined symbol if only one symbol table was used.

The symbol table is emitted in `lto-streamer-out.cc:produce_symtab`.

- Global declarations and types (`.gnu.lto_.decls`)

This section contains an intermediate language dump of all declarations and types required to represent the callgraph, static variables and top-level debug info.

The contents of this section are emitted in `lto-streamer-out.cc:produce_asm_for_decls`. Types and symbols are emitted in a topological order that preserves the sharing of pointers when the file is read back in (`lto.cc:read_cgraph_and_symbols`).

- The callgraph (`.gnu.lto_.cgraph`)

This section contains the basic data structure used by the GCC inter-procedural optimization infrastructure. This section stores an annotated multi-graph which represents the functions and call sites as well as the variables, aliases and top-level `asm` statements. This section is emitted in `lto-streamer-out.cc:output_cgraph` and read in `lto-cgraph.cc:input_cgraph`.

- IPA references (`.gnu.lto_.refs`)

This section contains references between function and static variables. It is emitted by `lto-cgraph.cc:output_refs` and read by `lto-cgraph.cc:input_refs`.

- Function bodies (`.gnu.lto_.function_body.<name>`)

This section contains function bodies in the intermediate language representation. Every function body is in a separate section to allow copying of the section independently to different object files or reading the function on demand.

Functions are emitted in `lto-streamer-out.cc:output_function` and read in `lto-streamer-in.cc:input_function`.

- Static variable initializers (`.gnu.lto_.vars`)

This section contains all the symbols in the global variable pool. It is emitted by `lto-cgraph.cc:output_varpool` and read in `lto-cgraph.cc:input_cgraph`.

- Summaries and optimization summaries used by IPA passes (`.gnu.lto_.<xxx>`, where `<xxx>` is one of `jmpfuncs`, `pureconst` or `reference`)

These sections are used by IPA passes that need to emit summary information during LTO generation to be read and aggregated at link time. Each pass is responsible for implementing two pass manager hooks: one for writing the summary and another for reading it in. The format of these sections is entirely up to each individual pass. The only requirement is that the writer and reader hooks agree on the format.

24.3 Using summary information in IPA passes

Programs are represented internally as a *callgraph* (a multi-graph where nodes are functions and edges are call sites) and a *varpool* (a list of static and external variables in the program).

The inter-procedural optimization is organized as a sequence of individual passes, which operate on the callgraph and the varpool. To make the implementation of WHOPR possible, every inter-procedural optimization pass is split into several stages that are executed at different times during WHOPR compilation:

- LGEN time
 1. *Generate summary* (`generate_summary` in `struct ipa_opt_pass_d`). This stage analyzes every function body and variable initializer is examined and stores relevant information into a pass-specific data structure.
 2. *Write summary* (`write_summary` in `struct ipa_opt_pass_d`). This stage writes all the pass-specific information generated by `generate_summary`. Summaries go into their own `LTO_section_*` sections that have to be declared in `lto-streamer.h:enum lto_section_type`. A new section is created by calling `create_output_block` and data can be written using the `lto_output_*` routines.
- WPA time
 1. *Read summary* (`read_summary` in `struct ipa_opt_pass_d`). This stage reads all the pass-specific information in exactly the same order that it was written by `write_summary`.
 2. *Execute* (`execute` in `struct opt_pass`). This performs inter-procedural propagation. This must be done without actual access to the individual function bodies or variable initializers. Typically, this results in a transitive closure operation over the summary information of all the nodes in the callgraph.
 3. *Write optimization summary* (`write_optimization_summary` in `struct ipa_opt_pass_d`). This writes the result of the inter-procedural propagation into the object file. This can use the same data structures and helper routines used in `write_summary`.
- LTRANS time
 1. *Read optimization summary* (`read_optimization_summary` in `struct ipa_opt_pass_d`). The counterpart to `write_optimization_summary`. This reads the interprocedural optimization decisions in exactly the same format emitted by `write_optimization_summary`.
 2. *Transform* (`function_transform` and `variable_transform` in `struct ipa_opt_pass_d`). The actual function bodies and variable initializers are updated based on the information passed down from the *Execute* stage.

The implementation of the inter-procedural passes are shared between LTO, WHOPR and classic non-LTO compilation.

- During the traditional file-by-file mode every pass executes its own *Generate summary*, *Execute*, and *Transform* stages within the single execution context of the compiler.
- In LTO compilation mode, every pass uses *Generate summary* and *Write summary* stages at compilation time, while the *Read summary*, *Execute*, and *Transform* stages are executed at link time.
- In WHOPR mode all stages are used.

To simplify development, the GCC pass manager differentiates between normal inter-procedural passes (see Section 8.4.2 [Regular IPA passes], page 148), small inter-procedural passes (see Section 8.4.1 [Small IPA passes], page 147) and late inter-procedural passes (see Section 8.4.3 [Late IPA passes], page 150). A small or late IPA pass (`SIMPLE_IPA_PASS`) does everything at once and thus cannot be executed during WPA in WHOPR mode. It defines only the *Execute* stage and during this stage it accesses and modifies the function bodies. Such passes are useful for optimization at LGEN or LTRANS time and are used, for example, to implement early optimization before writing object files. The simple inter-procedural passes can also be used for easier prototyping and development of a new inter-procedural pass.

24.3.1 Virtual clones

One of the main challenges of introducing the WHOPR compilation mode was addressing the interactions between optimization passes. In LTO compilation mode, the passes are executed in a sequence, each of which consists of analysis (or *Generate summary*), propagation (or *Execute*) and *Transform* stages. Once the work of one pass is finished, the next pass sees the updated program representation and can execute. This makes the individual passes dependent on each other.

In WHOPR mode all passes first execute their *Generate summary* stage. Then summary writing marks the end of the LGEN stage. At WPA time, the summaries are read back into memory and all passes run the *Execute* stage. Optimization summaries are streamed and sent to LTRANS, where all the passes execute the *Transform* stage.

Most optimization passes split naturally into analysis, propagation and transformation stages. But some do not. The main problem arises when one pass performs changes and the following pass gets confused by seeing different callgraphs between the *Transform* stage and the *Generate summary* or *Execute* stage. This means that the passes are required to communicate their decisions with each other.

To facilitate this communication, the GCC callgraph infrastructure implements *virtual clones*, a method of representing the changes performed by the optimization passes in the callgraph without needing to update function bodies.

A *virtual clone* in the callgraph is a function that has no associated body, just a description of how to create its body based on a different function (which itself may be a virtual clone).

The description of function modifications includes adjustments to the function's signature (which allows, for example, removing or adding function arguments), substitutions to perform on the function body, and, for inlined functions, a pointer to the function that it will be inlined into.

It is also possible to redirect any edge of the callgraph from a function to its virtual clone. This implies updating of the call site to adjust for the new function signature.

Most of the transformations performed by inter-procedural optimizations can be represented via virtual clones. For instance, a constant propagation pass can produce a virtual clone of the function which replaces one of its arguments by a constant. The inliner can represent its decisions by producing a clone of a function whose body will be later integrated into a given function.

Using *virtual clones*, the program can be easily updated during the *Execute* stage, solving most of pass interactions problems that would otherwise occur during *Transform*.

Virtual clones are later materialized in the LTRANS stage and turned into real functions. Passes executed after the virtual clone were introduced also perform their *Transform* stage on new functions, so for a pass there is no significant difference between operating on a real function or a virtual clone introduced before its *Execute* stage.

Optimization passes then work on virtual clones introduced before their *Execute* stage as if they were real functions. The only difference is that clones are not visible during the *Generate Summary* stage.

To keep function summaries updated, the callgraph interface allows an optimizer to register a callback that is called every time a new clone is introduced as well as when the actual function or variable is generated or when a function or variable is removed. These hooks are registered in the *Generate summary* stage and allow the pass to keep its information intact until the *Execute* stage. The same hooks can also be registered during the *Execute* stage to keep the optimization summaries updated for the *Transform* stage.

24.3.2 IPA references

GCC represents IPA references in the callgraph. For a function or variable *A*, the *IPA reference* is a list of all locations where the address of *A* is taken and, when *A* is a variable, a list of all direct stores and reads to/from *A*. References represent an oriented multi-graph on the union of nodes of the callgraph and the varpool. See `ipa-reference.cc:ipa_reference_write_optimization_summary` and `ipa-reference.cc:ipa_reference_read_optimization_summary` for details.

24.3.3 Jump functions

Suppose that an optimization pass sees a function *A* and it knows the values of (some of) its arguments. The *jump function* describes the value of a parameter of a given function call in function *A* based on this knowledge.

Jump functions are used by several optimizations, such as the inter-procedural constant propagation pass and the devirtualization pass. The inliner also uses jump functions to perform inlining of callbacks.

24.4 Whole program assumptions, linker plugin and symbol visibilities

Link-time optimization gives relatively minor benefits when used alone. The problem is that propagation of inter-procedural information does not work well across functions and variables that are called or referenced by other compilation units (such as from a dynamically linked library). We say that such functions and variables are *externally visible*.

To make the situation even more difficult, many applications organize themselves as a set of shared libraries, and the default ELF visibility rules allow one to overwrite any externally visible symbol with a different symbol at runtime. This basically disables any optimizations across such functions and variables, because the compiler cannot be sure that the function body it is seeing is the same function body that will be used at runtime. Any function or variable not declared `static` in the sources degrades the quality of inter-procedural optimization.

To avoid this problem the compiler must assume that it sees the whole program when doing link-time optimization. Strictly speaking, the whole program is rarely visible even at link-time. Standard system libraries are usually linked dynamically or not provided with the link-time information. In GCC, the whole program option (`-fwhole-program`) asserts that every function and variable defined in the current compilation unit is static, except for function `main` (note: at link time, the current unit is the union of all objects compiled with LTO). Since some functions and variables need to be referenced externally, for example by another DSO or from an assembler file, GCC also provides the function and variable attribute `externally_visible` which can be used to disable the effect of `-fwhole-program` on a specific symbol.

The whole program mode assumptions are slightly more complex in C++, where inline functions in headers are put into *COMDAT* sections. COMDAT function and variables can be defined by multiple object files and their bodies are unified at link-time and dynamic link-time. COMDAT functions are changed to local only when their address is not taken and thus un-sharing them with a library is not harmful. COMDAT variables always remain externally visible, however for readonly variables it is assumed that their initializers cannot be overwritten by a different value.

GCC provides the function and variable attribute `visibility` that can be used to specify the visibility of externally visible symbols (or alternatively an `-fdefault-visibility` command line option). ELF defines the `default`, `protected`, `hidden` and `internal` visibilities.

The most commonly used is visibility is `hidden`. It specifies that the symbol cannot be referenced from outside of the current shared library. Unfortunately, this information cannot be used directly by the link-time optimization in the compiler since the whole shared library also might contain non-LTO objects and those are not visible to the compiler.

GCC solves this problem using linker plugins. A *linker plugin* is an interface to the linker that allows an external program to claim the ownership of a given object file. The linker then performs the linking procedure by querying the plugin about the symbol table of the claimed objects and once the linking decisions are complete, the plugin is allowed to provide the final object file before the actual linking is made. The linker plugin obtains the symbol resolution information which specifies which symbols provided by the claimed objects are bound from the rest of a binary being linked.

GCC is designed to be independent of the rest of the toolchain and aims to support linkers without plugin support. For this reason it does not use the linker plugin by default. Instead, the object files are examined by `collect2` before being passed to the linker and objects found to have LTO sections are passed to `lto1` first. This mode does not work for library archives. The decision on what object files from the archive are needed depends on the actual linking and thus GCC would have to implement the linker itself. The resolution information is missing too and thus GCC needs to make an educated guess based on `-fwhole-program`.

Without the linker plugin GCC also assumes that symbols are declared `hidden` and not referred by non-LTO code by default.

24.5 Internal flags controlling `lto1`

The following flags are passed into `lto1` and are not meant to be used directly from the command line.

- `-fwpa` This option runs the serial part of the link-time optimizer performing the inter-procedural propagation (WPA mode). The compiler reads in summary information from all inputs and performs an analysis based on summary information only. It generates object files for subsequent runs of the link-time optimizer where individual object files are optimized using both summary information from the WPA mode and the actual function bodies. It then drives the LTRANS phase.
- `-fltrans` This option runs the link-time optimizer in the local-transformation (LTRANS) mode, which reads in output from a previous run of the LTO in WPA mode. In the LTRANS mode, LTO optimizes an object and produces the final assembly.
- `-fltrans-output-list=file` This option specifies a file to which the names of LTRANS output files are written. This option is only meaningful in conjunction with `-fwpa`.
- `-fresolution=file` This option specifies the linker resolution file. This option is only meaningful in conjunction with `-fwpa` and as option to pass through to the LTO linker plugin.

25 Match and Simplify

The GIMPLE and GENERIC pattern matching project match-and-simplify tries to address several issues.

1. unify expression simplifications currently spread and duplicated over separate files like fold-const.cc, gimple-fold.cc and builtins.cc
2. allow for a cheap way to implement building and simplifying non-trivial GIMPLE expressions, avoiding the need to go through building and simplifying GENERIC via fold_buildN and then gimplifying via force_gimple_operand

To address these the project introduces a simple domain-specific language to write expression simplifications from which code targeting GIMPLE and GENERIC is auto-generated. The GENERIC variant follows the fold_buildN API while for the GIMPLE variant and to address 2) new APIs are introduced.

25.1 GIMPLE API

```
tree gimple_simplify (enum tree_code, tree, tree,      [GIMPLE function]
                     gimple_seq *, tree (*)(tree))
tree gimple_simplify (enum tree_code, tree, tree,      [GIMPLE function]
                     tree, gimple_seq *, tree (*)(tree))
tree gimple_simplify (enum tree_code, tree, tree,      [GIMPLE function]
                     tree, tree, gimple_seq *, tree (*)(tree))
tree gimple_simplify (enum built_in_function, tree,    [GIMPLE function]
                     tree, gimple_seq *, tree (*)(tree))
tree gimple_simplify (enum built_in_function, tree,    [GIMPLE function]
                     tree, tree, gimple_seq *, tree (*)(tree))
tree gimple_simplify (enum built_in_function, tree,    [GIMPLE function]
                     tree, tree, tree, gimple_seq *, tree (*)(tree))
```

The main GIMPLE API entry to the expression simplifications mimicking that of the GENERIC fold_{unary,binary,ternary} functions.

thus providing n-ary overloads for operation or function. The additional arguments are a gimple_seq where built statements are inserted on (if NULL then simplifications requiring new statements are not performed) and a valueization hook that can be used to tie simplifications to a SSA lattice.

In addition to those APIs fold_stmt is overloaded with a valueization hook:

```
fold_stmt (gimple_stmt_iterator *, tree (*)(tree));      [bool]
```

On top of these a fold_buildN-like API for GIMPLE is introduced:

```
tree gimple_build (gimple_seq *, location_t, enum      [GIMPLE function]
                  tree_code, tree, tree, tree (*valueize) (tree) = NULL);
tree gimple_build (gimple_seq *, location_t, enum      [GIMPLE function]
                  tree_code, tree, tree, tree, tree (*valueize) (tree) = NULL);
tree gimple_build (gimple_seq *, location_t, enum      [GIMPLE function]
                  tree_code, tree, tree, tree, tree, tree (*valueize) (tree) =
                  NULL);
```

```

tree gimple_build (gimple_seq *, location_t, enum      [GIMPLE function]
                  built_in_function, tree, tree, tree (*valueize) (tree) =
                  NULL);
tree gimple_build (gimple_seq *, location_t, enum      [GIMPLE function]
                  built_in_function, tree, tree, tree, tree (*valueize) (tree) =
                  NULL);
tree gimple_build (gimple_seq *, location_t, enum      [GIMPLE function]
                  built_in_function, tree, tree, tree, tree, tree (*valueize)
                  (tree) = NULL);
tree gimple_convert (gimple_seq *, location_t,          [GIMPLE function]
                    tree, tree);

```

which is supposed to replace `force_gimple_operand (fold_buildN (...), ...)` and calls to `fold_convert`. Overloads without the `location_t` argument exist. Built statements are inserted on the provided sequence and simplification is performed using the optional valueization hook.

25.2 The Language

The language in which to write expression simplifications resembles other domain-specific languages GCC uses. Thus it is lisp-y. Let's start with an example from the `match.pd` file:

```

(simplify
  (bit_and @0 integer_all_onesp)
  @0)

```

This example contains all required parts of an expression simplification. A simplification is wrapped inside a `(simplify ...)` expression. That contains at least two operands - an expression that is matched with the GIMPLE or GENERIC IL and a replacement expression that is returned if the match was successful.

Expressions have an operator ID, `bit_and` in this case. Expressions can be lower-case tree codes with `_expr` stripped off or builtin function code names in all-caps, like `BUILT_IN_SQRT`.

`@n` denotes a so-called capture. It captures the operand and lets you refer to it in other places of the match-and-simplify. In the above example it is referred to in the replacement expression. Captures are `@` followed by a number or an identifier.

```

(simplify
  (bit_xor @0 @0)
  { build_zero_cst (type); })

```

In this example `@0` is mentioned twice which constrains the matched expression to have two equal operands. Usually matches are constrained to equal types. If operands may be constants and conversions are involved, matching by value might be preferred in which case use `@@0` to denote a by-value match and the specific operand you want to refer to in the result part. This example also introduces operands written in C code. These can be used in the expression replacements and are supposed to evaluate to a tree node which has to be a valid GIMPLE operand (so you cannot generate expressions in C code).

```

(simplify
  (trunc_mod integer_zerop@0 @1)
  (if (!integer_zerop (@1))
    @0))

```

Here @0 captures the first operand of the trunc_mod expression which is also predicated with `integer_zerop`. Expression operands may be either expressions, predicates or captures. Captures can be unconstrained or capture expressions or predicates.

This example introduces an optional operand of `simplify`, the `if-expression`. This condition is evaluated after the expression matched in the IL and is required to evaluate to true to enable the replacement expression in the second operand position. The expression operand of the `if` is a standard C expression which may contain references to captures. The `if` has an optional third operand which may contain the replacement expression that is enabled when the condition evaluates to false.

A `if` expression can be used to specify a common condition for multiple `simplify` patterns, avoiding the need to repeat that multiple times:

```
(if (!TYPE_SATURATING (type)
    && !FLOAT_TYPE_P (type) && !FIXED_POINT_TYPE_P (type))
  (simplify
    (minus (plus @0 @1) @0)
    @1)
  (simplify
    (minus (minus @0 @1) @0)
    (negate @1)))
```

Note that `ifs` in outer position do not have the optional `else` clause but instead have multiple `then` clauses.

`ifs` can be nested.

There exists a `switch` expression which can be used to chain conditions avoiding nesting `ifs` too much:

```
(simplify
  (simple_comparison @0 REAL_CST@1)
  (switch
    /* a CMP (-0) -> a CMP 0 */
    (if (REAL_VALUE_MINUS_ZERO (TREE_REAL_CST (@1)))
      (cmp @0 { build_real (TREE_TYPE (@1), dconst0); })))
    /* x != NaN is always true, other ops are always false. */
    (if (REAL_VALUE_ISNAN (TREE_REAL_CST (@1))
        && !HONOR_SNANS (@1))
      { constant_boolean_node (cmp == NE_EXPR, type); })))
```

Is equal to

```
(simplify
  (simple_comparison @0 REAL_CST@1)
  (switch
    /* a CMP (-0) -> a CMP 0 */
    (if (REAL_VALUE_MINUS_ZERO (TREE_REAL_CST (@1)))
      (cmp @0 { build_real (TREE_TYPE (@1), dconst0); })))
    /* x != NaN is always true, other ops are always false. */
    (if (REAL_VALUE_ISNAN (TREE_REAL_CST (@1))
        && !HONOR_SNANS (@1))
      { constant_boolean_node (cmp == NE_EXPR, type); }))))
```

which has the second `if` in the `else` operand of the first. The `switch` expression takes `if` expressions as operands (which may not have `else` clauses) and as a last operand a replacement expression which should be enabled by default if no other condition evaluated to true.

Captures can also be used for capturing results of sub-expressions.

```
#if GIMPLE
```

```

(simplify
  (pointer_plus (addr@2 @0) INTEGER_CST_P@1)
  (if (is_gimple_min_invariant (@2)))
  {
    poly_int64 off;
    tree base = get_addr_base_and_unit_offset (@0, &off);
    off += tree_to_uhwi (@1);
    /* Now with that we should be able to simply write
       (addr (mem_ref (addr @base) (plus @off @1))) */
    build1 (ADDR_EXPR, type,
            build2 (MEM_REF, TREE_TYPE (TREE_TYPE (@2)),
                    build_fold_addr_expr (base),
                    build_int_cst (ptr_type_node, off)));
  })
#endif

```

In the above example, @2 captures the result of the expression (addr @0). For the outermost expression only its type can be captured, and the keyword `type` is reserved for this purpose. The above example also gives a way to conditionalize patterns to only apply to GIMPLE or GENERIC by means of using the pre-defined preprocessor macros GIMPLE and GENERIC and using preprocessor directives.

```

(simplify
  (bit_and:c integral_op_p@0 (bit_ior:c (bit_not @0) @1))
  (bit_and @1 @0))

```

Here we introduce flags on match expressions. The flag used above, `c`, denotes that the expression should be also matched commutated. Thus the above match expression is really the following four match expressions:

```

(bit_and integral_op_p@0 (bit_ior (bit_not @0) @1))
(bit_and (bit_ior (bit_not @0) @1) integral_op_p@0)
(bit_and integral_op_p@0 (bit_ior @1 (bit_not @0)))
(bit_and (bit_ior @1 (bit_not @0)) integral_op_p@0)

```

Usual canonicalizations you know from GENERIC expressions are applied before matching, so for example constant operands always come second in commutative expressions.

The second supported flag is `s` which tells the code generator to fail the pattern if the expression marked with `s` does have more than one use and the simplification results in an expression with more than one operator. For example in

```

(simplify
  (pointer_plus (pointer_plus:s @0 @1) @3)
  (pointer_plus @0 (plus @1 @3)))

```

this avoids the association if (pointer_plus @0 @1) is used outside of the matched expression and thus it would stay live and not trivially removed by dead code elimination. Now consider $((x + 3) + -3)$ with the temporary holding $(x + 3)$ used elsewhere. This simplifies down to x which is desirable and thus flagging with `s` does not prevent the transform. Now consider $((x + 3) + 1)$ which simplifies to $(x + 4)$. Despite being flagged with `s` the simplification will be performed. The simplification of $((x + a) + 1)$ to $(x + (a + 1))$ will not be performed in this case though.

More features exist to avoid too much repetition.

```

(for op (plus pointer_plus minus bit_ior bit_xor)
  (simplify
    (op @0 integer_zerop)
    @0))

```

A **for** expression can be used to repeat a pattern for each operator specified, substituting **op**. **for** can be nested and a **for** can have multiple operators to iterate.

```
(for opa (plus minus)
  opb (minus plus)
  (for opc (plus minus)
    (simplify...
```

In this example the pattern will be repeated four times with **opa**, **opb**, **opc** being **plus**, **minus**, **plus**; **plus**, **minus**, **minus**; **minus**, **plus**, **plus**; **minus**, **plus**, **minus**.

To avoid repeating operator lists in **for** you can name them via

```
(define_operator_list pmm plus minus mult)
```

and use them in **for** operator lists where they get expanded.

```
(for opa (pmm trunc_div)
  (simplify...
```

So this example iterates over **plus**, **minus**, **mult** and **trunc_div**.

Using operator lists can also remove the need to explicitly write a **for**. All operator list uses that appear in a **simplify** or **match** pattern in operator positions will implicitly be added to a new **for**. For example

```
(define_operator_list Sqrt BUILT_IN_SQRTF BUILT_IN_SQRT BUILT_IN_SQRTL)
(define_operator_list POW BUILT_IN_POWF BUILT_IN_POW BUILT_IN_POWL)
(simplify
  (Sqrt (Pow @0 @1))
  (Pow (abs @0) (mult @1 { built_real (TREE_TYPE (@1), dconsthalf); })))
```

is the same as

```
(for Sqrt (BUILT_IN_SQRTF BUILT_IN_SQRT BUILT_IN_SQRTL)
  POW (BUILT_IN_POWF BUILT_IN_POW BUILT_IN_POWL)
  (simplify
    (Sqrt (Pow @0 @1))
    (Pow (abs @0) (mult @1 { built_real (TREE_TYPE (@1), dconsthalf); }))))
```

fors and operator lists can include the special identifier **null** that matches nothing and can never be generated. This can be used to pad an operator list so that it has a standard form, even if there isn't a suitable operator for every form.

Another building block are **with** expressions in the result expression which nest the generated code in a new C block followed by its argument:

```
(simplify
  (convert (mult @0 @1))
  (with { tree utype = unsigned_type_for (type); }
    (convert (mult (convert:utype @0) (convert:utype @1)))))
```

This allows code nested in the **with** to refer to the declared variables. In the above case we use the feature to specify the type of a generated expression with the **:type** syntax where **type** needs to be an identifier that refers to the desired type. Usually the types of the generated result expressions are determined from the context, but sometimes like in the above case it is required that you specify them explicitly.

Another modifier for generated expressions is **^** which tells the machinery to try more matches for some special cases. For example, normally the **cond** only allows the **gimple** assign when matching. It will also try to match the **gimple PHI** besides **gimple assign** if appending the **^** to the **cond**. Aka **cond^**. Consider below example

```
(match (unsigned_sat_add @0 @1)
  (cond^ (ge (plus:c02 @0 @1) @0) @2 integer_minus_onep))
```

The above matching will generate the predicate function named `gimple_unsigned_sat_add` that accepts both the gimple assign and gimple PHI.

Another modifier for generated expressions is `!` which tells the machinery to only consider the simplification in case the marked expression simplified to a simple operand. Consider for example

```
(simplify
  (plus (vec_cond:s @0 @1 @2) @3)
  (vec_cond @0 (plus! @1 @3) (plus! @2 @3)))
```

which moves the outer `plus` operation to the inner arms of the `vec_cond` expression but only if the actual plus operations both simplify. Note that on `GENERIC` a simple operand means that the result satisfies `!EXPR_P` which can be limiting if the operation itself simplifies but the remaining operand is an (unrelated) expression.

As intermediate conversions are often optional there is a way to avoid the need to repeat patterns both with and without such conversions. Namely you can mark a conversion as being optional with a `?`:

```
(simplify
  (eq (convert@0 @1) (convert? @2))
  (eq @1 (convert @2)))
```

which will match both `(eq (convert @1) (convert @2))` and `(eq (convert @1) @2)`. The optional converts are supposed to be all either present or not, thus `(eq (convert? @1) (convert? @2))` will result in two patterns only. If you want to match all four combinations you have access to two additional conditional converts as in `(eq (convert1? @1) (convert2? @2))`.

The support for `?` marking extends to all unary operations including predicates you declare yourself with `match`.

Predicates available from the GCC middle-end need to be made available explicitly via `define_predicates`:

```
(define_predicates
  integer_onep integer_zerop integer_all_onesp)
```

You can also define predicates using the pattern matching language and the `match` form:

```
(match negate_expr_p
  INTEGER_CST
  (if (TYPE_OVERFLOW_WRAPS (type)
      || may_negate_without_overflow_p (t))))
(match negate_expr_p
  (negate @0))
```

This shows that for `match` expressions there is `t` available which captures the outermost expression (something not possible in the `simplify` context). As you can see `match` has an identifier as first operand which is how you refer to the predicate in patterns. Multiple `match` for the same identifier add additional cases where the predicate matches.

Predicates can also match an expression in which case you need to provide a template specifying the identifier and where to get its operands from:

```
(match (logical_inverted_value @0)
  (eq @0 integer_zerop))
(match (logical_inverted_value @0)
  (bit_not truth_valued_p@0))
```

You can use the above predicate like

```
(simplify
```

```
(bit_and @0 (logical_inverted_value @0))  
{ build_zero_cst (type); }
```

Which will match a bitwise and of an operand with its logical inverted value.

26 Static Analyzer

26.1 Analyzer Internals

26.1.1 Overview

At a high-level, we’re doing coverage-guided symbolic execution of the user’s code.

The analyzer implementation works on the gimple-SSA representation. (I chose this in the hopes of making it easy to work with LTO to do whole-program analysis).

The implementation is read-only: it doesn’t attempt to change anything, just emit warnings.

The gimple representation can be seen using `-fdump-ipa-analyzer`.

Tip: If the analyzer ICEs before this is written out, one workaround is to use `--param=analyzer-bb-explosion-factor=0` to force the analyzer to bail out after analyzing the first basic block.

First, we build a directed graph to represent the user’s code. For historical reasons we call this the **supergraph**, although this is now a misnomer as we no longer add callgraph edges to this graph. The nodes and edges in the supergraph are called “supernodes” and “superedges”, and often referred to in code as **snodes** and **sedges**.

We make a node in the supergraph before every gimple statement, with edges representing the transitions between statements within a basic block, along with additional nodes and edges at CFG edges.

The nodes in the supergraph represent locations in the user’s code, and discrete points between operations. The edges represent transitions between these locations. Each edge in the supergraph can have an optional **operation** associated with it, representing a single state transition that occurs along the edge, such as

- individual non-control-flow gimple statements (such as an assignment)
- control flow statements on a CFG edge that impose a condition for the transition to be possible (e.g. a branch of a conditional or a **switch** case)
- the collection of phi nodes at the entry to a basic block, with an associated CFG edge (so that these all take effect simultaneously)
- etc

There can be multiple nodes and edges in the supergraph corresponding to a single CFG edge so that e.g. we can handle filtering states on a condition separately from handling the effect of the phi nodes if the condition was satisfied.

The analyzer in GCC 10 - GCC 15 attempted to have a single supernode per basic block for the sake of efficiency, but given that state transitions can happen mid-block, this became unmaintainable, hence we now have fine-grained nodes with one node/edge per gimple statement.

Having built the supergraph from the CFGs of all of the functions in the user’s code, we manipulate it:

- We fixup locations to try to ensure that every supernode has a reasonable **location_t** value referring to the location in the user’s source. This is necessary, since in the

gimple IR seen by the analyzer, many gimple statements have no location associated with them.

- We simplify the supergraph to remove redundant nodes and edges, such as those that are simply no-ops that add no useful location information. This can eliminate about 5-10% of the nodes.
- We sort and renumber the nodes into an order that we hope will lead to efficient state merging when exploring the graph (see below).

The supergraph can be seen at each stage using `-fdump-analyzer-supergraph`, which creates a series of `SRC.supergraph.N.KIND.dot` GraphViz files showing the state of the supergraph after each of the above.

We then build an `analysis_plan` which walks the callgraph to determine which calls might be suitable for being summarized (rather than fully explored) and thus in what order to explore the functions.

Next is the heart of the analyzer: we use a worklist to explore state within the supergraph, building an "exploded graph". Nodes in the exploded graph correspond to `<point, state>` pairs, as in "Precise Interprocedural Dataflow Analysis via Graph Reachability" (Thomas Reps, Susan Horwitz and Mooly Sagiv) - but note that we're not using the algorithm described in that paper, just the "exploded graph" terminology.

We reuse nodes for `<point, state>` pairs we've already seen, and avoid tracking state too closely, so that (hopefully) we rapidly converge on a final exploded graph, and terminate the analysis. We also bail out if the number of exploded `<point, state>` nodes gets larger than a particular multiple of the total number of supernodes, (to ensure termination in the face of pathological state-explosion cases, or bugs). We also stop exploring a point once we hit a limit of states for that point.

We can identify problems directly when processing a `<point, state>` instance. For example, if we're finding the successors of

```
<point: before-stmt: "free (ptr);",
  state: {"ptr": freed}>
```

then we can detect a double-free of "ptr". We can then emit a path to reach the problem by finding the simplest route through the graph.

Program points in the analysis are a combination of a supernode together with a "call string" identifying the stack of callsites below them, so that paths in the exploded graph correspond to interprocedurally valid paths: we always return to the correct call site, propagating state information accordingly. We avoid infinite recursion by stopping the analysis if a callsite appears more than `analyzer-max-recursion-depth` in a callstring (defaulting to 2).

26.1.2 Graphs

Nodes and edges in the exploded graph are called "exploded nodes" and "exploded edges" and often referred to in the code as `enodes` and `eedges` (especially when distinguishing them from the `snodes` and `sedges` in the supergraph).

Each graph numbers its nodes, giving unique identifiers - supernodes are referred to throughout dumps in the form `'SN': index` and exploded nodes in the form `'EN: index`' (e.g. `'SN: 2'` and `'EN:29'`).

The supergraph can be seen using `-fdump-analyzer-supergraph`.

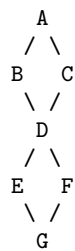
The exploded graph can be seen using `-fdump-analyzer-exploded-graph` and other dump options. Exploded nodes are color-coded in the .dot output based on state-machine states to make it easier to see state changes at a glance.

26.1.3 State Tracking

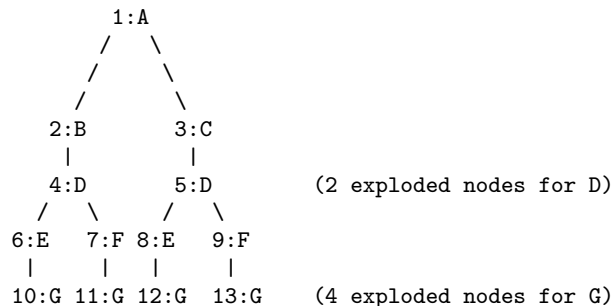
There's a tension between:

- precision of analysis in the straight-line case, vs
- exponential blow-up in the face of control flow.

For example, in general, given this CFG:



we want to avoid differences in state-tracking in B and C from leading to blow-up. If we don't prevent state blowup, we end up with exponential growth of the exploded graph like this:



Similar issues arise with loops.

To prevent this, we follow various approaches:

- state pruning: which tries to discard state that won't be relevant later on withing the function. This can be disabled via `-fno-analyzer-state-purge`.
- state merging. We can try to find the commonality between two `program_state` instances to make a third, simpler `program_state`. We have two strategies here:
 - the worklist keeps new nodes for the same `program_point` together, and tries to merge them before processing, and thus before they have successors. Hence, in the above, the two nodes for D (4 and 5) reach the front of the worklist together, and we create a node for D with the merger of the incoming states.
 - try merging with the state of existing enodes for the `program_point` (which may have already been explored). There will be duplication, but only one set of duplication; subsequent duplicates are more likely to hit the cache. In particular,

(hopefully) all merger chains are finite, and so we guarantee termination. This is intended to help with loops: we ought to explore the first iteration, and then have a "subsequent iterations" exploration, which uses a state merged from that of the first, to be more abstract.

We avoid merging pairs of states that have state-machine differences, as these are the kinds of differences that are likely to be most interesting. So, for example, given:

```

if (condition)
    ptr = malloc (size);
else
    ptr = local_buf;

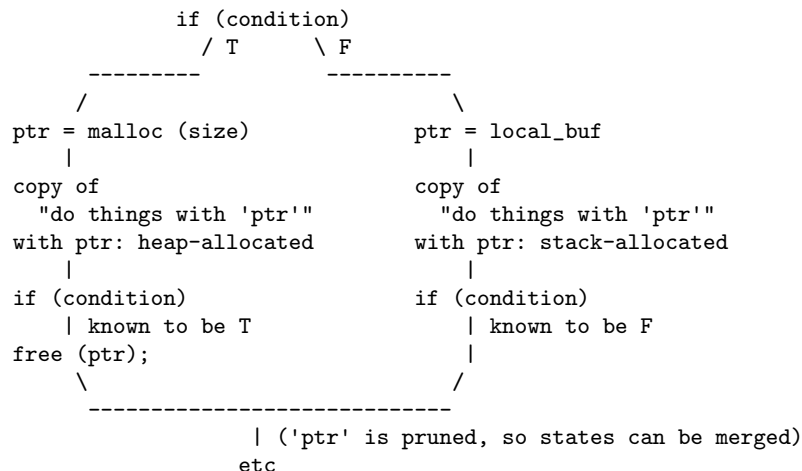
.... do things with 'ptr'

if (condition)
    free (ptr);

...etc

```

then we end up with an exploded graph that looks like this:



where some duplication has occurred, but only for the places where the the different paths are worth exploring separately.

Merging can be disabled via `-fno-analyzer-state-merge`.

26.1.4 Region Model

Part of the state stored at a `exploded_node` is a `region_model`. This is an implementation of the region-based ternary model described in "A Memory Model for Static Analysis of C Programs" (https://www.researchgate.net/publication/221430855_A_Memory_Model_for_Static_Analysis_of_C_Programs) (Zhongxing Xu, Ted Kremenek, and Jian Zhang).

A `region_model` encapsulates a representation of the state of memory, with a `store` recording a binding between `region` instances, to `svalue` instances. The bindings are organized into clusters, where regions accessible via well-defined pointer arithmetic are in

the same cluster. The representation is graph-like because values can be pointers to regions. It also stores a `constraint_manager`, capturing relationships between the values.

Because each node in the `exploded_graph` has a `region_model`, and each of the latter is graph-like, the `exploded_graph` is in some ways a graph of graphs.

There are several “dump” functions for use when debugging the analyzer.

Consider this example C code:

```
void *
calls_malloc (size_t n)
{
    void *result = malloc (1024);
    return result; /* HERE */
}

void test (size_t n)
{
    void *ptr = calls_malloc (n * 4);
    /* etc. */
}
```

and the state at the point `/* HERE */` for the interprocedural analysis case where `calls_malloc` returns back to `test`.

Here’s an example of printing a `program_state` at `/* HERE */`, showing the `region_model` within it, along with state for the `malloc` state machine.

```
(gdb) break region_model::on_return
[..snip...]
(gdb) run
[..snip...]
(gdb) up
[..snip...]
(gdb) call state->dump()
State
Region Model
Current Frame: frame: 'calls_malloc'@2
Store
  m_called_unknown_fn: false
  frame: 'test'@1
  _1: (INIT_VAL(n_2(D)))*(size_t)4
  frame: 'calls_malloc'@2
  result_4: &HEAP_ALLOCATED_REGION(27)
  _5: &HEAP_ALLOCATED_REGION(27)
Dynamic Extents
  HEAP_ALLOCATED_REGION(27): (INIT_VAL(n_2(D)))*(size_t)4
'malloc' state machine
0x468cb40: &HEAP_ALLOCATED_REGION(27): unchecked ({free}) ('result_4')
```

Within the store, there are bindings clusters for the SSA names for the various local variables within frames for `test` and `calls_malloc`. For example,

- within `test` the whole cluster for `_1` is bound to a `binop_svalue` representing `n * 4`, and
- within `test` the whole cluster for `result_4` is bound to a `region_svalue` pointing at `HEAP_ALLOCATED_REGION(12)`.

Additionally, this latter pointer has the `unchecked` state for the `malloc` state machine indicating it hasn’t yet been checked against `NULL` since the allocation call.

We also see that the state has captured the size of the heap-allocated region (“Dynamic Extents”).

This visualization can also be seen within the output of `-fdump-analyzer-exploded-nodes-2` and `-fdump-analyzer-exploded-nodes-3`.

As well as the above visualizations of states, there are tree-like visualizations for instances of `svalue` and `region`, showing their IDs and how they are constructed from simpler symbols:

```
(gdb) break region_model::set_dynamic_extents
[..snip...]
(gdb) run
[..snip...]
(gdb) up
[..snip...]
(gdb) call size_in_bytes->dump()
(17): 'long unsigned int': binop_svalue(mult_expr: '*')
(15): 'size_t': initial_svalue
      m_reg: (12): 'size_t': decl_region('n_2(D)')
            parent: (9): frame_region('test', index: 0, depth: 1)
                  parent: (1): stack region
                        parent: (0): root region
(16): 'size_t': constant_svalue ('4')
```

i.e. that `size_in_bytes` is a `binop_svalue` expressing the result of multiplying

- the initial value of the `PARAM_DECL n_2(D)` for the parameter `n` within the frame for `test` by
- the constant value 4.

The above visualizations rely on the `text_art::widget` framework, which performs significant work to lay out the output, so there is also an earlier, simpler, form of dumping available. For states there is:

```
(gdb) call state->dump(eg.m_ext_state, true)
rmodel:
stack depth: 2
  frame (index 1): frame: 'calls_malloc'@2
  frame (index 0): frame: 'test'@1
clusters within frame: 'test'@1
  cluster for: _1: (INIT_VAL(n_2(D))*(size_t)4)
clusters within frame: 'calls_malloc'@2
  cluster for: result_4: &HEAP_ALLOCATED_REGION(27)
  cluster for: _5: &HEAP_ALLOCATED_REGION(27)
m_called_unknown_fn: FALSE
constraint_manager:
  equiv classes:
  constraints:
dynamic_extents:
  HEAP_ALLOCATED_REGION(27): (INIT_VAL(n_2(D))*(size_t)4)
malloc:
  0x468cb40: &HEAP_ALLOCATED_REGION(27): unchecked ({free}) ('result_4')
```

or for `region_model` just:

```
(gdb) call state->m_region_model->debug()
stack depth: 2
  frame (index 1): frame: 'calls_malloc'@2
  frame (index 0): frame: 'test'@1
clusters within frame: 'test'@1
```

```

    cluster for: _1: (INIT_VAL(n_2(D))*(size_t)4)
clusters within frame: 'calls_malloc'@2
    cluster for: result_4: &HEAP_ALLOCATED_REGION(27)
    cluster for: _5: &HEAP_ALLOCATED_REGION(27)
m_called_unknown_fn: FALSE
constraint_manager:
    equiv classes:
    constraints:
dynamic_extents:
    HEAP_ALLOCATED_REGION(27): (INIT_VAL(n_2(D))*(size_t)4)

```

and for instances of `svalue` and `region` there is this older dump implementation, which takes a `bool simple` flag controlling the verbosity of the dump:

```

(gdb) call size_in_bytes->dump(true)
(INIT_VAL(n_2(D))*(size_t)4)

(gdb) call size_in_bytes->dump(false)
binop_svalue (mult_expr, initial_svalue('size_t', decl_region(frame_region('test', index: 0, depth: 1), '

```

26.1.5 Analyzer Paths

We need to explain to the user what the problem is, and to persuade them that there really is a problem. Hence having a `diagnostics::paths::path` isn't just an incidental detail of the analyzer; it's required.

Paths ought to be:

- interprocedurally-valid
- feasible

Without state-merging, all paths in the exploded graph are feasible (in terms of constraints being satisfied). With state-merging, paths in the exploded graph can be infeasible.

We collate warnings and only emit them for the simplest path e.g. for a bug in a utility function, with lots of routes to calling it, we only emit the simplest path (which could be intraprocedural, if it can be reproduced without a caller).

We thus want to find the shortest feasible path through the exploded graph from the origin to the exploded node at which the diagnostic was saved. Unfortunately, if we simply find the shortest such path and check if it's feasible we might falsely reject the diagnostic, as there might be a longer path that is feasible. Examples include the cases where the diagnostic requires us to go at least once around a loop for a later condition to be satisfied, or where for a later condition to be satisfied we need to enter a suite of code that the simpler path skips.

We attempt to find the shortest feasible path to each diagnostic by first constructing a “trimmed graph” from the exploded graph, containing only those nodes and edges from which there are paths to the target node, and using Dijkstra's algorithm to order the trimmed nodes by minimal distance to the target.

We then use a worklist to iteratively build a “feasible graph” (actually a tree), capturing the pertinent state along each path, in which every path to a “feasible node” is feasible by construction, restricting ourselves to the trimmed graph to ensure we stay on target, and ordering the worklist so that the first feasible path we find to the target node is the shortest possible path. Hence we start by trying the shortest possible path, but if that fails, we explore progressively longer paths, eventually trying iterations through loops. The

exploration is captured in the `feasible_graph`, which can be dumped as a `.dot` file via `-fdump-analyzer-feasibility` to visualize the exploration. The indices of the feasible nodes show the order in which they were created. We effectively explore the tree of feasible paths in order of shortest path until we either find a feasible path to the target node, or hit a limit and give up.

This is something of a brute-force approach, but the trimmed graph hopefully keeps the complexity manageable.

This algorithm can be disabled (for debugging purposes) via `-fno-analyzer-feasibility`, which simply uses the shortest path, and notes if it is infeasible.

The above gives us a shortest feasible `exploded_path` through the `exploded_graph` (a list of `exploded_edge *`). We use this `exploded_path` to build a `diagnostics::paths::path` (a list of `events` for the diagnostic subsystem) - specifically a `checker_path`.

Having built the `checker_path`, we prune it to try to eliminate events that aren't relevant, to minimize how much the user has to read.

After pruning, we notify each event in the path of its ID and record the IDs of interesting events, allowing for events to refer to other events in their descriptions. The `pending_diagnostic` class has various vfuncs to support emitting more precise descriptions, so that e.g.

- a deref-of-unchecked-malloc diagnostic might use:

```
returning possibly-NULL pointer to 'make_obj' from 'allocator'
```

for a `return_event` to make it clearer how the unchecked value moves from callee back to caller
- a double-free diagnostic might use:

```
second 'free' here; first 'free' was at (3)
```

and a use-after-free might use

```
use after 'free' here; memory was freed at (2)
```

At this point we can emit the diagnostic.

26.1.6 Limitations

- Only for C so far
- The implementation of call summaries is currently very simplistic.
- Lack of function pointer analysis
- The constraint-handling code assumes reflexivity in some places (that values are equal to themselves), which is not the case for NaN. As a simple workaround, constraints on floating-point values are currently ignored.
- There are various other limitations in the region model (grep for `TODO/xfail` in the testsuite).
- The `constraint_manager`'s implementation of transitivity is currently too expensive to enable by default and so must be manually enabled via `-fanalyzer-transitivity`).
- The checkers are currently hardcoded and don't allow for user extensibility (e.g. adding allocate/release pairs).
- Although the analyzer's test suite has a proof-of-concept test case for LTO, LTO support hasn't had extensive testing. There are various lang-specific things in the analyzer

that assume C rather than LTO. For example, SSA names are printed to the user in “raw” form, rather than printing the underlying variable name.

26.2 Debugging the Analyzer

When debugging the analyzer I normally use all of these options together:

```
./xgcc -B. \
-S \
-fanalyzer \
OTHER_GCC_ARGS \
-wrapper gdb,--args \
-fdump-analyzer-stderr \
-fdump-ipa-analyzer=stderr
```

where:

- `./xgcc -B.` is the usual way to invoke a self-built GCC from within the `BUILDDIR/gcc` subdirectory.
- `-S` so that the driver (`./xgcc`) invokes `cc1`, but doesn't bother running the assembler or linker (since the analyzer runs inside `cc1`).
- `-fanalyzer` enables the analyzer, obviously.
- `-wrapper gdb,--args` invokes `cc1` under the debugger so that I can debug `cc1` and set breakpoints and step through things.
- `-fdump-analyzer-stderr` so that the logging interface is enabled and goes to `stderr`, which often gives valuable context into what's happening when stepping through the analyzer
- `-fdump-ipa-analyzer=stderr` which dumps the GIMPLE IR seen by the analyzer pass to `stderr`

Other useful options:

- `-fdump-analyzer-supergraph` which dumps `SRC.supergraph.N.KIND.dot` GraphViz files that I can look at (with `python-xdot`)
- `-fdump-analyzer-exploded-graph` which dumps a `SRC.eg.dot` GraphViz file
- `-fdump-analyzer-exploded-nodes-2` which dumps a `SRC.eg.txt` file containing the full `exploded_graph`.
- `-fdiagnostics-add-output=experimental-html:show-state-diagrams=yes` which writes out the diagnostics in HTML form, and generates SVG state diagrams visualizing the state of memory at each event (inspired by the “ddd” debugger). These can be seen by pressing ‘j’ and ‘k’ to single-step forward and backward through events. Note that these SVG diagrams are created from an intermediate SARIF directed graph representation generated from `program_state` objects. The SARIF representation can be easier to read - for example, rather than storing the contents of memory via byte offsets, it uses fields for structs and element indexes for arrays, recursively. However it is a different representation, and thus bugs could be hidden by this transformation. Generating the SVG diagrams requires an invocation of “dot” per event, so it noticeably slows down diagnostic emission, hence the opt-in command-line flag. The SARIF and “dot” representations can be seen by `--analyzer_dump_xml` and `--analyzer_dump_dot` below (writing them to `stderr`), or by adding `show-state-diagrams-sarif=yes`

and `show-state-diagrams-dot-src=yes` to the html sink, which shows them within the generated HTML next to the generated SVG.

Assuming that you have the python support scripts for gdb installed (which you should do, it makes debugging GCC much easier), you can use:

```
(gdb) break-on-saved-diagnostic
```

to put a breakpoint at the place where a diagnostic is saved during `exploded_graph` exploration, to see where a particular diagnostic is being saved, and:

```
(gdb) break-on-diagnostic
```

to put a breakpoint at the place where diagnostics are actually emitted.

26.2.1 Special Functions for Debugging the Analyzer

The analyzer recognizes various special functions by name, for use in debugging the analyzer, and for use in DejaGnu tests.

The declarations of these functions can be seen in the testsuite in `analyzer-decls.h`. None of these functions are actually implemented in terms of code, merely as `known_function` subclasses (in `gcc/analyzer/kf-analyzer.cc`).

`__analyzer_break`

Add:

```
__analyzer_break ();
```

to the source being analyzed to trigger a breakpoint in the analyzer when that source is reached. By putting a series of these in the source, it's much easier to effectively step through the program state as it's analyzed.

`__analyzer_describe`

The analyzer handles:

```
__analyzer_describe (0, expr);
```

by emitting a warning describing the 2nd argument (which can be of any type), at a verbosity level given by the 1st argument. This is for use when debugging, and may be of use in DejaGnu tests.

`__analyzer_dump`

```
__analyzer_dump ();
```

will dump the copious information about the analyzer's state each time it reaches the call in its traversal of the source.

`__analyzer_dump_capacity`

```
extern void __analyzer_dump_capacity (const void *ptr);
```

will emit a warning describing the capacity of the base region of the region pointed to by the 1st argument.

`__analyzer_dump_dot`

```
__analyzer_dump_dot ();
```

will dump GraphViz `.dot` source to stderr reaches the call in its traversal of the source. This `.dot` source implements a diagram describing the analyzer's state.

`__analyzer_dump_escaped`

```
extern void __analyzer_dump_escaped (void);
```

will emit a warning giving the number of decls that have escaped on this analysis path, followed by a comma-separated list of their names, in alphabetical order.

__analyzer_dump_path

```
__analyzer_dump_path ();
```

will emit a placeholder “note” diagnostic with a path to that call site, if the analyzer finds a feasible path to it. This can be useful for writing DejaGnu tests for constraint-tracking and feasibility checking.

__analyzer_dump_exploded_nodes

For every callsite to `__analyzer_dump_exploded_nodes` the analyzer will emit a warning after it finished the analysis containing information on all of the exploded nodes at that program point.

```
__analyzer_dump_exploded_nodes (0);
```

will output the number of “processed” nodes, and the IDs of both “processed” and “merger” nodes, such as:

```
warning: 2 processed enodes: [EN: 56, EN: 58] merger(s): [EN: 54-55, EN: 57, EN: 59]■
```

With a non-zero argument

```
__analyzer_dump_exploded_nodes (1);
```

it will also dump all of the states within the “processed” nodes.

__analyzer_dump_named_constant

When the analyzer sees a call to `__analyzer_dump_named_constant` it will emit a warning describing what is known about the value of a given named constant, for parts of the analyzer that interact with target headers.

For example:

```
__analyzer_dump_named_constant ("O_RDONLY");
```

might lead to the analyzer emitting the warning:

```
warning: named constant 'O_RDONLY' has value '1'
```

__analyzer_dump_region_model

```
__analyzer_dump_region_model ();
```

will dump the region_model’s state to stderr.

__analyzer_dump_state

```
__analyzer_dump_state ("malloc", ptr);
```

will emit a warning describing the state of the 2nd argument (which can be of any type) with respect to the state machine with a name matching the 1st argument (which must be a string literal). This is for use when debugging, and may be of use in DejaGnu tests.

__analyzer_dump_sarif

```
__analyzer_dump_sarif ();
```

will dump the copious information about the analyzer’s state each time it reaches the call in its traversal of the source.

__analyzer_eval

```
__analyzer_eval (expr);
```

will emit a warning with text "TRUE", "FALSE" or "UNKNOWN" based on the truthfulness of the argument. This is useful for writing DejaGnu tests.

__analyzer_get_unknown_ptr

```
__analyzer_get_unknown_ptr ();
```

will obtain an unknown void *.

```
__analyzer_get_strlen
    __analyzer_get_strlen (buf);
```

will emit a warning if PTR doesn't point to a null-terminated string. TODO: eventually get the strlen of the buffer (without the optimizer touching it).

26.2.2 Other Debugging Techniques

To compare two different exploded graphs, try `-fdump-analyzer-exploded-nodes-2 -fdump-noaddr`. This will dump a `SRC.eg.txt` file containing the full `exploded_graph`. I use `diff -u50 -p` to compare two different such files (e.g. before and after a patch) to find the first place where the two graphs diverge. The option `-fdump-noaddr` will suppress printing pointers within the dumps (which would otherwise hide the real differences with irrelevant churn).

The option `-fdump-analyzer-json` will dump both the supergraph and the exploded graph in compressed JSON form.

One approach when tracking down where a particular bogus state is introduced into the `exploded_graph` is to add custom code to `program_state::validate`.

The debug function `region::is_named_decl_p` can be used when debugging, such as for assertions and conditional breakpoints. For example, when tracking down a bug in handling a decl called `yy_buffer_stack`, I temporarily added a:

```
gcc_assert (!m_base_region->is_named_decl_p ("yy_buffer_stack"));
```

to `binding_cluster::mark_as_escaped` to trap a point where `yy_buffer_stack` was mistakenly being treated as having escaped.

27 User Experience Guidelines

To borrow a slogan from Elm (<https://elm-lang.org/news/compiler-as-assistants>),

Compilers should be assistants, not adversaries. A compiler should not just detect bugs, it should then help you understand why there is a bug. It should not berate you in a robot voice, it should give you specific hints that help you write better code. Ultimately, a compiler should make programming faster and more fun!

—Evan Czaplicki

This chapter provides guidelines on how to implement diagnostics and command-line options in ways that we hope achieve the above ideal.

27.1 Guidelines for Diagnostics

27.1.1 Talk in terms of the user’s code

Diagnostics should be worded in terms of the user’s source code, and the source language, rather than GCC’s own implementation details.

27.1.2 Diagnostics are actionable

A good diagnostic is *actionable*: it should assist the user in taking action.

Consider what an end user will want to do when encountering a diagnostic.

Given an error, an end user will think: “How do I fix this?”

Given a warning, an end user will think:

- “Is this a real problem?”
- “Do I care?”
- if they decide it’s genuine: “How do I fix this?”

A good diagnostic provides pertinent information to allow the user to easily answer the above questions.

27.1.3 The user’s attention is important

A perfect compiler would issue a warning on every aspect of the user’s source code that ought to be fixed, and issue no other warnings. Naturally, this ideal is impossible to achieve.

Warnings should have a good *signal-to-noise ratio*: we should have few *false positives* (falsely issuing a warning when no warning is warranted) and few *false negatives* (failing to issue a warning when one *is* justified).

Note that a false positive can mean, in practice, a warning that the user doesn’t agree with. Ideally a diagnostic should contain enough information to allow the user to make an informed choice about whether they should care (and how to fix it), but a balance must be drawn against overloading the user with irrelevant data.

27.1.4 Sometimes the user didn't write the code

GCC is typically used in two different ways:

- Semi-interactive usage: GCC is used as a development tool when the user is writing code, as the “compile” part of the “edit-compile-debug” cycle. The user is actively hacking on the code themselves (perhaps a project they wrote, or someone else's), where they just made a change to the code and want to see what happens, and to be warned about mistakes.
- Batch rebuilds: where the user is recompiling one or more existing packages, and GCC is a detail that's being invoked by various build scripts. Examples include a user trying to bring up an operating system consisting of hundreds of packages on a new CPU architecture, where the packages were written by many different people, or simply rebuilding packages after a dependency changed, where the user is hoping “nothing breaks”, since they are unfamiliar with the code.

Keep both of these styles of usage in mind when implementing diagnostics.

27.1.5 Precision of Wording

Provide the user with details that allow them to identify what the problem is. For example, the vaguely-worded message:

```
demo.c:1:1: warning: 'noinline' attribute ignored [-Wattributes]
  1 | int foo __attribute__((noinline));
    | ~~~
```

doesn't tell the user why the attribute was ignored, or what kind of entity the compiler thought the attribute was being applied to (the source location for the diagnostic is also poor; see [discussion of `input_location`], page 788). A better message would be:

```
demo.c:1:24: warning: attribute 'noinline' on variable 'foo' was
ignored [-Wattributes]
  1 | int foo __attribute__((noinline));
    | ~~~ ~~~~~
demo.c:1:24: note: attribute 'noinline' is only applicable to functions
```

which spells out the missing information (and fixes the location information, as discussed below).

The above example uses a note to avoid a combinatorial explosion of possible messages.

27.1.6 Try the diagnostic on real-world code

It's worth testing a new warning on many instances of real-world code, written by different people, and seeing what it complains about, and what it doesn't complain about.

This may suggest heuristics that silence common false positives.

It may also suggest ways to improve the precision of the message.

27.1.7 Make mismatches clear

Many diagnostics relate to a mismatch between two different places in the user's source code. Examples include:

- a type mismatch, where the type at a usage site does not match the type at a declaration
- the argument count at a call site does not match the parameter count at the declaration

- something is erroneously duplicated (e.g. an error, due to breaking a uniqueness requirement, or a warning, if it's suggestive of a bug)
- an “opened” syntactic construct (such as an open-parenthesis) is not closed

In each case, the diagnostic should indicate **both** pertinent locations (so that the user can easily see the problem and how to fix it).

The standard way to do this is with a note (via `inform`). For example:

```
auto_diagnostic_group d;
if (warning_at (loc, OPT_Wduplicated_cond,
               "duplicated %<if%> condition"))
    inform (EXPR_LOCATION (t), "previously used here");
```

which leads to:

```
demo.c: In function 'test':
demo.c:5:17: warning: duplicated 'if' condition [-Wduplicated-cond]
   5 |     else if (flag > 3)
     |             ~~~~~~
demo.c:3:12: note: previously used here
   3 |     if (flag > 3)
     |     ~~~~~~
```

The `inform` call should be guarded by the return value from the `warning_at` call so that the note isn't emitted when the warning is suppressed.

For cases involving punctuation where the locations might be near each other, they can be conditionally consolidated via `gcc_rich_location::add_location_if_nearby`:

```
auto_diagnostic_group d;
gcc_rich_location richloc (primary_loc);
bool added_secondary = richloc.add_location_if_nearby (secondary_loc);
error_at (&richloc, "main message");
if (!added_secondary)
    inform (secondary_loc, "message for secondary");
```

This will emit either one diagnostic with two locations:

```
demo.c:42:10: error: main message
      (foo)
      ~ ^
```

or two diagnostics:

```
demo.c:42:4: error: main message
      foo)
      ^
demo.c:40:2: note: message for secondary
      (
      ^
```

27.1.8 Location Information

GCC's `location_t` type can support both ordinary locations, and locations relating to a macro expansion.

As of GCC 6, ordinary locations changed from supporting just a point in the user's source code to supporting three points: the *caret* location, plus a start and a finish:

```
a = foo && bar;
~~~~~
|   |   |
|   |   finish
```

```

|   caret
start

```

Tokens coming out of libcpp have locations of the form `caret == start`, such as for `foo` here:

```

a = foo && bar;
  ~~~
  | |
  | finish
  caret == start

```

Compound expressions should be reported using the location of the expression as a whole, rather than just of one token within it.

For example, in `-Wformat`, rather than underlining just the first token of a bad argument:

```

printf("hello %i %s", (long)0, "world");
      ~~~
      %li

```

the whole of the expression should be underlined, so that the user can easily identify what is being referred to:

```

printf("hello %i %s", (long)0, "world");
      ~~~~~
      %li

```

Avoid using the `input_location` global, and the diagnostic functions that implicitly use it—use `error_at` and `warning_at` rather than `error` and `warning`, and provide the most appropriate `location_t` value available at that phase of the compilation. It's possible to supply secondary `location_t` values via `rich_location`.

For example, in the example of imprecise wording above, generating the diagnostic using `warning`:

```

// BAD: implicitly uses input_location
warning (OPT_Wattributes, "%qE attribute ignored", name);

```

leads to:

```

// BAD: uses input_location
demo.c:1:1: warning: 'noinline' attribute ignored [-Wattributes]
1 | int foo __attribute__((noinline));
  | ~~~

```

which thus happened to use the location of the `int` token, rather than that of the attribute. Using `warning_at` with the location of the attribute, providing the location of the declaration in question as a secondary location, and adding a note:

```

auto_diagnostic_group d;
gcc_rich_location richloc (attrib_loc);
richloc.add_range (decl_loc);
if (warning_at (OPT_Wattributes, &richloc,
               "attribute %qE on variable %qE was ignored", name))
    inform (attrib_loc, "attribute %qE is only applicable to functions");

```

would lead to:

```

// OK: use location of attribute, with a secondary location
demo.c:1:24: warning: attribute 'noinline' on variable 'foo' was
ignored [-Wattributes]
1 | int foo __attribute__((noinline));
  | ~~~~~
demo.c:1:24: note: attribute 'noinline' is only applicable to functions

```

27.1.9 Coding Conventions

See the diagnostics section (<https://gcc.gnu.org/codingconventions.html#Diagnostics>) of the GCC coding conventions.

In the C++ front end, when comparing two types in a message, use ‘%H’ and ‘%I’ rather than ‘%T’, as this allows the diagnostics subsystem to highlight differences between template-based types. For example, rather than using ‘%qT’:

```
// BAD: a pair of %qT used in C++ front end for type comparison
error_at (loc, "could not convert %qE from %qT to %qT", expr,
          TREE_TYPE (expr), type);
```

which could lead to:

```
error: could not convert 'map<int, double>()' from 'map<int,double>'
      to 'map<int,int>'
```

using ‘%H’ and ‘%I’ (via ‘%qH’ and ‘%qI’):

```
// OK: compare types in C++ front end via %qH and %qI
error_at (loc, "could not convert %qE from %qH to %qI", expr,
          TREE_TYPE (expr), type);
```

allows the above output to be simplified to:

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

where the `double` and `int` are colorized to highlight them.

27.1.10 Group logically-related diagnostics

Use `auto_diagnostic_group` when issuing multiple related diagnostics (seen in various examples on this page). This informs the diagnostic subsystem that all diagnostics issued within the lifetime of the `auto_diagnostic_group` are related. For example, `-fdiagnostics-add-output=sarif` will treat the first diagnostic emitted within the group as a top-level diagnostic, and all subsequent diagnostics within the group as its children. Also, if a warning in the group is inhibited at nesting depth `D`, all subsequent notes at that depth or deeper will be inhibited as well, until an error or another warning is emitted, the depth decreases below `D`, or the group is popped.

27.1.11 Quoting

Text should be quoted by either using the ‘q’ modifier in a directive such as ‘%qE’, or by enclosing the quoted text in a pair of ‘%<’ and ‘%>’ directives, and never by using explicit quote characters. The directives handle the appropriate quote characters for each language and apply the correct color or highlighting.

The following elements should be quoted in GCC diagnostics:

- Language keywords.
- Tokens.
- Boolean, numerical, character, and string constants that appear in the source code.
- Identifiers, including function, macro, type, and variable names.

Other elements such as numbers that do not refer to numeric constants that appear in the source code should not be quoted. For example, in the message:

```
argument %d of %qE must be a pointer type
```

since the argument number does not refer to a numerical constant in the source code it should not be quoted.

27.1.12 Use color consistently when highlighting mismatches

As of GCC 15, the diagnostics subsystem has a concept of “highlight colors”. These should be used to consistently colorize both the text within diagnostic messages and underlined ranges of quoted source when highlighting mismatches, for all messages with a logically-related group of diagnostics.

See `diagnostic-highlight-colors.h` for symbolic names for color codes, covering e.g.

- `highlight_colors::expected` versus `highlight_colors::actual`
- `highlight_colors::lhs` versus `highlight_colors::rhs`

For example, given:

```
error: invalid operands to binary + (have 'S' {aka 'struct s'} and 'T' {aka 'struct t'})
  return callee_4a () + callee_4b ();
      ~~~~~~ ^ ~~~~~~
      |           |
      |           T {aka struct t}
      S {aka struct s}
```

- the text “S {aka struct s}” in the message and the left-hand label in the quoted source should be colorized as `highlight_colors::lhs` (which equates to the color name `highlight-a`)
- the text “T {aka struct t}” in the message and the right-hand label in the quoted source should be colorized as `highlight_colors::rhs` (which equates to the color name `highlight-b`)

Doing so ought to make it easier for the user to understand what the diagnostic is telling them.

When issuing followup `note` diagnostics, all diagnostics within the group should use a consistent scheme to highlight the mismatching elements, so that color contrasts the differences. For example, given:

```
warning: format '%i' expects argument of type 'int', but argument 2 has type 'const char *' [-Wformat=]
279 |   printf("hello " INT_FMT " world", msg);
    |           ~~~~~~ ~~~
    |           |
    |           const char *

note: format string is defined here
278 | #define INT_FMT "%i"
    |           ^^
    |           |
    |           int
    |           %s
```

- the text `%i` and `int` referring to the format string and the expected type due to it should be colorized as `highlight-a` both in the diagnostics message and in the range quoted in the `range`.
- the text `const char *` in the diagnostic message and in the quoted range should be colorized as `highlight-b`.

This can be implemented by using e.g. `highlight_colors::actual` and `highlight_colors::expected` when adding ranges to `rich_location` instances, and e.g. by using the `%e` format code for `pretty_printer` to use a `pp_element *`, and using appropriate member functions of `pp_element` to add colorization.

27.1.13 Spelling and Terminology

See the terminology and markup (<https://gcc.gnu.org/codingconventions.html#Spelling>) section of the GCC coding conventions.

27.1.14 Fix-it hints

GCC's diagnostic subsystem can emit *fix-it hints*: small suggested edits to the user's source code.

They are printed by default underneath the code in question. They can also be viewed via `-fdiagnostics-generate-patch` and `-fdiagnostics-parseable-fixits`. With the latter, an IDE ought to be able to offer to automatically apply the suggested fix.

Fix-it hints contain code fragments, and thus they should not be marked for translation.

Fix-it hints can be added to a diagnostic by using a `rich_location` rather than a `location_t` - the fix-it hints are added to the `rich_location` using one of the various `add_fixit` member functions of `rich_location`. They are documented with `rich_location` in `libcpp/line-map.h`. It's easiest to use the `gcc_rich_location` subclass of `rich_location` found in `gcc-rich-location.h`, as this implicitly supplies the `line_table` variable.

For example:

```
if (const char *suggestion = hint.suggestion ())
{
    gcc_rich_location richloc (location);
    richloc.add_fixit_replace (suggestion);
    error_at (&richloc,
              "%qE does not name a type; did you mean %qs?",
              id, suggestion);
}
```

which can lead to:

```
spellcheck-typenames.C:73:1: error: 'singled' does not name a type; did
you mean 'signed'?
73 | singled char ch;
   | ~~~~~~
   | signed
```

Non-trivial edits can be built up by adding multiple fix-it hints to one `rich_location`. It's best to express the edits in terms of the locations of individual tokens. Various handy functions for adding fix-it hints for idiomatic C and C++ can be seen in `gcc-rich-location.h`.

27.1.14.1 Fix-it hints should work

When implementing a fix-it hint, please verify that the suggested edit leads to fixed, compilable code. (Unfortunately, this currently must be done by hand using `-fdiagnostics-generate-patch`. It would be good to have an automated way of verifying that fix-it hints actually fix the code).

For example, a “gotcha” here is to forget to add a space when adding a missing reserved word. Consider a C++ fix-it hint that adds `typename` in front of a template declaration. A naive way to implement this might be:

```
gcc_rich_location richloc (loc);
// BAD: insertion is missing a trailing space
richloc.add_fixit_insert_before ("typename");
error_at (&richloc, "need %<typename%> before %<%T::%E%> because "
          "%qT is a dependent scope",
```

```
parser->scope, id, parser->scope);
```

When applied to the code, this might lead to:

```
T::type x;
```

being “corrected” to:

```
typenameT::type x;
```

In this case, the correct thing to do is to add a trailing space after `typename`:

```
gcc_rich_location richloc (loc);
// OK: note that here we have a trailing space
richloc.add_fixit_insert_before ("typename ");
error_at (&richloc, "need %<typename%> before %<%T::%E%> because "
            "%qT is a dependent scope",
            parser->scope, id, parser->scope);
```

leading to this corrected code:

```
typename T::type x;
```

27.1.14.2 Express deletion in terms of deletion, not replacement

It’s best to express deletion suggestions in terms of deletion fix-it hints, rather than replacement fix-it hints. For example, consider this:

```
auto_diagnostic_group d;
gcc_rich_location richloc (location_of (retval));
tree name = DECL_NAME (arg);
richloc.add_fixit_replace (IDENTIFIER_POINTER (name));
warning_at (&richloc, OPT_Wredundant_move,
            "redundant move in return statement");
```

which is intended to e.g. replace a `std::move` with the underlying value:

```
return std::move (retval);
~~~~~
retval
```

where the change has been expressed as replacement, replacing with the name of the declaration. This works for simple cases, but consider this case:

```
#ifdef SOME_CONFIG_FLAG
# define CONFIGURY_GLOBAL global_a
#else
# define CONFIGURY_GLOBAL global_b
#endif

int fn ()
{
    return std::move (CONFIGURY_GLOBAL /* some comment */);
}
```

The above implementation erroneously strips out the macro and the comment in the fix-it hint:

```
return std::move (CONFIGURY_GLOBAL /* some comment */);
~~~~~
global_a
```

and thus this resulting code:

```
return global_a;
```

It’s better to do deletions in terms of deletions; deleting the `std::move (` and the trailing close-paren, leading to this:

```
return std::move (CONFIGURY_GLOBAL /* some comment */);
```

```
~~~~~  
CONFIGURY_GLOBAL /* some comment */
```

and thus this result:

```
return CONFIGURY_GLOBAL /* some comment */;
```

Unfortunately, the pertinent `location_t` values are not always available.

27.1.14.3 Multiple suggestions

In the rare cases where you need to suggest more than one mutually exclusive solution to a problem, this can be done by emitting multiple notes and calling `rich_location::fixits_cannot_be_auto_applied` on each note's `rich_location`. If this is called, then the fix-it hints in the `rich_location` will be printed, but will not be added to generated patches.

27.2 Guidelines for Options

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 Iverson 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 Nossun 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 CACAO integration, `fdlibm` updates.
- Gael Thomas for `VMClassLoader` boot packages support suggestions.

- Andreas Tobler for Darwin and Solaris testing and fixing, `Qt4` support for Darwin / 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.

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.

F

`fltrans` 764
`fltrans-output-list` 764
`fresolution` 764

`fwpa` 764

M

`msoft-float` 9

Concept Index

!

‘!’ in constraint 390

#

‘#’ in constraint 391

in template 376

#pragma 708

\$

‘\$’ in constraint 390

%

‘%’ in constraint 391

% in GTY option 738

‘%’ in template 375

&

‘&’ in constraint 391

(

(gimple_stmt_iterator 765

(nil) 284

*

‘*’ in constraint 391

* in template 377

*gimple_build_asm_vec 245

*gimple_build_assign 246

*gimple_build_bind 247

*gimple_build_call 248

*gimple_build_call_from_tree 248

*gimple_build_call_vec 248

*gimple_build_catch 250

*gimple_build_cond 250

*gimple_build_cond_from_tree 250

*gimple_build_debug_bind 251

*gimple_build_eh_filter 253

*gimple_build_goto 254

*gimple_build_label 253

*gimple_build_omp_atomic_load 254

*gimple_build_omp_atomic_store 254

*gimple_build_omp_continue 255

*gimple_build_omp_critical 255

*gimple_build_omp_for 256

*gimple_build_omp_parallel 257

*gimple_build_omp_sections 259

*gimple_build_omp_single 260

*gimple_build_resx 261

*gimple_build_return 261

*gimple_build_switch 261

*gimple_build_try 262

+

‘+’ in constraint 390

—

‘_’ in constraint 392

-fsection-anchors 289, 627

/

‘/c’ in RTL dump 293

‘/f’ in RTL dump 293

‘/i’ in RTL dump 294

‘/j’ in RTL dump 294

‘/s’ in RTL dump 294

‘/u’ in RTL dump 294

‘/v’ in RTL dump 295

:

‘:’ in constraint 387

<

‘<’ in constraint 385

=

‘=’ in constraint 390

>

‘>’ in constraint 386

?

‘?’ in constraint 390

^

‘^’ in constraint 390

__absvdi2	7	__bid_divdd3	15
__absvsi2	7	__bid_divsd3	15
__addda3	21	__bid_divtd3	15
__adddf3	9	__bid_eqdd2	19
__adddq3	21	__bid_eqsd2	19
__addha3	21	__bid_eqtd2	19
__addhq3	21	__bid_extendddtd2	16
__addqq3	21	__bid_extendddtfd	17
__addsa3	21	__bid_extendddxf	16
__addsf3	9	__bid_extenddfdd	17
__addsq3	21	__bid_extenddfdd	16
__addta3	21	__bid_extendsddd2	15
__addtf3	9	__bid_extendsddf	16
__adduda3	21	__bid_extendsdtd2	15
__addudq3	21	__bid_extendsdtfd	16
__adduha3	21	__bid_extendsdxf	16
__adduhq3	21	__bid_extendsfdd	16
__adduqq3	21	__bid_extendsfsd	17
__addusa3	21	__bid_extendsftd	16
__addusq3	21	__bid_extendtftd	17
__adduta3	21	__bid_extendxftd	16
__addvdi3	7	__bid_fixddbittint	18
__addvsi3	7	__bid_fixdddi	17
__addxf3	9	__bid_fixddsi	17
__ashlda3	27	__bid_fixsdbittint	18
__ashldi3	5	__bid_fixsddi	17
__ashldq3	27	__bid_fixsdsi	17
__ashlha3	27	__bid_fixtdbittint	18
__ashlhq3	27	__bid_fixtddi	17
__ashlqq3	27	__bid_fixtdsi	17
__ashlsa3	27	__bid_fixunsdddi	17
__ashlsi3	5	__bid_fixunsddsi	17
__ashlsq3	27	__bid_fixunssddi	17
__ashlta3	27	__bid_fixunssdsi	17
__ashlti3	5	__bid_fixunstddi	18
__ashluda3	27	__bid_fixunstdsi	17
__ashludq3	27	__bid_floatbittintdd	19
__ashluha3	27	__bid_floatbittintsd	19
__ashluhq3	27	__bid_floatbittinttd	19
__ashluqq3	27	__bid_floatdidd	18
__ashlusa3	27	__bid_floatdisd	18
__ashlusq3	27	__bid_floatditd	18
__ashluta3	27	__bid_floatsidd	18
__ashrda3	28	__bid_floatsisd	18
__ashrdi3	5	__bid_floatsitd	18
__ashrdq3	27	__bid_floatunsdidd	18
__ashrha3	27	__bid_floatunsdisd	18
__ashrhq3	27	__bid_floatunsditd	18
__ashrqq3	27	__bid_floatunssidd	18
__ashrsa3	28	__bid_floatunssisd	18
__ashrsi3	5	__bid_floatunssitd	18
__ashrsq3	27	__bid_gedd2	20
__ashrta3	28	__bid_gesd2	20
__ashrti3	5	__bid_getd2	20
__bid_adddd3	14	__bid_gtd2	20
__bid_adddsd3	14	__bid_gtsd2	20
__bid_addtd3	14	__bid_gtt2	20
		__bid_ledd2	20

__bid_lesd2.....	20	__cmpudq2.....	29
__bid_letd2.....	20	__cmpuha2.....	29
__bid_ltdd2.....	20	__cmpuhq2.....	29
__bid_ltsd2.....	20	__cmpuqq2.....	29
__bid_lttd2.....	20	__cmpusa2.....	29
__bid_muldd3.....	15	__cmpusq2.....	29
__bid_mulsd3.....	15	__cmputa2.....	29
__bid_multd3.....	15	__CTOR_LIST_.....	669
__bid_nedd2.....	19	__ctzdi2.....	7
__bid_negdd2.....	15	__ctzsi2.....	7
__bid_negsd2.....	15	__ctzti2.....	7
__bid_negtd2.....	15	__divda3.....	25
__bid_nesd2.....	19	__divdc3.....	14
__bid_netd2.....	19	__divdf3.....	9
__bid_subdd3.....	14	__divdi3.....	5
__bid_subsd3.....	14	__divdq3.....	25
__bid_subtd3.....	15	__divha3.....	25
__bid_truncdddf.....	17	__divhq3.....	24
__bid_truncddsd2.....	16	__divmodbitint4.....	8
__bid_truncddsf.....	16	__divqq3.....	24
__bid_truncdfsd.....	16	__divsa3.....	25
__bid_truncsdsf.....	17	__divsc3.....	14
__bid_trunctddd2.....	16	__divsf3.....	9
__bid_trunctddf.....	16	__divsi3.....	5
__bid_trunctdsd2.....	16	__divsq3.....	25
__bid_trunctdsf.....	16	__divta3.....	25
__bid_trunctdtf.....	17	__divtc3.....	14
__bid_trunctdx.....	16	__divtf3.....	9
__bid_trunctfdd.....	16	__divti3.....	5
__bid_trunctfsd.....	16	__divxc3.....	14
__bid_truncxfdd.....	16	__divxf3.....	9
__bid_truncxfsd.....	16	__dpd_adddd3.....	14
__bid_unorddd2.....	19	__dpd_addsd3.....	14
__bid_unordsd2.....	19	__dpd_addtd3.....	14
__bid_unordtd2.....	19	__dpd_divdd3.....	15
__bswapdi2.....	8	__dpd_divsd3.....	15
__bswapsi2.....	8	__dpd_divtd3.....	15
__builtin_classify_type.....	609	__dpd_eqdd2.....	19
__builtin_next_arg.....	609	__dpd_eqsd2.....	19
__builtin_saverregs.....	608	__dpd_eqtd2.....	19
__clear_cache.....	60	__dpd_extendddtd2.....	15
__clzdi2.....	7	__dpd_extenddddtf.....	16
__clzsi2.....	7	__dpd_extenddddx.....	16
__clzti2.....	7	__dpd_extendddfd.....	17
__cmpda2.....	29	__dpd_extenddftd.....	16
__cmpdf2.....	12	__dpd_extendsddd2.....	15
__cmpdi2.....	6	__dpd_extendsddf.....	16
__cmpdq2.....	29	__dpd_extendsdtd2.....	15
__cmpha2.....	29	__dpd_extendsdtf.....	16
__cmphq2.....	29	__dpd_extendsdx.....	16
__cmpqq2.....	29	__dpd_extendsfdd.....	16
__cmpsa2.....	29	__dpd_extendsfsd.....	17
__cmpsf2.....	12	__dpd_extendsftd.....	16
__cmpsq2.....	29	__dpd_extendtftd.....	17
__cmpta2.....	29	__dpd_extendxftd.....	16
__cmptf2.....	12	__dpd_fixdddi.....	17
__cmpti2.....	6	__dpd_fixddsi.....	17
__cmpuda2.....	29	__dpd_fixsddi.....	17

__dpd_fixsdsi	17	__dpd_truncxfdd	16
__dpd_fixtddi	17	__dpd_truncxfsd	16
__dpd_fixtdsi	17	__dpd_unorddd2	19
__dpd_fixunsdddi	17	__dpd_unordsd2	19
__dpd_fixunsddsi	17	__dpd_unordtd2	19
__dpd_fixunssddi	17	__DTOR_LIST__	669
__dpd_fixunssdsi	17	__eqdf2	12
__dpd_fixunstddi	17	__eqsf2	12
__dpd_fixunstdsi	17	__eqtf2	12
__dpd_floatdidd	18	__extenddftf2	10
__dpd_floatdisd	18	__extenddfxf2	10
__dpd_floatditd	18	__extendsfdf2	10
__dpd_floatsidd	18	__extendsftf2	10
__dpd_floatsisd	18	__extendsfxf2	10
__dpd_floatsitd	18	__ffsdi2	7
__dpd_floatunsdidd	18	__ffsti2	7
__dpd_floatunsdisd	18	__fixdfbitint	11
__dpd_floatunssditd	18	__fixdfdi	10
__dpd_floatunssidd	18	__fixdfsi	10
__dpd_floatunssisd	18	__fixdfti	10
__dpd_floatunssitd	18	__fixsfbtint	11
__dpd_gedd2	20	__fixsfdi	10
__dpd_gesd2	20	__fixsfsi	10
__dpd_getd2	20	__fixsfti	10
__dpd_gtdd2	20	__fixtfbtint	12
__dpd_gtsd2	20	__fixtfdi	10
__dpd_gttdd2	20	__fixtfsi	10
__dpd_ledd2	20	__fixtfti	10
__dpd_lesd2	20	__fixunsdfdi	10
__dpd_letd2	20	__fixunsdfsi	10
__dpd_ltdd2	20	__fixunsdfti	11
__dpd_ltsd2	20	__fixunssfdi	10
__dpd_lttdd2	20	__fixunssfsi	10
__dpd_muldd3	15	__fixunssfti	11
__dpd_mulsd3	15	__fixunstfdi	10
__dpd_multdd3	15	__fixunstfsi	10
__dpd_nedd2	19	__fixunstfti	11
__dpd_negdd2	15	__fixunsxfdi	10
__dpd_negsd2	15	__fixunsxfsi	10
__dpd_negtd2	15	__fixunsxfti	11
__dpd_nesd2	19	__fixxfbitint	11
__dpd_netd2	19	__fixxfdi	10
__dpd_subdd3	14	__fixxfsi	10
__dpd_subsd3	14	__fixxfti	10
__dpd_subtd3	14	__floatbitintbf	12
__dpd_truncdddf	17	__floatbitintdf	12
__dpd_truncddsd2	16	__floatbitinthf	12
__dpd_truncddsf	16	__floatbitintsf	12
__dpd_truncdfsd	16	__floatbitinttf	12
__dpd_truncsdsf	17	__floatbitintxf	12
__dpd_trunctddd2	16	__floatdidf	11
__dpd_trunctddf	16	__floatdisf	11
__dpd_trunctdsd2	16	__floatditf	11
__dpd_trunctdsf	16	__floatdixf	11
__dpd_trunctdtf	17	__floatsidf	11
__dpd_trunctdxf	16	__floatsisf	11
__dpd_trunctfdd	16	__floatsitf	11
__dpd_trunctfsd	16	__floatsixf	11

__floattidf.....	11	__fractdiqq.....	40
__floattisf.....	11	__fractdisa.....	40
__floattitf.....	11	__fractdisq.....	40
__floattixf.....	11	__fractdita.....	40
__floatundidf.....	11	__fractdiuda.....	40
__floatundisf.....	11	__fractdiudq.....	40
__floatunditf.....	11	__fractdiuha.....	40
__floatundixf.....	11	__fractdiuhq.....	40
__floatunsidf.....	11	__fractdiuqq.....	40
__floatunsisf.....	11	__fractdiusa.....	40
__floatunsitf.....	11	__fractdiusq.....	40
__floatunsixf.....	11	__fractdiuta.....	40
__floatuntidf.....	11	__fractdqda.....	31
__floatuntisf.....	11	__fractdqdf.....	31
__floatuntitf.....	11	__fractdqdi.....	31
__floatuntixf.....	11	__fractdqha.....	31
__fractdadf.....	33	__fractdqhi.....	31
__fractdadi.....	33	__fractdqhq2.....	31
__fractdadq.....	32	__fractdqqi.....	31
__fractdaha2.....	32	__fractdqqq2.....	31
__fractdahi.....	33	__fractdqlsa.....	31
__fractdahq.....	32	__fractdqlsf.....	31
__fractdaqi.....	33	__fractdqlsi.....	31
__fractdaqq.....	32	__fractdqlsq2.....	31
__fractdasa2.....	32	__fractdqta.....	31
__fractdasf.....	33	__fractdqti.....	31
__fractdasi.....	33	__fractdquda.....	31
__fractdasq.....	32	__fractdqudq.....	31
__fractdata2.....	33	__fractdquha.....	31
__fractdati.....	33	__fractdquhq.....	31
__fractdauda.....	33	__fractdquqq.....	31
__fractdaudq.....	33	__fractdqusa.....	31
__fractdauha.....	33	__fractdqusq.....	31
__fractdauhq.....	33	__fractdquta.....	31
__fractdauqq.....	33	__fracthada2.....	32
__fractdausa.....	33	__fracthadf.....	32
__fractdausq.....	33	__fracthadi.....	32
__fractdauta.....	33	__fracthadq.....	31
__fractdfda.....	41	__fracthahi.....	32
__fractdfdq.....	41	__fracthahq.....	31
__fractdfha.....	41	__fracthaqi.....	32
__fractdfhq.....	41	__fracthaqq.....	31
__fractdfqq.....	41	__fracthasa2.....	31
__fractdfsa.....	41	__fracthasf.....	32
__fractdfsq.....	41	__fracthasi.....	32
__fractdfta.....	41	__fracthasq.....	31
__fractdfuda.....	41	__fracthata2.....	32
__fractdfudq.....	41	__fracthati.....	32
__fractdfuha.....	41	__fracthauda.....	32
__fractdfuhq.....	41	__fracthaudq.....	32
__fractdfuqq.....	41	__fracthauha.....	32
__fractdfusa.....	41	__fracthauhq.....	32
__fractdfusq.....	41	__fracthauqq.....	32
__fractdfuta.....	41	__fracthausa.....	32
__fractdida.....	40	__fracthausq.....	32
__fractdidq.....	40	__fracthauta.....	32
__fractdiha.....	40	__fracthida.....	39
__fractdihq.....	40	__fracthidq.....	39

__fracthiha.....	39	__fractqqhq2.....	29
__fracthihq.....	39	__fractqqqi.....	30
__fracthiqq.....	39	__fractqqsa.....	29
__fracthisa.....	39	__fractqqsf.....	30
__fracthisq.....	39	__fractqqsi.....	30
__fracthita.....	39	__fractqqsq2.....	29
__fracthiuda.....	40	__fractqqta.....	29
__fracthiudq.....	40	__fractqqti.....	30
__fracthiuha.....	40	__fractqquda.....	30
__fracthiuhq.....	39	__fractqqudq.....	30
__fracthiuqq.....	39	__fractqquha.....	30
__fracthiusa.....	40	__fractqquhq.....	29
__fracthiusq.....	39	__fractqquqq.....	29
__fracthiuta.....	40	__fractqqusa.....	30
__fracthqda.....	30	__fractqqusq.....	30
__fracthqdf.....	30	__fractqquta.....	30
__fracthqdi.....	30	__fractsada2.....	32
__fracthqdq2.....	30	__fractsadf.....	32
__fracthqha.....	30	__fractsadi.....	32
__fracthqhi.....	30	__fractsadq.....	32
__fracthqqi.....	30	__fractsaha2.....	32
__fracthqqq2.....	30	__fractsahi.....	32
__fracthqsa.....	30	__fractsahq.....	32
__fracthqsf.....	30	__fractsaqi.....	32
__fracthqsi.....	30	__fractsaqq.....	32
__fracthqsq2.....	30	__fractsasf.....	32
__fracthqta.....	30	__fractsasi.....	32
__fracthqti.....	30	__fractsasq.....	32
__fracthquda.....	30	__fractsata2.....	32
__fracthqudq.....	30	__fractsati.....	32
__fractquha.....	30	__fractsauda.....	32
__fractquhq.....	30	__fractsaudq.....	32
__fractquqq.....	30	__fractsauha.....	32
__fractqusa.....	30	__fractsauhq.....	32
__fractqusq.....	30	__fractsauqq.....	32
__fractquta.....	30	__fractsausa.....	32
__fractqida.....	39	__fractsausq.....	32
__fractqidq.....	39	__fractsauta.....	32
__fractqiha.....	39	__fractsfda.....	41
__fractqihq.....	39	__fractsfdq.....	41
__fractqiqq.....	39	__fractsfha.....	41
__fractqisa.....	39	__fractsfhq.....	41
__fractqisq.....	39	__fractsfqq.....	41
__fractqita.....	39	__fractsfsa.....	41
__fractqiuda.....	39	__fractsfsq.....	41
__fractqiudq.....	39	__fractsfta.....	41
__fractqiuha.....	39	__fractsfuda.....	41
__fractqiuhq.....	39	__fractsfudq.....	41
__fractqiuqq.....	39	__fractsfuha.....	41
__fractqiusa.....	39	__fractsfuhq.....	41
__fractqiusq.....	39	__fractsfuqq.....	41
__fractqiuta.....	39	__fractsfusa.....	41
__fractqqda.....	29	__fractsfusq.....	41
__fractqqdf.....	30	__fractsfuta.....	41
__fractqqdi.....	30	__fractsida.....	40
__fractqqdq2.....	29	__fractsidq.....	40
__fractqqha.....	29	__fractsiha.....	40
__fractqqhi.....	30	__fractsihq.....	40

__fractsiqq.....	40	__fracttiha.....	40
__fractsisa.....	40	__fracttihq.....	40
__fractsisq.....	40	__fracttiqq.....	40
__fractsitaa.....	40	__fracttisa.....	40
__fractsiuda.....	40	__fracttisq.....	40
__fractsiudq.....	40	__fractttita.....	40
__fractsiuha.....	40	__fractttiuda.....	41
__fractsiuhq.....	40	__fractttiudq.....	41
__fractsiuqq.....	40	__fractttiuha.....	41
__fractsiusa.....	40	__fractttiuhq.....	40
__fractsiusq.....	40	__fractttiuqq.....	40
__fractsiuta.....	40	__fractttiusa.....	41
__fractsqda.....	30	__fractttiusq.....	41
__fractsqdf.....	31	__fractttiuta.....	41
__fractsqdi.....	31	__fracttudada.....	38
__fractsqdq2.....	30	__fracttudadf.....	38
__fractsqha.....	30	__fracttudadi.....	38
__fractsqhi.....	31	__fracttudadq.....	38
__fractsqhq2.....	30	__fracttudaha.....	38
__fractsqqi.....	31	__fracttudahi.....	38
__fractsqqq2.....	30	__fracttudahq.....	37
__fractsqsa.....	30	__fracttudaqi.....	38
__fractsqsf.....	31	__fracttudaqq.....	37
__fractsqsi.....	31	__fracttudasa.....	38
__fractsqta.....	30	__fracttudasf.....	38
__fractsqti.....	31	__fracttudasq.....	37
__fractsquda.....	31	__fracttudata.....	38
__fractsqudq.....	31	__fracttudati.....	38
__fractsquha.....	31	__fracttudaudq.....	38
__fractsquhq.....	30	__fracttudauiha2.....	38
__fractsquqq.....	30	__fracttudauihq.....	38
__fractsqusa.....	31	__fracttudauiqq.....	38
__fractsqusq.....	30	__fracttudausa2.....	38
__fractsquta.....	31	__fracttudausq.....	38
__fracttada2.....	33	__fracttudauiha2.....	38
__fracttadf.....	33	__fracttudauiqq.....	38
__fracttadi.....	33	__fracttudauiqq.....	38
__fracttadq.....	33	__fracttudauiqq.....	38
__fracttaha2.....	33	__fracttudauiqq.....	38
__fracttahi.....	33	__fracttudauiqq.....	38
__fracttahq.....	33	__fracttudauiqq.....	38
__fracttaqi.....	33	__fracttudauiqq.....	38
__fracttaqq.....	33	__fracttudauiqq.....	38
__fracttasa2.....	33	__fracttudauiqq.....	38
__fracttasf.....	33	__fracttudauiqq.....	38
__fracttasi.....	33	__fracttudauiqq.....	38
__fracttasq.....	33	__fracttudauiqq.....	38
__fracttati.....	33	__fracttudauiqq.....	38
__fracttauda.....	33	__fracttudauiqq.....	38
__fracttaudq.....	33	__fracttudauiqq.....	38
__fracttauha.....	33	__fracttudauiqq.....	38
__fracttauuhq.....	33	__fracttudauiqq.....	38
__fracttauqq.....	33	__fracttudauiqq.....	38
__fracttausa.....	33	__fracttudauiqq.....	38
__fracttausq.....	33	__fracttudauiqq.....	38
__fracttauta.....	33	__fracttudauiqq.....	38
__fracttida.....	40	__fracttudauiqq.....	38
__fracttidq.....	40	__fracttudauiqq.....	38

__fractuhada.....	36	__fractunsdiudq.....	55
__fractuhadf.....	37	__fractunsdiuha.....	55
__fractuhadi.....	37	__fractunsdiuhq.....	55
__fractuhadq.....	36	__fractunsdiuqq.....	55
__fractuhaha.....	36	__fractunsdiusa.....	55
__fractuhahi.....	37	__fractunsdiusq.....	55
__fractuhahq.....	36	__fractunsdiuta.....	56
__fractuhaqi.....	37	__fractunsdqdi.....	52
__fractuhaqq.....	36	__fractunsdqhi.....	52
__fractuhasa.....	36	__fractunsdqqi.....	52
__fractuhasf.....	37	__fractunsdqsi.....	52
__fractuhasi.....	37	__fractunsdqti.....	52
__fractuhasq.....	36	__fractunshadi.....	52
__fractuhata.....	36	__fractunshahi.....	52
__fractuhati.....	37	__fractunshaqi.....	52
__fractuhauda2.....	37	__fractunshasi.....	52
__fractuhaudq.....	37	__fractunshati.....	52
__fractuhaulhq.....	36	__fractunshida.....	54
__fractuhaulqq.....	36	__fractunshidq.....	54
__fractuhaus2.....	37	__fractunshiha.....	54
__fractuhausq.....	36	__fractunshihq.....	54
__fractuhausta2.....	37	__fractunshiqq.....	54
__fractuhqda.....	34	__fractunshisa.....	54
__fractuhqdf.....	35	__fractunshisq.....	54
__fractuhqdi.....	35	__fractunshita.....	54
__fractuhqdq.....	34	__fractunshiuda.....	55
__fractuhqha.....	34	__fractunshiudq.....	54
__fractuhqhi.....	35	__fractunshiuha.....	55
__fractuhqhq.....	34	__fractunshiuhq.....	54
__fractuhqqi.....	35	__fractunshiuqq.....	54
__fractuhqqq.....	34	__fractunshiusa.....	55
__fractuhqsa.....	34	__fractunshiusq.....	54
__fractuhqsf.....	35	__fractunshiuta.....	55
__fractuhqsi.....	35	__fractunshqdi.....	51
__fractuhqsq.....	34	__fractunshqhi.....	51
__fractuhqta.....	34	__fractunshqqi.....	51
__fractuhqti.....	35	__fractunshqsi.....	51
__fractuhquda.....	35	__fractunshqti.....	51
__fractuhqudq2.....	34	__fractunsqida.....	54
__fractuhquha.....	34	__fractunsqidq.....	54
__fractuhquqq2.....	34	__fractunsqiha.....	54
__fractuhqusa.....	34	__fractunsqihq.....	54
__fractuhqusq2.....	34	__fractunsqiqq.....	54
__fractuhquta.....	35	__fractunsqisa.....	54
__fractunsdadi.....	52	__fractunsqisq.....	54
__fractunsdahi.....	52	__fractunsqita.....	54
__fractunsdaq.....	52	__fractunsqiuda.....	54
__fractunsdasi.....	52	__fractunsqiudq.....	54
__fractunsdati.....	52	__fractunsqiuha.....	54
__fractunsdida.....	55	__fractunsqiuhq.....	54
__fractunsdidq.....	55	__fractunsqiuqq.....	54
__fractunsdiha.....	55	__fractunsqiusa.....	54
__fractunsdihq.....	55	__fractunsqiusq.....	54
__fractunsdiqq.....	55	__fractunsqiuta.....	54
__fractunsdisa.....	55	__fractunsqqdi.....	51
__fractunsdisq.....	55	__fractunsqqhi.....	51
__fractunsdita.....	55	__fractunsqqqi.....	51
__fractunsdiuda.....	56	__fractunsqqsi.....	51

__fractunssqti	51	__fractunsuhadi	53
__fractunssadi	52	__fractunsuhahi	53
__fractunssahi	52	__fractunsuhaqi	53
__fractunssaqi	52	__fractunsuhasi	53
__fractunssasi	52	__fractunsuhati	53
__fractunssati	52	__fractunsuhqdi	52
__fractunssida	55	__fractunsuhqhi	52
__fractunssidq	55	__fractunsuhqqi	52
__fractunssiha	55	__fractunsuhqsi	52
__fractunssihq	55	__fractunsuhqti	52
__fractunssiqq	55	__fractunsuqqdi	52
__fractunssisa	55	__fractunsuqqhi	52
__fractunssisq	55	__fractunsuqqqi	52
__fractunssita	55	__fractunsuqqsi	52
__fractunssiuda	55	__fractunsuqqti	52
__fractunssiudq	55	__fractunsusadi	53
__fractunssiuha	55	__fractunsusahi	53
__fractunssiuhq	55	__fractunsusaqi	53
__fractunssiuqq	55	__fractunsusasi	53
__fractunssiusa	55	__fractunsusati	53
__fractunssiusq	55	__fractunsusqdi	53
__fractunssiuta	55	__fractunsusqhi	53
__fractunssqdi	52	__fractunsusqqi	53
__fractunssqhi	51	__fractunsusqsi	53
__fractunssqqi	51	__fractunsusqti	53
__fractunssqsi	52	__fractunsutadi	54
__fractunssqti	52	__fractunsutahi	54
__fractunstadi	52	__fractunsutaqi	54
__fractunstahi	52	__fractunsutasi	54
__fractunstaqi	52	__fractunsutati	54
__fractunstasi	52	__fractuqqda	34
__fractunstati	52	__fractuqqdf	34
__fractunstida	56	__fractuqqdi	34
__fractunstidq	56	__fractuqqdq	34
__fractunstiha	56	__fractuqqha	34
__fractunstihq	56	__fractuqqhi	34
__fractunstiqq	56	__fractuqqhq	34
__fractunstisa	56	__fractuqqqi	34
__fractunstisq	56	__fractuqqqq	33
__fractunstita	56	__fractuqqsa	34
__fractunstiuda	56	__fractuqqsf	34
__fractunstiudq	56	__fractuqqsi	34
__fractunstiuha	56	__fractuqqsq	34
__fractunstiuhq	56	__fractuqqta	34
__fractunstiuqq	56	__fractuqqti	34
__fractunstiusa	56	__fractuqquda	34
__fractunstiusq	56	__fractuqqudq2	34
__fractunstiuta	56	__fractuqquha	34
__fractunsudadi	53	__fractuqquhq2	34
__fractunsudahi	53	__fractuqqusa	34
__fractunsudaqi	53	__fractuqqusq2	34
__fractunsudasi	53	__fractuqquta	34
__fractunsudati	53	__fractusada	37
__fractunsudqdi	53	__fractusadf	37
__fractunsudqhi	53	__fractusadi	37
__fractunsudqqi	53	__fractusadq	37
__fractunsudqsi	53	__fractusaha	37
__fractunsudqti	53	__fractusahi	37

__fractusahq.....	37	__fractutausa2.....	39
__fractusaqi.....	37	__fractutausq.....	39
__fractusaqq.....	37	__gedf2.....	13
__fractusasa.....	37	__gesf2.....	13
__fractusasf.....	37	__getf2.....	13
__fractusasi.....	37	__gtdf2.....	13
__fractusasq.....	37	__gtsf2.....	13
__fractusata.....	37	__gttf2.....	13
__fractusati.....	37	__ledf2.....	13
__fractusauda2.....	37	__lesf2.....	13
__fractusaudq.....	37	__letf2.....	13
__fractusauha2.....	37	__lshrdi3.....	5
__fractusauhq.....	37	__lshrsi3.....	5
__fractusauqq.....	37	__lshrti3.....	5
__fractusausq.....	37	__lshruda3.....	28
__fractusauta2.....	37	__lshrudq3.....	28
__fractusqda.....	35	__lshruha3.....	28
__fractusqdf.....	35	__lshruhq3.....	28
__fractusqdi.....	35	__lshruqq3.....	28
__fractusqdq.....	35	__lshrusa3.....	28
__fractusqha.....	35	__lshrusq3.....	28
__fractusqhi.....	35	__lshruta3.....	28
__fractusqhq.....	35	__ltdf2.....	13
__fractusqqi.....	35	__ltsf2.....	13
__fractusqqq.....	35	__lttf2.....	13
__fractusqsa.....	35	__main.....	733
__fractusqsf.....	35	__moddi3.....	6
__fractusqsi.....	35	__modsi3.....	6
__fractusqsq.....	35	__modti3.....	6
__fractusqta.....	35	__morestack_current_segment.....	60
__fractusqti.....	35	__morestack_initial_sp.....	60
__fractusquda.....	35	__morestack_segments.....	60
__fractusqudq2.....	35	__mulbitint3.....	8
__fractusquha.....	35	__mulda3.....	24
__fractusquhq2.....	35	__muldc3.....	13
__fractusquqq2.....	35	__muldf3.....	9
__fractusqusa.....	35	__muldi3.....	6
__fractusquta.....	35	__muldq3.....	23
__fractutada.....	38	__mulha3.....	23
__fractutadf.....	39	__mulhq3.....	23
__fractutadi.....	39	__mulqq3.....	23
__fractutadq.....	38	__mulsa3.....	24
__fractutaha.....	38	__mulsc3.....	13
__fractutahi.....	39	__mulsf3.....	9
__fractutahq.....	38	__mulsi3.....	6
__fractutaqi.....	39	__mulsq3.....	23
__fractutaqq.....	38	__multa3.....	24
__fractutasa.....	38	__multc3.....	13
__fractutasf.....	39	__multf3.....	9
__fractutasi.....	39	__multi3.....	6
__fractutasq.....	38	__muluda3.....	24
__fractutata.....	38	__muludq3.....	23
__fractutati.....	39	__muluha3.....	24
__fractutauda2.....	39	__muluhq3.....	23
__fractutaudq.....	39	__muluqq3.....	23
__fractutauha2.....	39	__mulusa3.....	24
__fractutauhq.....	38	__mulusq3.....	23
__fractutauqq.....	38	__muluta3.....	24

__mulvdi3	7	__satfractdfha	51
__mulvsi3	7	__satfractdfhq	51
__mulxc3	14	__satfractdfqq	51
__mulxf3	9	__satfractdfs	51
__nedf2	13	__satfractdfsq	51
__negda2	26	__satfractdfta	51
__negdf2	9	__satfractdfuda	51
__negdi2	6	__satfractdfudq	51
__negdq2	26	__satfractdfuha	51
__negha2	26	__satfractdfuhq	51
__neghq2	26	__satfractdfuqq	51
__negqq2	26	__satfractdfusa	51
__negsa2	26	__satfractdfusq	51
__negsf2	9	__satfractdfuta	51
__negsq2	26	__satfractdida	50
__negta2	26	__satfractdidq	50
__negtf2	9	__satfractdiha	50
__negti2	6	__satfractdihq	50
__neguda2	26	__satfractdiqq	50
__negudq2	26	__satfractdisa	50
__neguha2	26	__satfractdisq	50
__neguhq2	26	__satfractdita	50
__neguqq2	26	__satfractdiuda	50
__negusa2	26	__satfractdiudq	50
__negusq2	26	__satfractdiuha	50
__neguta2	26	__satfractdiuhq	50
__negvdi2	7	__satfractdiuqq	50
__negvsi2	7	__satfractdiusa	50
__negxf2	9	__satfractdiusq	50
__nesf2	13	__satfractdiuta	50
__netf2	13	__satfractdqda	43
__paritydi2	8	__satfractdqha	43
__paritysi2	8	__satfractdqhq2	42
__parityti2	8	__satfractdqqq2	42
__popcountdi2	8	__satfractdqsa	43
__popcountsi2	8	__satfractdqsq2	43
__popcountti2	8	__satfractdqta	43
__powidf2	13	__satfractdquda	43
__powisf2	13	__satfractdqudq	43
__powitf2	13	__satfractdquha	43
__powixf2	13	__satfractdquhq	43
__satfractdadq	44	__satfractdquqq	43
__satfractdaha2	44	__satfractdqusa	43
__satfractdahq	44	__satfractdqusq	43
__satfractdaqq	44	__satfractdquta	43
__satfractdasa2	44	__satfracthada2	43
__satfractdasq	44	__satfracthadq	43
__satfractdata2	44	__satfracthahq	43
__satfractdauda	44	__satfracthaqq	43
__satfractdaudq	44	__satfracthasa2	43
__satfractdauha	44	__satfracthasq	43
__satfractdauhq	44	__satfracthata2	43
__satfractdauqq	44	__satfracthauda	43
__satfractdausa	44	__satfracthaudq	43
__satfractdausq	44	__satfracthauha	43
__satfractdauta	44	__satfracthauhq	43
__satfractdfda	51	__satfracthauqq	43
__satfractdfdq	51	__satfracthausa	43

__satfracthausq	43	__satfractqquha	42
__satfracthauta	43	__satfractqquhq	42
__satfracthida	49	__satfractqquqq	42
__satfracthidq	49	__satfractqqusa	42
__satfracthiha	49	__satfractqqusq	42
__satfracthihq	49	__satfractqquta	42
__satfracthiqq	49	__satfractsada2	43
__satfracthisa	49	__satfractsadq	43
__satfracthisq	49	__satfractsaha2	43
__satfracthita	49	__satfractsahq	43
__satfracthiuda	49	__satfractsaqq	43
__satfracthiudq	49	__satfractsasq	43
__satfracthiuha	49	__satfractsata2	43
__satfracthiuhq	49	__satfractsauda	44
__satfracthiuqq	49	__satfractsaudq	44
__satfracthiusa	49	__satfractsauha	44
__satfracthiusq	49	__satfractsauhq	43
__satfracthiuta	49	__satfractsauqq	43
__satfracthqda	42	__satfractsausa	44
__satfracthqdq2	42	__satfractsausq	44
__satfracthqha	42	__satfractsauta	44
__satfracthqqq2	42	__satfractsfda	51
__satfracthqsa	42	__satfractsfdq	51
__satfracthqsq2	42	__satfractsfha	51
__satfracthqta	42	__satfractsfhq	51
__satfracthquda	42	__satfractsfq	51
__satfracthqudq	42	__satfractsfsa	51
__satfracthquha	42	__satfractsfsq	51
__satfracthquhq	42	__satfractsfta	51
__satfracthquqq	42	__satfractsfuda	51
__satfracthqusa	42	__satfractsfudq	51
__satfracthqusq	42	__satfractsfuha	51
__satfracthquta	42	__satfractsfuhq	51
__satfractqida	49	__satfractsfuqq	51
__satfractqidq	49	__satfractsfusa	51
__satfractqiha	49	__satfractsfusq	51
__satfractqihq	49	__satfractsfuta	51
__satfractqiqq	49	__satfractsida	50
__satfractqisa	49	__satfractsidq	50
__satfractqisq	49	__satfractsiha	50
__satfractqita	49	__satfractsihq	49
__satfractqiuda	49	__satfractsiqq	49
__satfractqiudq	49	__satfractsisa	50
__satfractqiuha	49	__satfractsisq	49
__satfractqiuhq	49	__satfractsita	50
__satfractqiuqq	49	__satfractsiuda	50
__satfractqiusa	49	__satfractsiudq	50
__satfractqiusq	49	__satfractsiuha	50
__satfractqiuta	49	__satfractsiuhq	50
__satfractqqda	41	__satfractsiuqq	50
__satfractqqdq2	41	__satfractsiusa	50
__satfractqqha	41	__satfractsiusq	50
__satfractqqhq2	41	__satfractsiuta	50
__satfractqqsa	41	__satfractsqda	42
__satfractqqsq2	41	__satfractsqddq2	42
__satfractqqta	42	__satfractsqha	42
__satfractqquda	42	__satfractsqhq2	42
__satfractqqudq	42	__satfractsqqq2	42

__satfractsqsa	42	__satfractudqha	46
__satfractsqta	42	__satfractudqhq	46
__satfractsquda	42	__satfractudqqq	46
__satfractsqudq	42	__satfractudqsa	46
__satfractsquha	42	__satfractudqsq	46
__satfractsquhq	42	__satfractudqta	46
__satfractsquqq	42	__satfractudquda	46
__satfractsqusa	42	__satfractudquha	46
__satfractsqusq	42	__satfractudquhq2	46
__satfractsquta	42	__satfractudquqq2	46
__satfracttada2	44	__satfractudqusa	46
__satfracttadq	44	__satfractudqusq2	46
__satfracttaha2	44	__satfractudquta	47
__satfracttahq	44	__satfractuhada	47
__satfracttaqq	44	__satfractuhadq	47
__satfracttasa2	44	__satfractuhaha	47
__satfracttasq	44	__satfractuhahq	47
__satfracttauda	44	__satfractuhaqq	47
__satfracttaudq	44	__satfractuhasa	47
__satfracttauha	44	__satfractuihasq	47
__satfracttauhq	44	__satfractuhata	47
__satfracttauqq	44	__satfractuhauda2	47
__satfracttausa	44	__satfractuhaudq	47
__satfracttausq	44	__satfractuhauhq	47
__satfracttauta	44	__satfractuhauqq	47
__satfracttida	50	__satfractuhausa2	47
__satfracttidq	50	__satfractuihausq	47
__satfracttiha	50	__satfractuhauta2	47
__satfracttihq	50	__satfractuhqda	45
__satfracttiqq	50	__satfractuhqdq	45
__satfracttisa	50	__satfractuhqha	45
__satfracttisq	50	__satfractuhqhq	45
__satfracttita	50	__satfractuhqqq	45
__satfracttiuda	50	__satfractuhqsa	45
__satfracttiudq	50	__satfractuhqsq	45
__satfracttiuha	50	__satfractuhqta	45
__satfracttiuhq	50	__satfractuhquda	45
__satfracttiuqq	50	__satfractuhqudq2	45
__satfracttiusa	50	__satfractuhquha	45
__satfracttiusq	50	__satfractuhquqq2	45
__satfracttiuta	50	__satfractuhqusa	45
__satfractudada	48	__satfractuhqusq2	45
__satfractudadq	48	__satfractuhquta	45
__satfractudaha	48	__satfractunsdida	58
__satfractudahq	48	__satfractunsdidq	58
__satfractudaqq	48	__satfractunsdiha	58
__satfractudasa	48	__satfractunsdihq	58
__satfractudasq	48	__satfractunsdiqq	58
__satfractudata	48	__satfractunsdisa	58
__satfractudaudq	48	__satfractunsdisq	58
__satfractudauha2	48	__satfractunsdita	58
__satfractudauhq	48	__satfractunsdiuda	58
__satfractudaqq	48	__satfractunsdiudq	58
__satfractudausa2	48	__satfractunsdiuha	58
__satfractudausq	48	__satfractunsdiuhq	58
__satfractudauta2	48	__satfractunsdiuqq	58
__satfractudqda	46	__satfractunsdiusa	58
__satfractudqdq	46	__satfractunsdiusq	58

__satfractunsiuta.....	58	__satfractunstiudq.....	59
__satfractunshida.....	57	__satfractunstiuha.....	59
__satfractunshidq.....	57	__satfractunstiuhq.....	59
__satfractunshiha.....	57	__satfractunstiuqq.....	58
__satfractunshihq.....	57	__satfractunstiusa.....	59
__satfractunshiqq.....	57	__satfractunstiusq.....	59
__satfractunshisa.....	57	__satfractunstiuta.....	59
__satfractunshisq.....	57	__satfractuqqda.....	45
__satfractunshita.....	57	__satfractuqqdq.....	45
__satfractunshiuda.....	57	__satfractuqqha.....	45
__satfractunshiudq.....	57	__satfractuqqhq.....	44
__satfractunshiuha.....	57	__satfractuqqqq.....	44
__satfractunshiuhq.....	57	__satfractuqqsa.....	45
__satfractunshiuqq.....	57	__satfractuqqsq.....	45
__satfractunshiusa.....	57	__satfractuqqta.....	45
__satfractunshiusq.....	57	__satfractuqquda.....	45
__satfractunshiuta.....	57	__satfractuqqudq2.....	45
__satfractunsqida.....	56	__satfractuqquha.....	45
__satfractunsqidq.....	56	__satfractuqquhq2.....	45
__satfractunsqiha.....	56	__satfractuqqusa.....	45
__satfractunsqihq.....	56	__satfractuqqusq2.....	45
__satfractunsqiqq.....	56	__satfractuqquta.....	45
__satfractunsqisa.....	56	__satfractusada.....	47
__satfractunsqisq.....	56	__satfractusadq.....	47
__satfractunsqita.....	56	__satfractusaha.....	47
__satfractunsqiuda.....	57	__satfractusahq.....	47
__satfractunsqiudq.....	57	__satfractusaq.....	47
__satfractunsqiuha.....	57	__satfractusasa.....	47
__satfractunsqiuhq.....	56	__satfractusasq.....	47
__satfractunsqiuqq.....	56	__satfractusata.....	47
__satfractunsqiusa.....	57	__satfractusauda2.....	47
__satfractunsqiusq.....	56	__satfractusaudq.....	47
__satfractunsqiuta.....	57	__satfractusauha2.....	47
__satfractunssida.....	57	__satfractusauhq.....	47
__satfractunssidq.....	57	__satfractusauqq.....	47
__satfractunssiha.....	57	__satfractusausq.....	47
__satfractunssihq.....	57	__satfractusauta2.....	48
__satfractunssiqq.....	57	__satfractusqda.....	46
__satfractunssisa.....	57	__satfractusqdq.....	46
__satfractunssisq.....	57	__satfractusqha.....	46
__satfractunssita.....	57	__satfractusqhq.....	45
__satfractunssiuda.....	58	__satfractusqqq.....	45
__satfractunssiudq.....	58	__satfractusqsa.....	46
__satfractunssiuha.....	58	__satfractusqsq.....	45
__satfractunssiuhq.....	57	__satfractusqta.....	46
__satfractunssiuqq.....	57	__satfractusquda.....	46
__satfractunssiusa.....	58	__satfractusqud2.....	46
__satfractunssiusq.....	58	__satfractusquha.....	46
__satfractunssiuta.....	58	__satfractusquhq2.....	46
__satfractunstida.....	58	__satfractusquqq2.....	46
__satfractunstidq.....	58	__satfractusqusa.....	46
__satfractunstiha.....	58	__satfractusquta.....	46
__satfractunstihq.....	58	__satfractutada.....	48
__satfractunstiqq.....	58	__satfractutadq.....	48
__satfractunstisa.....	58	__satfractutaha.....	48
__satfractunstisq.....	58	__satfractutahq.....	48
__satfractunstita.....	58	__satfractutaqq.....	48
__satfractunstiuda.....	59	__satfractutasa.....	48

__satfractutasq	48	__subdf3	9
__satfractutata	48	__subdq3	22
__satfractutauda2	49	__subha3	22
__satfractutaudq	49	__subhq3	22
__satfractutauha2	49	__subqq3	22
__satfractutauhq	48	__subsa3	22
__satfractutauqq	48	__subsf3	9
__satfractutausa2	49	__subsq3	22
__satfractutausq	48	__subta3	22
__splitstack_find	60	__subtf3	9
__ssadda3	21	__subuda3	22
__ssadddq3	21	__subudq3	22
__ssaddha3	21	__subuha3	22
__ssaddhq3	21	__subuhq3	22
__ssaddqq3	21	__subuqq3	22
__ssaddsa3	21	__subusa3	22
__ssadds3	21	__subusq3	22
__ssaddta3	21	__subuta3	22
__ssashlda3	28	__subvdi3	7
__ssashldq3	28	__subvsi3	7
__ssashlha3	28	__subxf3	9
__ssashlhq3	28	__truncdfsf2	10
__ssashlsa3	28	__trunctfdf2	10
__ssashlsq3	28	__trunctfsf2	10
__ssashlta3	28	__truncxfdf2	10
__ssdivda3	25	__truncxfsf2	10
__ssdivdq3	25	__ucmpdi2	7
__ssdivha3	25	__ucmpti2	7
__ssdivhq3	25	__udivdi3	6
__ssdivqq3	25	__udivmoddi4	6
__ssdivsa3	25	__udivmodti4	6
__ssdivsq3	25	__udivsi3	6
__ssdivta3	25	__udivti3	6
__ssmulda3	24	__udivuda3	25
__ssmuldq3	24	__udivudq3	25
__ssmulha3	24	__udivuha3	25
__ssmulhq3	24	__udivuhq3	25
__ssmulqq3	24	__udivuqq3	25
__ssmulsa3	24	__udivusa3	25
__ssmulsq3	24	__udivusq3	25
__ssmulta3	24	__udivuta3	25
__ssnegda2	26	__umoddi3	6
__ssnegdq2	26	__umodsi3	6
__ssnegha2	26	__umodti3	6
__ssneghq2	26	__unorddf2	12
__ssnegqq2	26	__unordsf2	12
__ssnegsa2	26	__unordtf2	12
__ssnegsq2	26	__usadduda3	22
__ssnegta2	26	__usaddudq3	22
__sssubda3	23	__usadduha3	22
__sssubdq3	23	__usadduhq3	22
__sssubha3	23	__usadduqq3	22
__sssubhq3	23	__usaddusa3	22
__sssubqq3	23	__usaddusq3	22
__sssubsa3	23	__usadduta3	22
__sssubsq3	23	__usashluda3	28
__sssubta3	23	__usashludq3	28
__subda3	22	__usashluha3	28

<code>__usashluhq3</code>	28
<code>__usashluqq3</code>	28
<code>__usashlusa3</code>	28
<code>__usashlusq3</code>	28
<code>__usashluta3</code>	29
<code>__usdivuda3</code>	26
<code>__usdivudq3</code>	26
<code>__usdivuha3</code>	26
<code>__usdivuhq3</code>	25
<code>__usdivuqq3</code>	25
<code>__usdivusa3</code>	26
<code>__usdivusq3</code>	26
<code>__usdivuta3</code>	26
<code>__usmuluda3</code>	24
<code>__usmuludq3</code>	24
<code>__usmuluha3</code>	24
<code>__usmuluhq3</code>	24
<code>__usmuluqq3</code>	24
<code>__usmulusa3</code>	24
<code>__usmulusq3</code>	24
<code>__usmuluta3</code>	24
<code>__usneguda2</code>	27
<code>__usnegudq2</code>	27
<code>__usneguha2</code>	27
<code>__usneguhq2</code>	27
<code>__usneguqq2</code>	27
<code>__usnegusa2</code>	27
<code>__usnegusq2</code>	27
<code>__usneguta2</code>	27
<code>__ussubuda3</code>	23
<code>__ussubudq3</code>	23
<code>__ussubuha3</code>	23
<code>__ussubuhq3</code>	23
<code>__ussubuqq3</code>	23
<code>__ussubusa3</code>	23
<code>__ussubusq3</code>	23
<code>__ussubuta3</code>	23

‘

“real” instructions, RTL SSA	339
------------------------------------	-----

@

‘@’ in instruction pattern names	526
--	-----

\

\\	376
----------	-----

0

‘0’ in constraint	387
-------------------------	-----

A

<code>abort</code>	3
<code>abs</code>	315
<code>abs</code> and attributes	502
<code>ABS_EXPR</code>	195
<code>absence_set</code>	512
<code>absm2</code> instruction pattern	447
absolute value	315
<code>ABSU_EXPR</code>	195
access to operands	286
access to special operands	287
accessors	286
<code>ACCUM_TYPE_SIZE</code>	552
<code>ACCUMULATE_OUTGOING_ARGS</code>	588
<code>ACCUMULATE_OUTGOING_ARGS</code> and stack frames	603
<code>acosm2</code> instruction pattern	448
<code>ADA_LONG_TYPE_SIZE</code>	551
Adding a new GIMPLE statement code	268
<code>ADDITIONAL_REGISTER_NAMES</code>	673
<code>addm3</code> instruction pattern	436
<code>addmodecc</code> instruction pattern	460
<code>addptrm3</code> instruction pattern	437
<code>addr_diff_vec</code>	325
<code>addr_diff_vec</code> , length of	506
<code>addr_vec</code>	325
<code>addr_vec</code> , length of	506
<code>ADDR_EXPR</code>	194
address constraints	387
<code>address_operand</code>	382, 387
addressing modes	616
<code>addvm4</code> instruction pattern	437
<code>ADJUST_FIELD_ALIGN</code>	545
<code>ADJUST_INSN_LENGTH</code>	506
<code>ADJUST_REG_ALLOC_ORDER</code>	559
aggregates as return values	600
alias	280
<code>ALL_REGS</code>	563
<code>allocate_stack</code> instruction pattern	470
alternate entry points	330
analyzer	773
analyzer, debugging	781
analyzer, internals	773
anchored addresses	627
<code>and</code>	314
<code>and</code> and attributes	501
<code>and</code> , canonicalization of	485
<code>andm3</code> instruction pattern	436
<code>andnm3</code> instruction pattern	437
<code>ANNOTATE_EXPR</code>	195
annotations	271
<code>APPLY_RESULT_SIZE</code>	599
<code>arg_pointer_rtx</code>	585
<code>ARG_POINTER_CFA_OFFSET</code>	578
<code>ARG_POINTER_REGNUM</code>	583
<code>ARG_POINTER_REGNUM</code> and virtual registers	307
<code>ARGS_GROW_DOWNWARD</code>	575
arguments in registers	589

arguments on stack	587	ASM_OUTPUT_DWARF_DATAREL	684
arithmetic library	9	ASM_OUTPUT_DWARF_DELTA	684
arithmetic shift	315	ASM_OUTPUT_DWARF_OFFSET	684
arithmetic shift with signed saturation	315	ASM_OUTPUT_DWARF_PCREL	684
arithmetic shift with unsigned saturation	315	ASM_OUTPUT_DWARF_TABLE_REF	684
arithmetic, in RTL	312	ASM_OUTPUT_DWARF_VMS_DELTA	684
ARITHMETIC_TYPE_P	220	ASM_OUTPUT_EXTERNAL	666
array	182	ASM_OUTPUT_FDESC	657
ARRAY_RANGE_REF	194	ASM_OUTPUT_FUNCTION_LABEL	661
ARRAY_REF	194	ASM_OUTPUT_INTERNAL_LABEL	661
ARRAY_TYPE	182	ASM_OUTPUT_LABEL	661
AS_NEEDS_DASH_FOR_PIPED_INPUT	531	ASM_OUTPUT_LABEL_REF	666
ashift	315	ASM_OUTPUT_LABELREF	666
ashift and attributes	502	ASM_OUTPUT_LOCAL	660
ashiftrt	315	ASM_OUTPUT_MAX_SKIP_ALIGN	681
ashiftrt and attributes	502	ASM_OUTPUT_MEASURED_SIZE	661
ashlm3 instruction pattern	446	ASM_OUTPUT_OPCODE	673
ashrm3 instruction pattern	446	ASM_OUTPUT_POOL_EPILOGUE	658
asinm2 instruction pattern	448	ASM_OUTPUT_POOL_PROLOGUE	657
asm_fprintf	675	ASM_OUTPUT_REG_POP	676
asm_input	325	ASM_OUTPUT_REG_PUSH	676
asm_input and '/v'	291	ASM_OUTPUT_SIZE_DIRECTIVE	661
asm_noperands	333	ASM_OUTPUT_SKIP	681
asm_operands and '/v'	291	ASM_OUTPUT_SOURCE_FILENAME	654
asm_operands, RTL sharing	347	ASM_OUTPUT_SPECIAL_POOL_ENTRY	658
asm_operands, usage	327	ASM_OUTPUT_SYMBOL_REF	666
ASM_APP_OFF	654	ASM_OUTPUT_TYPE_DIRECTIVE	662
ASM_APP_ON	654	ASM_OUTPUT_WEAK_ALIAS	668
ASM_COMMENT_START	654	ASM_OUTPUT_WEAKREF	664
ASM_DECLARE_COLD_FUNCTION_NAME	663	ASM_PREFERRED_EH_DATA_FORMAT	580
ASM_DECLARE_COLD_FUNCTION_SIZE	663	ASM_SPEC	531
ASM_DECLARE_FUNCTION_NAME	662	ASM_WEAKEN_DECL	664
ASM_DECLARE_FUNCTION_SIZE	662	ASM_WEAKEN_LABEL	664
ASM_DECLARE_OBJECT_NAME	663	assemble_name	661
ASM_DECLARE_REGISTER_GLOBAL	663	assemble_name_raw	661
ASM_FINAL_SPEC	531	assembler format	653
ASM_FINISH_DECLARE_OBJECT	664	assembler instructions in RTL	327
ASM_FORMAT_PRIVATE_NAME	667	ASSEMBLER_DIALECT	675
ASM_FPRINTF_EXTENSIONS	675	assigning attribute values to insns	503
ASM_GENERATE_INTERNAL_LABEL	667	ASSUME_EXTENDED_UNWIND_CONTEXT	585
ASM_MAYBE_OUTPUT_ENCODED_ADDR_RTX	580	asterisk in template	377
ASM_NO_SKIP_IN_TEXT	681	atan2m3 instruction pattern	449
ASM_OUTPUT_ADDR_DIFF_ELT	676	atanm2 instruction pattern	448
ASM_OUTPUT_ADDR_VEC_ELT	676	atomic	742
ASM_OUTPUT_ALIGN	681	atomic_add_fetch_cmp_0mode	
ASM_OUTPUT_ALIGN_WITH_NOP	681	instruction pattern	478
ASM_OUTPUT_ALIGNED_BSS	660	atomic_add_fetchmode instruction pattern	477
ASM_OUTPUT_ALIGNED_COMMON	659	atomic_addmode instruction pattern	477
ASM_OUTPUT_ALIGNED_DECL_COMMON	659	atomic_and_fetch_cmp_0mode	
ASM_OUTPUT_ALIGNED_DECL_LOCAL	660	instruction pattern	478
ASM_OUTPUT_ALIGNED_LOCAL	660	atomic_and_fetchmode instruction pattern	477
ASM_OUTPUT_ASCHII	657	atomic_andmode instruction pattern	477
ASM_OUTPUT_CASE_END	677	atomic_bit_test_and_complementmode	
ASM_OUTPUT_CASE_LABEL	676	instruction pattern	478
ASM_OUTPUT_COMMON	659	atomic_bit_test_and_resetmode	
ASM_OUTPUT_DEBUG_LABEL	667	instruction pattern	478
ASM_OUTPUT_DEF	667	atomic_bit_test_and_setmode	
ASM_OUTPUT_DEF_FROM_DECLS	668	instruction pattern	478

atomic_compare_and_swapmode	
instruction pattern	476
atomic_exchangemode instruction pattern	477
atomic_fetch_addmode instruction pattern	477
atomic_fetch_andmode instruction pattern	477
atomic_fetch_nandmode instruction pattern	477
atomic_fetch_ormode instruction pattern	477
atomic_fetch_submode instruction pattern	477
atomic_fetch_xormode instruction pattern	477
atomic_loadmode instruction pattern	476
atomic_nand_fetchmode instruction pattern	477
atomic_nandmode instruction pattern	477
atomic_or_fetch_cmp_0mode	
instruction pattern	478
atomic_or_fetchmode instruction pattern	477
atomic_ormode instruction pattern	477
atomic_storemode instruction pattern	476
atomic_sub_fetch_cmp_0mode	
instruction pattern	478
atomic_sub_fetchmode instruction pattern	477
atomic_submode instruction pattern	477
atomic_test_and_set instruction pattern	477
atomic_xor_fetch_cmp_0mode	
instruction pattern	478
atomic_xor_fetchmode instruction pattern	477
atomic_xormode instruction pattern	477
attr	503, 504
attr_flag	503
attribute expressions	501
attribute specifications	505
attribute specifications example	505
ATTRIBUTE_ALIGNED_VALUE	544
attributes	191
attributes, defining	499
attributes, target-specific	688
autoincrement addressing, availability	3
autoincrement/decrement addressing	385
automata_option	513
automaton based pipeline description	508, 509
automaton based scheduler	508
avgm3_ceil instruction pattern	446
avgm3_floor instruction pattern	446
AVOID_CCMode_COPIES	562

B

backslash	376
barrier	331
barrier and '/f'	292
barrier and '/v'	290
BASE_REG_CLASS	565
basic block	349
basic blocks, RTL SSA	339
Basic Statements	207
basic-block.h	349
basic_block	339, 349
BASIC_BLOCK	349
bb_seq	264

BB_HEAD, BB_END	356
BIGGEST_ALIGNMENT	544
BIGGEST_FIELD_ALIGNMENT	545
BImode	295
BIND_EXPR	195
BINFO_TYPE	222
bit-fields	318
BIT_AND_EXPR	195
BIT_IOR_EXPR	195
BIT_NOT_EXPR	195
BIT_XOR_EXPR	195
BITFIELD_NBYTES_LIMITED	548
BITINT_TYPE	182
bitreverse	316
BITS_BIG_ENDIAN	541
BITS_BIG_ENDIAN , effect on sign_extract	318
BITS_PER_UNIT	302
BITS_PER_WORD	542
bitwise complement	314
bitwise exclusive-or	315
bitwise inclusive-or	315
bitwise logical-and	314
BLKmode	297
BLKmode , and function return values	337
BLOCK_FOR_INSN, gimple_bb	355
BLOCK_REG_PADDING	593
blockage instruction pattern	473
Blocks	209
BND32mode	298
BND64mode	298
bool	717
BOOL_TYPE_SIZE	552
BOOLEAN_TYPE	182
branch prediction	354
BRANCH_COST	633
break_out_memory_refs	618
BREAK_STMT	226
BSS_SECTION_ASM_OP	648
bswap	316
bswapm2 instruction pattern	447
BTF_DEBUGGING_INFO	685
btruncm2 instruction pattern	450
build0	180
build1	180
build2	180
build3	180
build4	180
build5	180
build6	180
builtin_longjmp instruction pattern	471
builtin_setjmp_receiver	
instruction pattern	471
builtin_setjmp_setup instruction pattern	471
byte_mode	302
BYTES_BIG_ENDIAN	541
BYTES_BIG_ENDIAN , effect on subreg	310

C

<code>c_register_pragma</code>	708
<code>c_register_pragma_with_expansion</code>	708
C statements for assembler output.....	376
<code>C_COMMON_OVERRIDE_OPTIONS</code>	539
<code>cache</code>	740
<code>cadd270m3</code> instruction pattern.....	451
<code>cadd90m3</code> instruction pattern.....	451
<code>call</code>	293, 322
<code>call</code> instruction pattern.....	466
<code>call</code> usage.....	337
<code>call</code> , in <code>call_insn</code>	292
<code>call</code> , in <code>mem</code>	291
<code>call-clobbered</code> register.....	557
<code>call-saved</code> register.....	557
<code>call-used</code> register.....	557
<code>call_insn</code>	330
<code>call_insn</code> and <code>‘/c’</code>	292
<code>call_insn</code> and <code>‘/f’</code>	292
<code>call_insn</code> and <code>‘/i’</code>	291
<code>call_insn</code> and <code>‘/j’</code>	292
<code>call_insn</code> and <code>‘/s’</code>	290, 292
<code>call_insn</code> and <code>‘/u’</code>	290, 291
<code>call_insn</code> and <code>‘/u’</code> or <code>‘/i’</code>	292
<code>call_insn</code> and <code>‘/v’</code>	290
<code>call_pop</code> instruction pattern.....	466
<code>call_used_regs</code>	558
<code>call_value</code> instruction pattern.....	466
<code>call_value_pop</code> instruction pattern.....	466
<code>CALL_EXPR</code>	195
<code>CALL_INSN_FUNCTION_USAGE</code>	330
<code>CALL_POPS_ARGS</code>	589
<code>CALL_REALLY_USED_REGISTERS</code>	557
<code>CALL_USED_REGISTERS</code>	557
<code>callback</code>	739
calling conventions.....	574
calling functions in RTL.....	337
<code>can_create_pseudo_p</code>	427
<code>can_fallthru</code>	350
<code>canadian</code>	63
canonicalization of instructions.....	484
<code>canonicalize_funcptr_for_compare</code> instruction pattern.....	469
<code>caret</code>	390, 787
<code>CASE_VECTOR_MODE</code>	702
<code>CASE_VECTOR_PC_RELATIVE</code>	702
<code>CASE_VECTOR_SHORTEN_MODE</code>	702
<code>casesi</code> instruction pattern.....	468
<code>cbranchmode4</code> instruction pattern.....	465
<code>CC1_SPEC</code>	531
<code>CC1PLUS_SPEC</code>	531
<code>CCmode</code>	297, 629
<code>CDImode</code>	298
<code>CEIL_DIV_EXPR</code>	195
<code>CEIL_MOD_EXPR</code>	195
<code>ceilm2</code> instruction pattern.....	450
<code>CFA_FRAME_BASE_OFFSET</code>	578
CFG verification.....	357
CFG, Control Flow Graph.....	349
<code>cfghooks.h</code>	355
<code>cgraph_finalize_function</code>	145
<code>chain_circular</code>	741
<code>chain_next</code>	741
<code>chain_prev</code>	741
<code>change_address</code>	426
<code>CHAR_TYPE_SIZE</code>	552
<code>check_raw_ptrsm</code> instruction pattern.....	433
<code>check_stack</code> instruction pattern.....	470
<code>check_war_ptrsm</code> instruction pattern.....	433
<code>CHImode</code>	298
class definitions, register.....	563
class preference constraints.....	390
class, scope.....	222
<code>CLASS_MAX_NREGS</code>	572
<code>CLASS_TYPE_P</code>	220
classes of RTX codes.....	284
<code>CLASSTYPE_DECLARED_CLASS</code>	222
<code>CLASSTYPE_HAS_MUTABLE</code>	223
<code>CLASSTYPE_NON_POD_P</code>	223
<code>CLEANUP_DECL</code>	226
<code>CLEANUP_EXPR</code>	226
<code>CLEANUP_POINT_EXPR</code>	195
<code>CLEANUP_STMT</code>	226
Cleanups.....	210
<code>clear_cache</code> instruction pattern.....	480
<code>CLEAR_INSN_CACHE</code>	614
<code>CLEAR_RATIO</code>	635
<code>clobber</code>	322
<code>clrsb</code>	315
<code>clrsbm2</code> instruction pattern.....	454
<code>clz</code>	315
<code>CLZ_DEFINED_VALUE_AT_ZERO</code>	706
<code>clzm2</code> instruction pattern.....	454
<code>cmla_conjm4</code> instruction pattern.....	452
<code>cmlam4</code> instruction pattern.....	452
<code>cmls_conjm4</code> instruction pattern.....	453
<code>cmlsm4</code> instruction pattern.....	452
<code>cmpmemm</code> instruction pattern.....	457
<code>cmpstrm</code> instruction pattern.....	457
<code>cmpstrnm</code> instruction pattern.....	456
<code>cmul_conjm4</code> instruction pattern.....	453
<code>cmulm4</code> instruction pattern.....	453
code generation RTL sequences.....	486
code iterators in <code>.md</code> files.....	523
<code>code_label</code>	330
<code>code_label</code> and <code>‘/i’</code>	290
<code>code_label</code> and <code>‘/v’</code>	290
<code>CODE_LABEL</code>	350
<code>CODE_LABEL_NUMBER</code>	330
codes, RTL expression.....	283
<code>COImode</code>	298
<code>COLLECT_EXPORT_LIST</code>	716
<code>COLLECT_SHARED_FINI_FUNC</code>	671
<code>COLLECT_SHARED_INIT_FUNC</code>	671
<code>COLLECT2_HOST_INITIALIZATION</code>	727
command-line options, guidelines for.....	793

commit_edge_insertions	356	cond_len_modmode instruction pattern	462
compact syntax	378	cond_len_mulmode instruction pattern	462
compare	313	cond_len_negmode instruction pattern	462
compare, canonicalization of	484	cond_len_one_cmplmode instruction pattern...	462
COMPARE_MAX_PIECES	635	cond_len_rintmode instruction pattern	462
comparison_operator	382	cond_len_roundmode instruction pattern	462
compiler passes and files	145	cond_len_smaxmode instruction pattern	462
complement, bitwise	314	cond_len_sminmode instruction pattern	462
complex_mode	299	cond_len_sqrtmode instruction pattern	462
COMPLEX_CST	192	cond_len_submode instruction pattern	462
COMPLEX_EXPR	195	cond_len_udivmode instruction pattern	462
COMPLEX_TYPE	182	cond_len_umaxmode instruction pattern	462
COMPONENT_REF	194	cond_len_uminmode instruction pattern	462
compose_tag instruction pattern	480	cond_len_umodmode instruction pattern	462
Compound Expressions	239	cond_len_vec_cbranch_allmode	
Compound Lvalues	239	instruction pattern	466
COMPOUND_EXPR	195	cond_len_vec_cbranch_anymode	
COMPOUND_LITERAL_EXPR	195	instruction pattern	465
COMPOUND_LITERAL_EXPR_DECL	202	cond_len_xormode instruction pattern	462
COMPOUND_LITERAL_EXPR_DECL_EXPR	202	cond_lshrmode instruction pattern	461
computed jump	352	cond_modmode instruction pattern	461
computing the length of an insn	505	cond_mulmode instruction pattern	461
concat	312	cond_negmode instruction pattern	460
concatn	312	cond_one_cmplmode instruction pattern	460
cond	317	cond_rintmode instruction pattern	460
cond and attributes	501	cond_roundmode instruction pattern	460
cond_addmode instruction pattern	461	cond_smaxmode instruction pattern	461
cond_andmode instruction pattern	461	cond_sminmode instruction pattern	461
cond_ashlmode instruction pattern	461	cond_sqrtmode instruction pattern	460
cond_ashrmode instruction pattern	461	cond_submode instruction pattern	461
cond_ceilmode instruction pattern	460	cond_udivmode instruction pattern	461
cond_copysignmode instruction pattern	461	cond_umaxmode instruction pattern	461
cond_divmode instruction pattern	461	cond_uminmode instruction pattern	461
cond_exec	324	cond_umodmode instruction pattern	461
cond_floormode instruction pattern	460	cond_vec_cbranch_allmode	
cond_fmamode instruction pattern	462	instruction pattern	465
cond_fmaxmode instruction pattern	461	cond_vec_cbranch_anymode	
cond_fminmode instruction pattern	461	instruction pattern	465
cond_fmsmode instruction pattern	462	cond_xormode instruction pattern	461
cond_fnmamode instruction pattern	462	COND_EXPR	195
cond_fnmsmode instruction pattern	462	condition code status	628
cond_iormode instruction pattern	461	condition codes	316
cond_len_addmode instruction pattern	462	conditional execution	514
cond_len_andmode instruction pattern	462	Conditional Expressions	239
cond_len_ashlmode instruction pattern	462	conditions, in patterns	370
cond_len_ashrmode instruction pattern	462	configuration file	726, 727
cond_len_ceilmode instruction pattern	462	configure terms	63
cond_len_copysignmode instruction pattern...	462	CONJ_EXPR	195
cond_len_divmode instruction pattern	462	const	306
cond_len_floormode instruction pattern	462	const_double	303
cond_len_fmamode instruction pattern	463	const_double, RTL sharing	347
cond_len_fmaxmode instruction pattern	462	const_double_operand	381
cond_len_fminmode instruction pattern	462	const_double_zero	303
cond_len_fmsmode instruction pattern	463	const_fixed	304
cond_len_fnmamode instruction pattern	463	const_int	302
cond_len_fnmsmode instruction pattern	463	const_int and attribute tests	501
cond_len_iormode instruction pattern	462	const_int and attributes	501
cond_len_lshrmode instruction pattern	462	const_int, RTL sharing	347

const_int_operand.....	381	CP_TYPE_VOLATILE_P.....	219
const_poly_int.....	304	CPLUSPLUS_CPP_SPEC.....	531
const_poly_int, RTL sharing.....	347	CPP_SPEC.....	530
const_string.....	305	CPSImode.....	298
const_string and attributes.....	501	cpymem instruction pattern.....	455
const_true_rtx.....	303	CQImode.....	298
const_vector.....	304	crc_revmm4 instruction pattern.....	481
const_vector, RTL sharing.....	347	crcmm4 instruction pattern.....	480
const0_rtx.....	302	cross compilation and floating point.....	685
const1_rtx.....	302	CROSSING_JUMP_P.....	290
const2_rtx.....	302	CRT_CALL_STATIC_FUNCTION.....	649
CONST_DECL.....	186	crtl->args.pops_args.....	603
CONST_DOUBLE_LOW.....	303	crtl->args.pretend_args_size.....	603
CONST_WIDE_INT.....	303	crtl->outgoing_args_size.....	588
CONST_WIDE_INT_ELT.....	304	CRTSTUFF_T_CFLAGS.....	729
CONST_WIDE_INT_NUNITS.....	304	CRTSTUFF_T_CFLAGS_S.....	729
CONST_WIDE_INT_VEC.....	303	CSImode.....	298
CONST0_RTX.....	306	ctstoremode4 instruction pattern.....	464
CONST1_RTX.....	306	CTF_DEBUGGING_INFO.....	685
CONST2_RTX.....	306	CTImode.....	298
constant attributes.....	507	ctrappmm4 instruction pattern.....	473
constant definitions.....	518	ctz.....	316
CONSTANT_ADDRESS_P.....	617	CTZ_DEFINED_VALUE_AT_ZERO.....	706
CONSTANT_P.....	617	ctzm2 instruction pattern.....	454
CONSTANT_POOL_ADDRESS_P.....	290	CUMULATIVE_ARGS.....	591
CONSTANT_POOL_BEFORE_FUNCTION.....	657	current_function_is_leaf.....	562
constants in constraints.....	386	current_function_uses_only_leaf_regs.....	562
constm1_rtx.....	302	current_insn_predicate.....	515
constraint modifier characters.....	390		
constraint, matching.....	387	D	
constraint_num.....	425	DAmode.....	297
constraint_satisfied_p.....	425	data bypass.....	510, 511
constraints.....	385	data dependence delays.....	508
constraints, defining.....	421	Data Dependency Analysis.....	366
constraints, machine specific.....	392	data structures.....	540
constraints, testing.....	425	DATA_ABI_ALIGNMENT.....	546
constructors, automatic calls.....	733	DATA_ALIGNMENT.....	545
constructors, output of.....	668	DATA_SECTION_ASM_OP.....	648
CONSTRUCTOR.....	195	dbr_sequence_length.....	675
container.....	181	DBR_OUTPUT_SEQEND.....	675
CONTINUE_STMT.....	226	DCmode.....	298
contributors.....	819	DDmode.....	296
controlling register usage.....	558	De Morgan's law.....	485
controlling the compilation driver.....	530	dead_or_set_p.....	496
conversions.....	319	debug_expr.....	328
CONVERT_EXPR.....	195	debug_implicit_ptr.....	328
copy_rtx.....	619	debug_insn.....	332
copy_rtx_if_shared.....	348	debug_marker.....	328
copysign.....	316	debug_parameter_ref.....	328
copysignm3 instruction pattern.....	451	DEBUG_EXPR_DECL.....	186
cosm2 instruction pattern.....	447	DEBUGGER_ARG_OFFSET.....	682
costs of instructions.....	631	DEBUGGER_AUTO_OFFSET.....	682
cp_namespace_decls.....	221	DEBUGGER_REGNO.....	682
cp_type_quals.....	219	decimal float library.....	14
CP_INTEGRAL_TYPE.....	219	DECL_ALIGN.....	186
CP_TYPE_CONST_NON_VOLATILE_P.....	219	DECL_ANTICIPATED.....	224
CP_TYPE_CONST_P.....	219	DECL_ARGUMENTS.....	216
CP_TYPE_RESTRICT_P.....	219		

DECL_ARRAY_DELETE_OPERATOR_P	226	define_cond_exec	514
DECL_ARTIFICIAL	186, 215, 217	define_constants	518
DECL_ASSEMBLER_NAME	215	define_constraint	422
DECL_ATTRIBUTES	191	define_cpu_unit	509
DECL_BASE_CONSTRUCTOR_P	225	define_delay	508
DECL_COMPLETE_CONSTRUCTOR_P	225	define_enum	520
DECL_COMPLETE_DESTRUCTOR_P	225	define_enum_attr	500, 520
DECL_CONST_MEMFUNC_P	224	define_expand	486
DECL_CONSTRUCTOR_P	224	define_insn	369
DECL_CONTEXT	221	define_insn example	371
DECL_CONV_FN_P	225	define_insn_and_rewrite	492
DECL_COPY_CONSTRUCTOR_P	225	define_insn_and_split	491
DECL_DESTRUCTOR_P	225	define_insn_reservation	510
DECL_EXTERN_C_FUNCTION_P	224	define_int_attr	524
DECL_EXTERNAL	186, 217	define_int_iterator	524
DECL_FUNCTION_MEMBER_P	224	define_memory_constraint	423
DECL_FUNCTION_SPECIFIC_OPTIMIZATION	215, 218	define_mode_attr	521
DECL_FUNCTION_SPECIFIC_TARGET	215, 217	define_mode_iterator	521
DECL_GLOBAL_CTOR_P	225	define_peekhole	495
DECL_GLOBAL_DTOR_P	225	define_peekhole2	497
DECL_INITIAL	186, 216	define_predicate	383
DECL_LINKONCE_P	224	define_query_cpu_unit	510
DECL_LOCAL_FUNCTION_P	224	define_register_constraint	422
DECL_MAIN_P	224	define_relaxed_memory_constraint	424
DECL_NAME	186, 215, 221	define_reservation	511
DECL_NAMESPACE_ALIAS	221	define_special_memory_constraint	423
DECL_NAMESPACE_STD_P	221	define_special_predicate	383
DECL_NON_THUNK_FUNCTION_P	225	define_split	489
DECL_NONCONVERTING_P	225	define_subst	516, 517, 518, 525
DECL_NONSTATIC_MEMBER_FUNCTION_P	224	define_subst_attr	525
DECL_OVERLOADED_OPERATOR_P	225	defining attributes and their values	499
DECL_PURE_P	217	defining constraints	421
DECL_RESULT	216	defining jump instruction patterns	482
DECL_SAVED_TREE	216	defining looping instruction patterns	482
DECL_SIZE	186	defining peephole optimizers	495
DECL_STATIC_FUNCTION_P	224	defining predicates	383
DECL_STMT	226	defining RTL sequences for code generation	486
DECL_STMT_DECL	226	degenerate phi node, RTL SSA	342
DECL_THUNK_P	225	delay slots, defining	507
DECL_VIRTUAL_P	217	deletable	741
DECL_VOLATILE_MEMFUNC_P	224	DELETE_IF_ORDINARY	727
declaration	186	Dependent Patterns	481
declarations, RTL	321	desc	740
DECLARE_LIBRARY_RENAMES	614	descriptors for nested functions	611
default	740	destructors, output of	668
default_file_start	653	deterministic finite state automaton	508, 513
DEFAULT_GDB_EXTENSIONS	682	DFmode	296
DEFAULT_INCOMING_FRAME_SP_OFFSET	577	diagnostics guidelines, fix-it hints	791
DEFAULT_PCC_STRUCT_RETURN	600	diagnostics, actionable	785
DEFAULT_SIGNED_CHAR	553	diagnostics, false positive	785
define_address_constraint	424	diagnostics, guidelines for	785
define_asm_attributes	504	diagnostics, locations	787
define_attr	499	diagnostics, true positive	785
define_automaton	509	digits in constraint	387
define_bypass	511	DImode	296
define_c_enum	519	DIR_SEPARATOR	726
define_code_attr	523	DIR_SEPARATOR_2	726
define_code_iterator	523	directory options .md	494

disabling certain registers 558
 dispatch table 676
 div 314
 div and attributes 502
 division 314
 divm3 instruction pattern 436
 divmodm4 instruction pattern 445
 DO_BODY 226
 DO_COND 226
 DO_STMT 226
 dollar sign 390
 DOLLARS_IN_IDENTIFIERS 709
 doloop_begin instruction pattern 469
 doloop_end instruction pattern 468
 DONE 487, 489, 498
 DONT_USE_BUILTIN_SETJMP 679
 DQmode 297
 driver 530
 DRIVER_SELF_SPECS 530
 dump examples 163
 dump setup 160
 dump types 162
 dump verbosity 162
 dump_basic_block 162
 dump_generic_expr 162
 dump_gimple_stmt 162
 dump_printf 162
 DUMPFILE_FORMAT 727
 DWARF_ALT_FRAME_RETURN_COLUMN 576
 DWARF_CIE_DATA_ALIGNMENT 679
 DWARF_FRAME_REGISTERS 585
 DWARF_FRAME_REGNUM 585
 DWARF_LAZY_REGISTER_VALUE 586
 DWARF_REG_TO_UNWIND_COLUMN 585
 DWARF_VERSION_DEFAULT 577
 DWARF_ZERO_REG 577
 DWARF2_ASM_LINE_DEBUG_INFO 683
 DWARF2_ASM_VIEW_DEBUG_INFO 683
 DWARF2_DEBUGGING_INFO 683
 DWARF2_FRAME_INFO 683
 DWARF2_FRAME_REG_OUT 585
 DWARF2_UNWIND_INFO 678
 DYNAMIC_CHAIN_ADDRESS 575

E

‘E’ in constraint 386
 earlyclobber operand 391
 edge 351
 edge in the flow graph 351
 edge iterators 351
 edge splitting 356
 EDGE_ABNORMAL 352
 EDGE_ABNORMAL, EDGE_ABNORMAL_CALL 353
 EDGE_ABNORMAL, EDGE_EH 352
 EDGE_ABNORMAL, EDGE_SIBCALL 352
 EDGE_FALLTHRU, force_nonfallthru 352
 EDOM, implicit usage 615

eh_return instruction pattern 472
 EH_FRAME_SECTION_NAME 678
 EH_FRAME_THROUGH_COLLECT2 678
 EH_RETURN_DATA_REGNO 579
 EH_RETURN_HANDLER_RTX 580
 EH_RETURN_STACKADJ_RTX 579
 EH_RETURN_TAKEN_RTX 580
 EH_TABLES_CAN_BE_READ_ONLY 678
 EH_USES 604
 ei_edge 351
 ei_end_p 351
 ei_last 351
 ei_next 351
 ei_one_before_end_p 351
 ei_prev 351
 ei_safe_edge 351
 ei_start 351
 ELIMINABLE_REGS 586
 ELSE_CLAUSE 226
 Embedded C 20
 Empty Statements 209
 EMPTY_CLASS_EXPR 226
 EMPTY_FIELD_BOUNDARY 547
 Emulated TLS 693
 enabled 420
 ENDFILE_SPEC 533
 endianness 3
 entry_value 328
 ENTRY_BLOCK_PTR, EXIT_BLOCK_PTR 349
 enum reg_class 564
 ENUMERAL_TYPE 182
 enumerations 519
 epilogue 601
 epilogue instruction pattern 472
 EPILOGUE_USES 604
 eq 317
 eq and attributes 502
 eq_attr 502
 EQ_EXPR 195
 equal 317
 errno, implicit usage 615
 EXACT_DIV_EXPR 195
 examining SSA_NAMES 279
 exception handling 352, 579
 exception_receiver instruction pattern 471
 exclamation point 390
 exclusion_set 512
 exclusive-or, bitwise 315
 EXIT_EXPR 195
 EXIT_IGNORE_STACK 604
 exp10m2 instruction pattern 449
 exp2m2 instruction pattern 449
 expander definitions 486
 expm1m2 instruction pattern 448
 expm2 instruction pattern 448
 expr_list 337
 EXPR_FILENAME 186
 EXPR_LINENO 186

EXPR_STMT	226
EXPR_STMT_EXPR	226
expression	191
expression codes	283
extended basic blocks, RTL SSA	339
extendmn2 instruction pattern	458
extensible constraints	387
EXTRA_SPECS	533
extract_last_m instruction pattern	439
extv instruction pattern	459
extvm instruction pattern	459
extvmisalignm instruction pattern	459
extzv instruction pattern	460
extzvm instruction pattern	459
extzvmisalignm instruction pattern	459

F

‘F’ in constraint	386
FAIL	487, 490, 498
fall-thru	351
false positive	785
FATAL_EXIT_CODE	727
FDL, GNU Free Documentation License	811
features, optional, in system conventions	538
feclearexceptm instruction pattern	448
fegetroundm instruction pattern	448
feraiseexceptm instruction pattern	448
ffs	315
ffsm2 instruction pattern	453
FIELD_DECL	186
file_end_indicate_exec_stack	654
files and passes of the compiler	145
files, generated	745
final_absence_set	512
final_presence_set	512
final_sequence	675
FINAL_PRESCAN_INSN	673
FIND_BASE_TERM	618
FINI_ARRAY_SECTION_ASM_OP	649
FINI_SECTION_ASM_OP	648
finite state automaton minimization	513
FIRST_PARM_OFFSET	575
FIRST_PARM_OFFSET and virtual registers	307
FIRST_PSEUDO_REGISTER	556
FIRST_STACK_REG	563
FIRST_VIRTUAL_REGISTER	307
fix	320
fix-it hints	791
fix_truncmn2 instruction pattern	458
FIX_TRUNC_EXPR	195
fixed register	556
fixed-point fractional library	20
fixed_regs	558
fixed_size_mode	300
FIXED_CONVERT_EXPR	195
FIXED_CST	192
FIXED_POINT_TYPE	182

FIXED_REGISTERS	556
fixmn2 instruction pattern	457
fixuns_truncmn2 instruction pattern	458
fixunsmn2 instruction pattern	458
flags in RTL expression	290
float	320
float_extend	319
float_truncate	320
FLOAT_EXPR	195
FLOAT_LIB_COMPARE_RETURNS_BOOL	615
FLOAT_STORE_FLAG_VALUE	706
FLOAT_WORDS_BIG_ENDIAN	541
FLOAT_WORDS_BIG_ENDIAN, (lack of) effect on subreg	310
floating point and cross compilation	685
floatmn2 instruction pattern	457
floatunsmn2 instruction pattern	457
FLOOR_DIV_EXPR	195
FLOOR_MOD_EXPR	195
floorm2 instruction pattern	450
flow-insensitive alias analysis	280
flow-sensitive alias analysis	280
fma	314
fmam4 instruction pattern	438
fmaxm3 instruction pattern	438
fminm3 instruction pattern	438
fmodm3 instruction pattern	447
fmsm4 instruction pattern	438
fnmam4 instruction pattern	438
fnmsm4 instruction pattern	438
fold_extract_last_m instruction pattern	439
fold_left_plus_m instruction pattern	439
for_user	739
FOR_BODY	226
FOR_COND	226
FOR_EXPR	226
FOR_INIT_STMT	226
FOR_STMT	226
force_reg	426
FORCE_CODE_SECTION_ALIGN	649
fract_convert	320
FRACT_TYPE_SIZE	552
fractional types	20
fractmn2 instruction pattern	458
fractunsmn2 instruction pattern	458
frame layout	574
frame_pointer_needed	602
frame_pointer_rtx	585
frame_related	293
frame_related, in insn, call_insn, jump_insn, barrier, and set	292
frame_related, in mem	291
frame_related, in reg	291
frame_related, in symbol_ref	292
FRAME_ADDR_RTX	576
FRAME_GROWS_DOWNWARD	574
FRAME_GROWS_DOWNWARD and virtual registers	307
FRAME_POINTER_CFA_OFFSET	578

FRAME_POINTER_REGNUM 583
 FRAME_POINTER_REGNUM and virtual registers... 307
 frequency, count, BB_FREQ_BASE..... 354
 ftruncm2 instruction pattern..... 458
 function 215, 223
 function entry and exit..... 601
 function entry point, alternate
 function entry point..... 353
 function properties..... 217
 function-call insns 337
 FUNCTION_ARG_REGNO_P 594
 FUNCTION_BOUNDARY..... 544
 FUNCTION_DECL..... 215, 223
 FUNCTION_MODE 707
 FUNCTION_PROFILER..... 605
 FUNCTION_TYPE 182
 FUNCTION_VALUE 598
 FUNCTION_VALUE_REGNO_P 599
 functions, leaf..... 562
 fundamental type..... 182
 fused multiply-add..... 314

G

‘g’ in constraint..... 386
 ‘G’ in constraint..... 386
 garbage collector, invocation 746
 garbage collector, troubleshooting..... 746
 gather_loadmn instruction pattern..... 430
 GCC and portability..... 3
 GCC_DRIVER_HOST_INITIALIZATION..... 727
 gcov_type..... 354
 ge 317
 ge and attributes 502
 GE_EXPR 195
 GEN_ERRNO_RTX 615
 gencodes 156
 general_operand..... 382
 GENERAL_REGS 563
 generated files 745
 generating assembler output 376
 generating insns 371
 generic predicates..... 381
 GENERIC..... 145, 179
 genflags 156
 get_attr..... 502
 get_attr_length..... 506
 get_insns..... 329
 get_last_insn 329
 get_thread_pointermode
 instruction pattern 479
 GET_CLASS_NARROWEST_MODE..... 302
 GET_CODE 283
 GET_MODE..... 301
 GET_MODE_ALIGNMENT..... 301
 GET_MODE_BITSIZE..... 301
 GET_MODE_CLASS 301
 GET_MODE_FBIT 301

GET_MODE_IBIT 301
 GET_MODE_INNER 301
 GET_MODE_MASK 301
 GET_MODE_NAME 301
 GET_MODE_NUNITS 301
 GET_MODE_SIZE 301
 GET_MODE_UNIT_SIZE..... 301
 GET_MODE_WIDER_MODE 301
 GET_RTX_CLASS 284
 GET_RTX_FORMAT 286
 GET_RTX_LENGTH 286
 geu..... 317
 geu and attributes..... 502
 ggc_collect..... 746
 GGC..... 737
 gimple..... 232
 gimple_addresses_taken..... 243
 gimple_asm_basic_p..... 245
 gimple_asm_clobber_op..... 245
 gimple_asm_input_op..... 245
 gimple_asm_nclobbers..... 245
 gimple_asm_ninputs..... 245
 gimple_asm_noutputs..... 245
 gimple_asm_output_op..... 245
 gimple_asm_set_basic..... 245
 gimple_asm_set_clobber_op..... 245
 gimple_asm_set_input_op..... 245
 gimple_asm_set_output_op..... 245
 gimple_asm_set_volatile..... 246
 gimple_asm_string..... 245
 gimple_asm_volatile_p..... 246
 gimple_assign_cast_p..... 241, 247
 gimple_assign_lhs..... 247
 gimple_assign_lhs_ptr..... 247
 gimple_assign_rhs_class..... 247
 gimple_assign_rhs_code..... 246
 gimple_assign_rhs1..... 247
 gimple_assign_rhs1_ptr..... 247
 gimple_assign_rhs2..... 247
 gimple_assign_rhs2_ptr..... 247
 gimple_assign_rhs3..... 247
 gimple_assign_rhs3_ptr..... 247
 gimple_assign_set_lhs..... 247
 gimple_assign_set_rhs1..... 247
 gimple_assign_set_rhs2..... 247
 gimple_assign_set_rhs3..... 247
 gimple_bb..... 242
 gimple_bind_add_seq..... 248
 gimple_bind_add_stmt..... 248
 gimple_bind_append_vars..... 248
 gimple_bind_block..... 248
 gimple_bind_body..... 248
 gimple_bind_set_block..... 248
 gimple_bind_set_body..... 248
 gimple_bind_set_vars..... 248
 gimple_bind_vars..... 247
 gimple_block..... 242
 gimple_build..... 765, 766

<code>gimple_build_debug_begin_stmt</code>	252	<code>gimple_debug_nonbind_marker_p</code>	242
<code>gimple_build_debug_inline_entry</code>	253	<code>gimple_def_ops</code>	243
<code>gimple_build_nop</code>	254	<code>gimple_eh_filter_failure</code>	253
<code>gimple_build_omp_master</code>	257	<code>gimple_eh_filter_set_failure</code>	253
<code>gimple_build_omp_ordered</code>	257	<code>gimple_eh_filter_set_types</code>	253
<code>gimple_build_omp_return</code>	258	<code>gimple_eh_filter_types</code>	253
<code>gimple_build_omp_section</code>	259	<code>gimple_eh_filter_types_ptr</code>	253
<code>gimple_build_omp_sections_switch</code>	259	<code>gimple_eh_must_not_throw_fndecl</code>	253
<code>gimple_build_omp_structured_block</code>	260	<code>gimple_eh_must_not_throw_set_fndecl</code>	253
<code>gimple_build_wce</code>	263	<code>gimple_expr_code</code>	242
<code>gimple_call_arg</code>	249	<code>gimple_goto_dest</code>	254
<code>gimple_call_arg_ptr</code>	249	<code>gimple_goto_set_dest</code>	254
<code>gimple_call_chain</code>	249	<code>gimple_has_mem_ops</code>	243
<code>gimple_call_copy_skip_args</code>	250	<code>gimple_has_ops</code>	243
<code>gimple_call_fn</code>	249	<code>gimple_has_volatile_ops</code>	244
<code>gimple_call_fndecl</code>	249	<code>gimple_label_label</code>	253
<code>gimple_call_lhs</code>	249	<code>gimple_label_set_label</code>	253
<code>gimple_call_lhs_ptr</code>	249	<code>gimple_loaded_syms</code>	244
<code>gimple_call_noreturn_p</code>	250	<code>gimple_locus</code>	242
<code>gimple_call_num_args</code>	249	<code>gimple_locus_empty_p</code>	242
<code>gimple_call_return_type</code>	249	<code>gimple_modified_p</code>	244
<code>gimple_call_set_arg</code>	249	<code>gimple_no_warning_p</code>	243
<code>gimple_call_set_chain</code>	249	<code>gimple_nop_p</code>	254
<code>gimple_call_set_fn</code>	249	<code>gimple_num_ops</code>	240, 243
<code>gimple_call_set_fndecl</code>	249	<code>gimple_omp_atomic_load_lhs</code>	254
<code>gimple_call_set_lhs</code>	249	<code>gimple_omp_atomic_load_rhs</code>	254
<code>gimple_call_set_tail</code>	249	<code>gimple_omp_atomic_load_set_lhs</code>	254
<code>gimple_call_tail_p</code>	249	<code>gimple_omp_atomic_load_set_rhs</code>	254
<code>gimple_catch_handler</code>	250	<code>gimple_omp_atomic_store_set_val</code>	254
<code>gimple_catch_set_handler</code>	250	<code>gimple_omp_atomic_store_val</code>	255
<code>gimple_catch_set_types</code>	250	<code>gimple_omp_body</code>	258
<code>gimple_catch_types</code>	250	<code>gimple_omp_continue_control_def</code>	255
<code>gimple_catch_types_ptr</code>	250	<code>gimple_omp_continue_control_def_ptr</code>	255
<code>gimple_code</code>	242	<code>gimple_omp_continue_control_use</code>	255
<code>gimple_cond_code</code>	250	<code>gimple_omp_continue_control_use_ptr</code>	255
<code>gimple_cond_false_label</code>	251	<code>gimple_omp_continue_set_control_def</code>	255
<code>gimple_cond_lhs</code>	251	<code>gimple_omp_continue_set_control_use</code>	255
<code>gimple_cond_make_false</code>	251	<code>gimple_omp_critical_name</code>	255
<code>gimple_cond_make_true</code>	251	<code>gimple_omp_critical_name_ptr</code>	255
<code>gimple_cond_rhs</code>	251	<code>gimple_omp_critical_set_name</code>	256
<code>gimple_cond_set_code</code>	250	<code>gimple_omp_for_clauses</code>	256
<code>gimple_cond_set_false_label</code>	251	<code>gimple_omp_for_clauses_ptr</code>	256
<code>gimple_cond_set_lhs</code>	251	<code>gimple_omp_for_cond</code>	257
<code>gimple_cond_set_rhs</code>	251	<code>gimple_omp_for_final</code>	256
<code>gimple_cond_set_true_label</code>	251	<code>gimple_omp_for_final_ptr</code>	256
<code>gimple_cond_true_label</code>	251	<code>gimple_omp_for_incr</code>	257
<code>gimple_convert</code>	766	<code>gimple_omp_for_incr_ptr</code>	257
<code>gimple_copy</code>	244	<code>gimple_omp_for_index</code>	256
<code>gimple_debug_begin_stmt_p</code>	242	<code>gimple_omp_for_index_ptr</code>	256
<code>gimple_debug_bind_get_value</code>	252	<code>gimple_omp_for_initial</code>	256
<code>gimple_debug_bind_get_value_ptr</code>	252	<code>gimple_omp_for_initial_ptr</code>	256
<code>gimple_debug_bind_get_var</code>	252	<code>gimple_omp_for_pre_body</code>	257
<code>gimple_debug_bind_has_value_p</code>	252	<code>gimple_omp_for_set_clauses</code>	256
<code>gimple_debug_bind_p</code>	242	<code>gimple_omp_for_set_cond</code>	257
<code>gimple_debug_bind_reset_value</code>	252	<code>gimple_omp_for_set_final</code>	257
<code>gimple_debug_bind_set_value</code>	252	<code>gimple_omp_for_set_incr</code>	257
<code>gimple_debug_bind_set_var</code>	252	<code>gimple_omp_for_set_index</code>	256
<code>gimple_debug_inline_entry_p</code>	242	<code>gimple_omp_for_set_initial</code>	256

<code>gimple_omp_for_set_pre_body</code>	257	<code>gimple_set_has_volatile_ops</code>	244
<code>gimple_omp_parallel_child_fn</code>	258	<code>gimple_set_locus</code>	242
<code>gimple_omp_parallel_child_fn_ptr</code>	258	<code>gimple_set_op</code>	243
<code>gimple_omp_parallel_clauses</code>	258	<code>gimple_set_plf</code>	243
<code>gimple_omp_parallel_clauses_ptr</code>	258	<code>gimple_set_use_ops</code>	244
<code>gimple_omp_parallel_combined_p</code>	257	<code>gimple_set_vdef_ops</code>	244
<code>gimple_omp_parallel_data_arg</code>	258	<code>gimple_set_visited</code>	243
<code>gimple_omp_parallel_data_arg_ptr</code>	258	<code>gimple_set_vuse_ops</code>	244
<code>gimple_omp_parallel_set_child_fn</code>	258	<code>gimple_simplify</code>	765
<code>gimple_omp_parallel_set_clauses</code>	258	<code>gimple_statement_with_ops</code>	233
<code>gimple_omp_parallel_set_combined_p</code>	258	<code>gimple_stored_syms</code>	244
<code>gimple_omp_parallel_set_data_arg</code>	258	<code>gimple_switch_default_label</code>	262
<code>gimple_omp_return_nowait_p</code>	259	<code>gimple_switch_index</code>	261
<code>gimple_omp_return_set_nowait</code>	259	<code>gimple_switch_label</code>	262
<code>gimple_omp_section_last_p</code>	259	<code>gimple_switch_num_labels</code>	261
<code>gimple_omp_section_set_last</code>	259	<code>gimple_switch_set_default_label</code>	262
<code>gimple_omp_sections_clauses</code>	259	<code>gimple_switch_set_index</code>	262
<code>gimple_omp_sections_clauses_ptr</code>	259	<code>gimple_switch_set_label</code>	262
<code>gimple_omp_sections_control</code>	259	<code>gimple_switch_set_num_labels</code>	261
<code>gimple_omp_sections_control_ptr</code>	259	<code>gimple_try_catch_is_cleanup</code>	262
<code>gimple_omp_sections_set_clauses</code>	259	<code>gimple_try_cleanup</code>	262
<code>gimple_omp_sections_set_control</code>	259	<code>gimple_try_eval</code>	262
<code>gimple_omp_set_body</code>	258	<code>gimple_try_kind</code>	262
<code>gimple_omp_single_clauses</code>	260	<code>gimple_try_set_catch_is_cleanup</code>	262
<code>gimple_omp_single_clauses_ptr</code>	260	<code>gimple_try_set_cleanup</code>	263
<code>gimple_omp_single_set_clauses</code>	260	<code>gimple_try_set_eval</code>	262
<code>gimple_op</code>	240, 243	<code>gimple_use_ops</code>	243
<code>gimple_op_ptr</code>	243	<code>gimple_vdef_ops</code>	244
<code>gimple_ops</code>	240, 243	<code>gimple_visited_p</code>	243
<code>gimple_phi_arg</code>	260, 277	<code>gimple_vuse_ops</code>	244
<code>gimple_phi_arg_def</code>	277	<code>gimple_wce_cleanup</code>	263
<code>gimple_phi_arg_edge</code>	277	<code>gimple_wce_cleanup_eh_only</code>	263
<code>gimple_phi_capacity</code>	260	<code>gimple_wce_set_cleanup</code>	263
<code>gimple_phi_num_args</code>	260, 277	<code>gimple_wce_set_cleanup_eh_only</code>	263
<code>gimple_phi_result</code>	260, 277	<code>GIMPLE</code>	145, 146, 231
<code>gimple_phi_result_ptr</code>	260	<code>GIMPLE API</code>	765
<code>gimple_phi_set_arg</code>	261	<code>GIMPLE class hierarchy</code>	234
<code>gimple_phi_set_result</code>	260	<code>GIMPLE Exception Handling</code>	237
<code>gimple_plf</code>	243	<code>GIMPLE instruction set</code>	237
<code>gimple_resx_region</code>	261	<code>GIMPLE sequences</code>	263
<code>gimple_resx_set_region</code>	261	<code>GIMPLE statement iterators</code>	350, 355
<code>gimple_return_retval</code>	261	<code>GIMPLE_ASM</code>	245
<code>gimple_return_set_retval</code>	261	<code>GIMPLE_ASSIGN</code>	246
<code>gimple_seq_add_seq</code>	263	<code>GIMPLE_BIND</code>	247
<code>gimple_seq_add_stmt</code>	263	<code>GIMPLE_CALL</code>	248
<code>gimple_seq_alloc</code>	264	<code>GIMPLE_CATCH</code>	250
<code>gimple_seq_copy</code>	264	<code>GIMPLE_COND</code>	250
<code>gimple_seq_deep_copy</code>	264	<code>GIMPLE_DEBUG</code>	251
<code>gimple_seq_empty_p</code>	264	<code>GIMPLE_DEBUG_BEGIN_STMT</code>	251
<code>gimple_seq_first</code>	264	<code>GIMPLE_DEBUG_BIND</code>	251
<code>gimple_seq_init</code>	264	<code>GIMPLE_DEBUG_INLINE_ENTRY</code>	251
<code>gimple_seq_last</code>	264	<code>GIMPLE_EH_FILTER</code>	253
<code>gimple_seq_reverse</code>	264	<code>GIMPLE_GOTO</code>	254
<code>gimple_seq_set_first</code>	264	<code>GIMPLE_LABEL</code>	253
<code>gimple_seq_set_last</code>	264	<code>GIMPLE_NOP</code>	254
<code>gimple_seq_singleton_p</code>	264	<code>GIMPLE_OMP_ATOMIC_LOAD</code>	254
<code>gimple_set_block</code>	242	<code>GIMPLE_OMP_ATOMIC_STORE</code>	254
<code>gimple_set_def_ops</code>	243	<code>GIMPLE_OMP_CONTINUE</code>	255

GIMPLE_OMP_CRITICAL	255
GIMPLE_OMP_FOR	256
GIMPLE_OMP_MASTER	257
GIMPLE_OMP_ORDERED	257
GIMPLE_OMP_PARALLEL	257
GIMPLE_OMP_RETURN	258
GIMPLE_OMP_SECTION	259
GIMPLE_OMP_SECTIONS	259
GIMPLE_OMP_SINGLE	260
GIMPLE_OMP_STRUCTURED_BLOCK	260
GIMPLE_PHI	260
GIMPLE_RESX	261
GIMPLE_RETURN	261
GIMPLE_SWITCH	261
GIMPLE_TRY	262
GIMPLE_WITH_CLEANUP_EXPR	263
gimplification	145, 146
gimplifier	145
simplify_assign	246
simplify_expr	146
simplify_function_tree	146
global_regs	558
GLOBAL_INIT_PRIORITY	226
GO_IF_LEGITIMATE_ADDRESS	618
greater than	317
gsi_after_labels	265
gsi_bb	266
gsi_commit_edge_inserts	268, 356
gsi_commit_one_edge_insert	268
gsi_end_p	265, 356
gsi_for_stmt	267
gsi_insert_after	267, 356
gsi_insert_before	267, 356
gsi_insert_on_edge	267, 356
gsi_insert_on_edge_immediate	267
gsi_insert_seq_after	267
gsi_insert_seq_before	267
gsi_insert_seq_on_edge	267
gsi_last	265, 356
gsi_last_bb	265
gsi_link_after	266
gsi_link_before	266
gsi_link_seq_after	266
gsi_link_seq_before	266
gsi_move_after	267
gsi_move_before	267
gsi_move_to_bb_end	267
gsi_next	265, 356
gsi_one_before_end_p	265
gsi_prev	265, 356
gsi_remove	266, 356
gsi_replace	266
gsi_seq	266
gsi_split_seq_after	266
gsi_split_seq_before	266
gsi_start	265, 356
gsi_start_bb	265
gsi_stmt	265

gsi_stmt_ptr	266
gt	317
gt and attributes	502
GT_EXPR	195
gtu	317
gtu and attributes	502
GTY	737
guidelines for diagnostics	785
guidelines for options	793
guidelines, user experience	785

H

‘H’ in constraint	386
HAmode	297
HANDLE_PRAGMA_PACK_WITH_EXPANSION	708
HANDLER	226
HANDLER_BODY	226
HANDLER_PARMS	226
hard registers	306
hard registers in constraint	386
HARD_FRAME_POINTER_IS_ARG_POINTER	584
HARD_FRAME_POINTER_IS_FRAME_POINTER	584
HARD_FRAME_POINTER_REGNUM	583
HARD_REGNO_CALLER_SAVE_MODE	601
HARD_REGNO_NREGS_HAS_PADDING	560
HARD_REGNO_NREGS_WITH_PADDING	560
HARD_REGNO_RENAME_OK	561
HAS_INIT_SECTION	670
HAS_LONG_COND_BRANCH	701
HAS_LONG_UNCOND_BRANCH	701
HAVE_DOS_BASED_FILE_SYSTEM	726
HAVE_POST_DECREMENT	616
HAVE_POST_INCREMENT	616
HAVE_POST_MODIFY_DISP	616
HAVE_POST_MODIFY_REG	617
HAVE_PRE_DECREMENT	616
HAVE_PRE_INCREMENT	616
HAVE_PRE_MODIFY_DISP	616
HAVE_PRE_MODIFY_REG	617
HCmode	298
HFmode	296
high	306
high-part multiplication	314
HImode	296
HImode, in insn	333
HONOR_REG_ALLOC_ORDER	559
host configuration	725
host functions	725
host hooks	725
host makefile fragment	732
HOST_BIT_BUCKET	726
HOST_EXECUTABLE_SUFFIX	726
HOST_HOOKS_EXTRA_SIGNALS	725
HOST_HOOKS_GT_PCH_ALLOC_GRANULARITY	725
HOST_HOOKS_GT_PCH_GET_ADDRESS	725
HOST_HOOKS_GT_PCH_USE_ADDRESS	725
HOST_LACKS_INODE_NUMBERS	727

HOST_LONG_FORMAT 728
 HOST_LONG_LONG_FORMAT 728
 HOST_OBJECT_SUFFIX 726
 HOST_PTR_PRINTF 728
 HOT_TEXT_SECTION_NAME 647
 HQmode 297

I

'i' in constraint 386
 'I' in constraint 386
 identifier 181
 IDENTIFIER_LENGTH 181
 IDENTIFIER_NODE 181
 IDENTIFIER_OPNAME_P 181
 IDENTIFIER_POINTER 181
 IDENTIFIER_TYPENAME_P 181
 IEEE 754-2008 14
 if_then_else 317
 if_then_else and attributes 501
 if_then_else usage 321
 IF_COND 226
 IF_STMT 226
 IFCVT_MACHDEP_INIT 711
 IFCVT_MODIFY_CANCEL 710
 IFCVT_MODIFY_FINAL 710
 IFCVT_MODIFY_INSN 710
 IFCVT_MODIFY_MULTIPLE_TESTS 710
 IFCVT_MODIFY_TESTS 710
 IFN_VEC_TRUNC_ADD_HIGH 203
 IFN_VEC_WIDEN_MINUS 203
 IFN_VEC_WIDEN_MINUS_EVEN 203
 IFN_VEC_WIDEN_MINUS_HI 203
 IFN_VEC_WIDEN_MINUS_LO 203
 IFN_VEC_WIDEN_MINUS_ODD 203
 IFN_VEC_WIDEN_PLUS 203
 IFN_VEC_WIDEN_PLUS_EVEN 203
 IFN_VEC_WIDEN_PLUS_HI 203
 IFN_VEC_WIDEN_PLUS_LO 203
 IFN_VEC_WIDEN_PLUS_ODD 203
 IMAGPART_EXPR 195
 Immediate Uses 275
 immediate_operand 381
 IMMEDIATE_PREFIX 675
 in_struct 294
 in_struct, in code_label and note 290
 in_struct, in insn and jump_insn
 and call_insn 290
 in_struct, in insn, call_insn, jump_insn
 and jump_table_data 292
 in_struct, in subreg 293
 include 494
 INCLUDE_DEFAULTS 535
 inclusive-or, bitwise 315
 INCOMING_FRAME_SP_OFFSET 577
 INCOMING_REG_PARM_STACK_SPACE 588
 INCOMING_REGNO 558
 INCOMING_RETURN_ADDR_RTX 576

INCOMING_STACK_BOUNDARY 544
 INDEX_REG_CLASS 565
 indirect_jump instruction pattern 468
 indirect_operand 382
 INDIRECT_REF 194
 init_machine_status 541
 init_one_libfunc 614
 INIT_ARRAY_SECTION_ASM_OP 649
 INIT_CUMULATIVE_ARGS 592
 INIT_CUMULATIVE_INCOMING_ARGS 592
 INIT_CUMULATIVE_LIBCALL_ARGS 592
 INIT_ENVIRONMENT 535
 INIT_EXPANDERS 541
 INIT_EXPR 195
 INIT_SECTION_ASM_OP 648, 670
 INITIAL_ELIMINATION_OFFSET 587
 INITIAL_FRAME_ADDRESS_RTX 575
 initialization routines 668
 inlining 691
 insert_insn_on_edge 356
 insn 329
 insn and '/f' 292
 insn and '/j' 292
 insn and '/s' 290, 292
 insn and '/u' 290
 insn and '/v' 290
 insn attributes 499
 insn canonicalization 484
 insn includes 494
 insn lengths, computing 505
 insn notes, notes 350
 insn splitting 489
 insn-attr.h 499
 insn_list 337
 INSN_ANNULLED_BRANCH_P 290
 INSN_BASE_REG_CLASS 565
 INSN_CODE 333
 INSN_DELETED_P 290
 INSN_FROM_TARGET_P 290
 INSN_INDEX_REG_CLASS 565
 INSN_REFERENCES_ARE_DELAYED 709
 INSN_SETS_ARE_DELAYED 709
 INSN_UID 328
 INSN_VAR_LOCATION 332
 insns 328
 insns, generating 371
 insns, recognizing 371
 instruction attributes 499
 instruction latency time 508, 510, 511
 instruction patterns 369
 instruction splitting 489
 instructions, RTL SSA 339
 insv instruction pattern 460
 insvm instruction pattern 459
 insvmisalignm instruction pattern 459
 int iterators in .md files 524
 INT_FAST16_TYPE 554
 INT_FAST32_TYPE 554

INT_FAST64_TYPE	554
INT_FAST8_TYPE	554
INT_LEAST16_TYPE	554
INT_LEAST32_TYPE	554
INT_LEAST64_TYPE	554
INT_LEAST8_TYPE	554
INT_TYPE_SIZE	551
INT16_TYPE	554
INT32_TYPE	554
INT64_TYPE	554
INT8_TYPE	554
INTEGER_CST	192
INTEGER_TYPE	182
inter-procedural optimization passes	147
Interdependence of Patterns	481
interlock delays	508
intermediate representation lowering	145
INTMAX_TYPE	554
INTPTR_TYPE	555
INVOKE_main	671
ior	315
ior and attributes	501
ior, canonicalization of	485
iorm3 instruction pattern	436
iornm3 instruction pattern	437
IPA passes	147
IRA_HARD_REGNO_ADD_COST_MULTIPLIER	559
is_a	300
is_gimple_addressable	241
is_gimple_asm_val	241
is_gimple_assign	241
is_gimple_call	241
is_gimple_call_addr	241
is_gimple_constant	241
is_gimple_debug	241
is_gimple_ip_invariant	241
is_gimple_ip_invariant_address	241
is_gimple_mem_ref_addr	241
is_gimple_min_invariant	241
is_gimple_omp	242
is_gimple_val	241
IS_ASM_LOGICAL_LINE_SEPARATOR	658
isfinitem2 instruction pattern	480
isnannm2 instruction pattern	480
isnormalm2 instruction pattern	480
issignalingm2 instruction pattern	451
iterators in .md files	520
IV analysis on GIMPLE	363
IV analysis on RTL	364

J

JMP_BUF_SIZE	679
jump	294
jump instruction pattern	466
jump instruction patterns	482
jump instructions and set	321
jump, in call_insn	292
jump, in insn	292
jump, in mem	290
jump_insn	329
jump_insn and '/f'	292
jump_insn and '/j'	290
jump_insn and '/s'	290, 292
jump_insn and '/u'	290
jump_insn and '/v'	290
jump_table_data	331
jump_table_data and '/s'	292
jump_table_data and '/v'	290
JUMP_ALIGN	680
JUMP_LABEL	329
JUMP_TABLES_IN_TEXT_SECTION	649
Jumps	209

L

label_ref	305
label_ref and '/v'	290
label_ref, RTL sharing	347
LABEL_ALIGN	681
LABEL_ALIGN_AFTER_BARRIER	681
LABEL_ALT_ENTRY_P	330
LABEL_ALTERNATE_NAME	353
LABEL_DECL	186
LABEL_KIND	330
LABEL_NUSES	330
LABEL_PRESERVE_P	290
LABEL_REF_NONLOCAL_P	290
lang_hooks.gimplify_expr	146
lang_hooks.parse_file	145
language-dependent trees	218
language-independent intermediate representation	145
large return values	600
LAST_STACK_REG	563
LAST_VIRTUAL_REGISTER	307
late IPA passes	150
lceil _m n2	451
LCSSA	362
LD_FINI_SWITCH	670
LD_INIT_SWITCH	670
LDD_SUFFIX	672
ldexp _m 3 instruction pattern	447
le	317
le and attributes	502
LE_EXPR	195
leaf functions	562
leaf_function_p	467
LEAF_REG_REMAP	562

LEAF_REGISTERS	562
left rotate	315
left shift	315
LEGITIMATE_PIC_OPERAND_P	653
LEGITIMIZE_RELOAD_ADDRESS	619
len_fold_extract_last_m instruction pattern	439
len_load_m instruction pattern	434
len_store_m instruction pattern	435
length	738
less than	317
less than or equal	317
leu	317
leu and attributes	502
lfloormn2	451
LIB_SPEC	532
LIB2FUNCS_EXTRA	729
LIBC_CPP_SPEC	531
LIBC_LINK_SPEC	532
LIBCALL_VALUE	598
libgcc.a	614
LIBGCC_SPEC	532
LIBGCC2_CFLAGS	729
LIBGCC2_GNU_PREFIX	552
LIBGCC2_UNWIND_ATTRIBUTE	719
library subroutine names	614
LIBRARY_PATH_ENV	710
LIMIT_RELOAD_CLASS	568
LINK_COMMAND_SPEC	534
LINK_EH_SPEC	532
LINK_GCC_C_SEQUENCE_SPEC	534
LINK_LIBGCC_SPECIAL_1	534
LINK_SPEC	532
list	181
Liveness representation	357
lo_sum	312
load address instruction	387
load_multiple instruction pattern	428
LOAD_EXTEND_OP	702
Local Register Allocator (LRA)	159
LOCAL_ALIGNMENT	546
LOCAL_CLASS_P	223
LOCAL_DECL_ALIGNMENT	547
LOCAL_INCLUDE_DIR	535
LOCAL_LABEL_PREFIX	675
LOCAL_REGNO	558
location information	787
log10m2 instruction pattern	449
log1pm2 instruction pattern	449
log2m2 instruction pattern	449
logbm2 instruction pattern	449
Logical Operators	239
logical-and, bitwise	314
LOGICAL_OP_NON_SHORT_CIRCUIT	637
logm2 instruction pattern	449
LONG_ACCUM_TYPE_SIZE	552
LONG_FRACT_TYPE_SIZE	552
LONG_LONG_ACCUM_TYPE_SIZE	552
LONG_LONG_FRACT_TYPE_SIZE	552
LONG_LONG_TYPE_SIZE	552
LONG_TYPE_SIZE	551
Loop analysis	359
Loop manipulation	362
Loop querying	361
Loop representation	359
Loop-closed SSA form	362
LOOP_ALIGN	681
LOOP_EXPR	195
looping instruction patterns	482
lowering, language-dependent intermediate representation	145
lrintmn2	450
LROTATE_EXPR	195
lroundmn2	450
LSHIFT_EXPR	195
lshiftrt	315
lshiftrt and attributes	502
lshrm3 instruction pattern	446
lt	317
lt and attributes	502
LT_EXPR	195
LTGT_EXPR	195
lto	757
ltrans	757
ltu	317

M

'm' in constraint	385
MACH_DEP_SECTION_ASM_FLAG	649
machine attributes	688
machine description macros	529
machine descriptions	369
machine mode conversions	319
machine mode wrapper classes	299
machine modes	295
machine specific constraints	392
machine-independent predicates	381
machine_mode	295
macros, target description	529
maddmn4 instruction pattern	445
make_safe_from	488
MAKE_DECL_ONE_ONLY	665
makefile fragment	729
makefile targets	68
MALLOC_ABI_ALIGNMENT	544
Manipulating GIMPLE statements	242
marking roots	745
mask_fold_left_plus_m instruction pattern	439
mask_gather_loadmn instruction pattern	430
mask_len_fold_left_plus_m instruction pattern	439
mask_len_gather_loadmn instruction pattern	430
mask_len_loadmn instruction pattern	435

<code>mask_len_scatter_storemn</code>		<code>MEM_KEEP_ALIAS_SET_P</code>	290
instruction pattern	431	<code>MEM_NOTRAP_P</code>	291
<code>mask_len_storemn</code> instruction pattern	436	<code>MEM_OFFSET</code>	288
<code>mask_len_strided_loadm</code>		<code>MEM_OFFSET_KNOWN_P</code>	288
instruction pattern	431	<code>MEM_POINTER</code>	291
<code>mask_len_strided_storem</code>		<code>MEM_READONLY_P</code>	291
instruction pattern	431	<code>MEM_REF</code>	194
<code>mask_scatter_storemn</code> instruction pattern	431	<code>MEM_SIZE</code>	288
<code>MASK_RETURN_ADDR</code>	678	<code>MEM_SIZE_KNOWN_P</code>	288
<code>maskloadmn</code> instruction pattern	434	<code>MEM_VOLATILE_P</code>	291
<code>maskstoremn</code> instruction pattern	434	memory model	281
Match and Simplify	765	memory reference, nonoffsettable	389
<code>match_dup</code>	372, 498	memory references in constraints	385
<code>match_dup</code> and attributes	506	<code>memory_barrier</code> instruction pattern	473
<code>match_op_dup</code>	374	<code>memory_blockage</code> instruction pattern	473
<code>match_operand</code>	372	<code>memory_operand</code>	382
<code>match_operand</code> and attributes	501	<code>MEMORY_MOVE_COST</code>	632
<code>match_operator</code>	373	<code>METHOD_TYPE</code>	182
<code>match_par_dup</code>	375	<code>MIN_UNITS_PER_WORD</code>	542
<code>match_parallel</code>	374	<code>MINIMUM_ALIGNMENT</code>	547
<code>match_scratch</code>	372, 498	<code>MINIMUM_ATOMIC_ALIGNMENT</code>	545
<code>match_test</code> and attributes	502	<code>minm3</code> instruction pattern	438
matching constraint	387	<code>minus</code>	312
matching operands	376	<code>minus</code> and attributes	502
math library	9	<code>minus</code> , canonicalization of	484
math, in RTL	312	<code>MINUS_EXPR</code>	195
<code>MATH_LIBRARY</code>	710	MIPS coprocessor-definition macros	695
<code>matherr</code>	615	miscellaneous register hooks	608
<code>MAX_BITS_PER_WORD</code>	542	mnemonic attribute	507
<code>MAX_BITSIZE_MODE_ANY_INT</code>	302	<code>mod</code>	314
<code>MAX_BITSIZE_MODE_ANY_MODE</code>	302	<code>mod</code> and attributes	502
<code>MAX_CONDITIONAL_EXECUTE</code>	710	mode classes	298
<code>MAX_FIXED_MODE_SIZE</code>	549	mode iterators in .md files	520
<code>MAX_MOVE_MAX</code>	703	mode switching	686
<code>MAX_OFILE_ALIGNMENT</code>	545	<code>MODE_ACCUM</code>	299
<code>MAX_REGS_PER_ADDRESS</code>	617	<code>MODE_BASE_REG_CLASS</code>	565
<code>MAX_STACK_ALIGNMENT</code>	545	<code>MODE_BASE_REG_REG_CLASS</code>	565
<code>maxm3</code> instruction pattern	438	<code>MODE_CC</code>	299, 629
<code>may_trap_p, tree_could_trap_p</code>	352	<code>MODE_CODE_BASE_REG_CLASS</code>	565
<code>maybe_undef</code>	741	<code>MODE_COMPLEX_FLOAT</code>	299
<code>mcount</code>	605	<code>MODE_COMPLEX_INT</code>	299
<code>MD_EXEC_PREFIX</code>	534	<code>MODE_DECIMAL_FLOAT</code>	298
<code>MD_FALLBACK_FRAME_STATE_FOR</code>	581	<code>MODE_FLOAT</code>	298
<code>MD_HANDLE_UNWABI</code>	581	<code>MODE_FRACT</code>	298
<code>MD_STARTFILE_PREFIX</code>	535	<code>MODE_INT</code>	298
<code>MD_STARTFILE_PREFIX_1</code>	535	<code>MODE_OPAQUE</code>	299
<code>mem</code>	312	<code>MODE_PARTIAL_INT</code>	298
<code>mem</code> and <code>'/c'</code>	291	<code>MODE_POINTER_BOUNDS</code>	299
<code>mem</code> and <code>'/f'</code>	291	<code>MODE_RANDOM</code>	299
<code>mem</code> and <code>'/j'</code>	290	<code>MODE_UACCUM</code>	299
<code>mem</code> and <code>'/u'</code>	291	<code>MODE_UFRACT</code>	299
<code>mem</code> and <code>'/v'</code>	291	modifiers in constraints	390
<code>mem</code> , RTL sharing	347	<code>MODIFY_EXPR</code>	195
<code>mem_thread_fence</code> instruction pattern	478	<code>modm3</code> instruction pattern	436
<code>MEM_ADDR_SPACE</code>	288	modulo scheduling	158
<code>MEM_ALIAS_SET</code>	287	<code>MOVE_MAX</code>	703
<code>MEM_ALIGN</code>	288	<code>MOVE_MAX_PIECES</code>	635
<code>MEM_EXPR</code>	287	<code>MOVE_RATIO</code>	634

<i>movm</i> instruction pattern	426	<i>negvm3</i> instruction pattern	447
<i>movmemm</i> instruction pattern	455	nested functions, support for	611
<i>movmisalignm</i> instruction pattern	428	<i>nested_ptr</i>	741
<i>movmodecc</i> instruction pattern	460	<i>next_bb</i> , <i>prev_bb</i> ,	
<i>movstr</i> instruction pattern	456	<i>FOR_EACH_BB</i> , <i>FOR_ALL_BB</i>	349
<i>movstrictm</i> instruction pattern	427	<i>NEXT_INSN</i>	329
<i>msubmn4</i> instruction pattern	445	<i>NEXT_OBJC_RUNTIME</i>	616
<i>mulhi3</i> instruction pattern	444	<i>nil</i>	284
<i>mulm3</i> instruction pattern	436	<i>NM_FLAGS</i>	672
<i>mulqihi3</i> instruction pattern	444	<i>NO_DOLLAR_IN_LABEL</i>	662
<i>mulsi3</i> instruction pattern	444	<i>NO_DOT_IN_LABEL</i>	662
<i>mult</i>	313	<i>NO_FUNCTION_CSE</i>	636
<i>mult</i> and attributes	502	<i>NO_PROFILE_COUNTERS</i>	605
<i>mult</i> , canonicalization of	484, 485	<i>NO_REGS</i>	563
<i>MULT_EXPR</i>	195	<i>NON_LVALUE_EXPR</i>	195
<i>MULT_HIGHPART_EXPR</i>	195	nondeterministic finite state automaton	513
<i>MULTIARCH_DIRNAME</i>	732	<i>nonimmediate_operand</i>	382
<i>MULTILIB_DEFAULTS</i>	534	<i>nonlocal_goto</i> handler	353
<i>MULTILIB_DIRNAMES</i>	730	<i>nonlocal_goto</i> instruction pattern	470
<i>MULTILIB_EXCEPTIONS</i>	730	<i>nonlocal_goto_receiver</i>	
<i>MULTILIB_EXTRA_OPTS</i>	731	instruction pattern	471
<i>MULTILIB_MATCHES</i>	730	<i>nonmemory_operand</i>	382
<i>MULTILIB_OPTIONS</i>	729	<i>nonoffsettable</i> memory reference	389
<i>MULTILIB_OSDIRNAMES</i>	731	<i>nop</i> instruction pattern	468
<i>MULTILIB_REQUIRED</i>	730	<i>NOP_EXPR</i>	195
<i>MULTILIB_REUSE</i>	731	<i>normal</i> predicates	380
multiple alternative constraints	389	<i>not</i>	314
<i>MULTIPLE_SYMBOL_SPACES</i>	709	<i>not</i> and attributes	501
multiplication	313	<i>not</i> equal	317
multiplication high part	314	<i>not</i> , canonicalization of	484
multiplication with signed saturation	313	<i>note</i>	331
multiplication with unsigned saturation	313	<i>note</i> and <i>‘/i’</i>	290
<i>mulvm4</i> instruction pattern	437	<i>note</i> and <i>‘/v’</i>	290
N			
<i>‘n’</i> in constraint	386	<i>NOTE_INSN_BASIC_BLOCK</i>	350
<i>N_REG_CLASSES</i>	564	<i>NOTE_INSN_BEGIN_STMT</i>	332
<i>name</i>	181	<i>NOTE_INSN_BLOCK_BEG</i>	331
named address spaces	699	<i>NOTE_INSN_BLOCK_END</i>	331
named patterns and conditions	370	<i>NOTE_INSN_DELETED</i>	331
<i>names</i> , pattern	426	<i>NOTE_INSN_DELETED_LABEL</i>	331
<i>namespace</i> , scope	221	<i>NOTE_INSN_EH_REGION_BEG</i>	331
<i>NAMESPACE_DECL</i>	186, 221	<i>NOTE_INSN_EH_REGION_END</i>	331
<i>NATIVE_SYSTEM_HEADER_COMPONENT</i>	535	<i>NOTE_INSN_FUNCTION_BEG</i>	331
<i>ne</i>	317	<i>NOTE_INSN_INLINE_ENTRY</i>	332
<i>ne</i> and attributes	502	<i>NOTE_INSN_VAR_LOCATION</i>	332
<i>NE_EXPR</i>	195	<i>NOTE_LINE_NUMBER</i>	331
<i>nearbyintm2</i> instruction pattern	450	<i>NOTE_SOURCE_FILE</i>	331
<i>neg</i>	313	<i>NOTE_VAR_LOCATION</i>	332
<i>neg</i> and attributes	502	<i>notmodecc</i> instruction pattern	464
<i>neg</i> , canonicalization of	484	<i>NUM_MACHINE_MODES</i>	301
<i>NEGATE_EXPR</i>	195	<i>NUM_MODES_FOR_MODE_SWITCHING</i>	687
<i>negation</i>	313	<i>NUM_POLY_INT_COEFFS</i>	165
<i>negation</i> with signed saturation	313	Number of iterations analysis	364
<i>negation</i> with unsigned saturation	313		
<i>negm2</i> instruction pattern	447		
<i>negmodecc</i> instruction pattern	464		

O

'o' in constraint	385
OACC_CACHE	214
OACC_DATA	214
OACC_DECLARE	214
OACC_ENTER_DATA	214
OACC_EXIT_DATA	214
OACC_HOST_DATA	214
OACC_KERNELS	214
OACC_LOOP	214
OACC_PARALLEL	214
OACC_SERIAL	214
OACC_UPDATE	214
OBJC_GEN_METHOD_LABEL	668
OBJC_JBLN	719
OBJECT_FORMAT_COFF	672
OFFSET_TYPE	182
offsettable address	385
OImode	296
OMP_ATOMIC	210
OMP_CLAUSE	210
OMP_CONTINUE	210
OMP_CRITICAL	210
OMP_DISTRIBUTE	210
OMP_FOR	210
OMP_LOOP	210
OMP_MASTER	210
OMP_METADIRECTIVE	210
OMP_NEXT_VARIANT	210
OMP_ORDERED	210
OMP_PARALLEL	210
OMP_RETURN	210
OMP_SECTION	210
OMP_SECTIONS	210
OMP_SIMD	210
OMP_SINGLE	210
OMP_TARGET_DEVICE_MATCHES	210
OMP_TASKLOOP	210
one_cmplm2 instruction pattern	455
OPAQUE_TYPE	182
OpenMP and OpenACC	625
operand access	286
Operand Access Routines	273
operand constraints	385
Operand Iterators	273
operand predicates	380
operand substitution	375
Operands	238
operands	271, 370
operator predicates	380
opt_mode	300
'optc-gen.awk'	135
OPTGROUP_ALL	161
OPTGROUP_INLINE	161
OPTGROUP_IPA	161
OPTGROUP_LOOP	161
OPTGROUP_OMP	161
OPTGROUP_OTHER	161

OPTGROUP_VEC	161
optimization dumps	160
optimization groups	161
optimization info file names	161
Optimization infrastructure for GIMPLE	271
OPTIMIZE_MODE_SWITCHING	686
option specification files	135
OPTION_DEFAULT_SPECS	530
optional hardware or system features	538
options, directory search	494
options, guidelines for	793
order of register allocation	559
ordered_comparison_operator	382
ORDERED_EXPR	195
Ordering of Patterns	481
ORIGINAL_REGNO	288
other register constraints	387
outgoing_args_size	588
OUTGOING_REG_PARM_STACK_SPACE	588
OUTGOING_REGNO	558
output of assembler code	653
output statements	376
output templates	375
output_asm_insn	377
OUTPUT_QUOTED_STRING	655
OVERLAPPING_REGISTER_NAMES	673
OVERLOAD	223
OVERRIDE_ABI_FORMAT	592
OVL_CURRENT	223
OVL_NEXT	223

P

'p' in constraint	387
PAD_VARARGS_DOWN	593
parallel	324
parameters, c++ abi	696
parameters, d abi	697
parameters, jit abi	699
parameters, miscellaneous	701
parameters, precompiled headers	695
parameters, rust abi	698
parity	316
paritym2 instruction pattern	455
PARM_BOUNDARY	544
PARM_DECL	186
PARSE_LDD_OUTPUT	672
pass dumps	145
pass_duplicate_computed_gotos	353
passes and files of the compiler	145
PATH_SEPARATOR	726
pattern conditions	370
pattern names	426
Pattern Ordering	481
patterns	369
PATTERN	333
pc	311
pc and attributes	506

pc, RTL sharing	347	pre_modify	327
pc_rtx	311	PRE_GCC3_DWARF_FRAME_REGISTERS	585
PC_REGNUM	558	PREDECREMENT_EXPR	195
PCC_BITFIELD_TYPE_MATTERS	547	predefined macros	537
PCC_STATIC_STRUCT_RETURN	601	predicates	380
PDImode	296	predicates and machine modes	380
peephole optimization, RTL representation	324	predication	514
peephole optimizer definitions	495	predict.def	354
per-function data	540	PREFERRED_DEBUGGING_TYPE	682
percent sign	375	PREFERRED_RELOAD_CLASS	567
phi nodes, RTL SSA	341	PREFERRED_STACK_BOUNDARY	544
PHI nodes	277	prefetch	326
PIC	652	prefetch and ‘/v’	291
PIC_OFFSET_TABLE_REG_CALL_CLOBBERED	653	prefetch instruction pattern	473
PIC_OFFSET_TABLE_REGNUM	652	PREFETCH_SCHEDULE_BARRIER_P	291
PID_TYPE	555	PREINCREMENT_EXPR	195
pipeline hazard recognizer	508, 509	presence_set	512
Plugins	749	preserving SSA form	278
plus	312	pretend_args_size	603
plus and attributes	502	prev_active_insn	496
plus, canonicalization of	484	PREV_INSN	328
PLUS_EXPR	195	PRINT_OPERAND	674
Pmode	707	PRINT_OPERAND_ADDRESS	674
pmode_register_operand	381	PRINT_OPERAND_PUNCT_VALID_P	674
pointer	182	probe_stack instruction pattern	470
POINTER_DIFF_EXPR	195	probe_stack_address instruction pattern	470
POINTER_PLUS_EXPR	195	processor functional units	508, 509
POINTER_SIZE	542	processor pipeline description	508
POINTER_TYPE	182	product	313
POINTERS_EXTEND_UNSIGNED	542	profile feedback	354
poly_int	165	profile representation	354
poly_int, invariant range	165	PROFILE_BEFORE_PROLOGUE	605
poly_int, main typedefs	166	PROFILE_HOOK	605
poly_int, runtime value	165	profiling, code generation	605
poly_int, template parameters	165	program counter	311
poly_int, use in target-independent code	166	prologue	601
poly_int, use in target-specific code	166	prologue instruction pattern	472
POLY_INT_CST	192	PROMOTE_MODE	542
polynomial integers	165	pseudo registers	306
pop_operand	382	PSImode	296
popcount	316	PTRDIFF_TYPE	553
popcountm2 instruction pattern	454	purge_dead_edges	352, 356
pops_args	603	push address instruction	387
portability	3	push_operand	382
position independent code	652	push_reload	619
post_dec	326	PUSH_ARGS_REVERSED	587
post_inc	326	PUSH_ROUNDING	588
post_modify	326	pushm1 instruction pattern	436
post_order_compute, inverted_post_order_		PUT_CODE	283
compute, dom_walker::walk	349	PUT_MODE	301
POST_LINK_SPEC	534	PUT_REG_NOTE_KIND	334
POSTDECREMENT_EXPR	195		
POSTINCREMENT_EXPR	195		
POWI_MAX_MULTS	717		
powm3 instruction pattern	449		
pragma	708		
pre_dec	326		
pre_inc	326		

Q

QCmode	298
QFmode	296
QImode	295
QImode, in insn	333
QQmode	296
qualified type	182, 219
querying function unit reservations	510
question mark	390
quotient	314

R

'r' in constraint	386
rawmemchr instruction pattern	457
RDIV_EXPR	195
READONLY_DATA_SECTION_ASM_OP	648
real operands	271
REAL_CST	192
REAL_LIBGCC_SPEC	532
REAL_NM_FILE_NAME	672
REAL_TYPE	182
REAL_VALUE_ABS	686
REAL_VALUE_ATOF	686
REAL_VALUE_FIX	685
REAL_VALUE_ISINF	686
REAL_VALUE_ISNAN	686
REAL_VALUE_NEGATE	686
REAL_VALUE_NEGATIVE	686
REAL_VALUE_TO_TARGET_DECIMAL128	659
REAL_VALUE_TO_TARGET_DECIMAL32	659
REAL_VALUE_TO_TARGET_DECIMAL64	659
REAL_VALUE_TO_TARGET_DOUBLE	659
REAL_VALUE_TO_TARGET_LONG_DOUBLE	659
REAL_VALUE_TO_TARGET_SINGLE	659
REAL_VALUE_TYPE	685
REAL_VALUE_UNSIGNED_FIX	686
REALPART_EXPR	195
recog_data.operand	673
recognizing insns	371
RECORD_TYPE	182, 222
redirect_edge_and_branch	355
redirect_edge_and_branch, redirect_jump ..	356
reduc_and_scal_m instruction pattern	439
reduc_fmax_scal_m instruction pattern	438
reduc_fmin_scal_m instruction pattern	438
reduc_ior_scal_m instruction pattern	439
reduc_plus_scal_m instruction pattern	438
reduc_sbool_and_scal_m instruction pattern	439
reduc_sbool_ior_scal_m instruction pattern	439
reduc_sbool_xor_scal_m instruction pattern	439
reduc_smax_scal_m instruction pattern	438
reduc_smin_scal_m instruction pattern	438
reduc_umax_scal_m instruction pattern	438
reduc_umin_scal_m instruction pattern	438

reduc_xor_scal_m instruction pattern	439
reference	182
REFERENCE_TYPE	182
reg	306
reg and '/f'	291
reg and '/i'	291
reg and '/v'	291
reg, RTL sharing	347
reg_class_contents	558
reg_class_for_constraint	425
reg_label and '/v'	290
reg_names	558, 674
REG_ALLOC_ORDER	559
REG_BR_PRED	336
REG_BR_PROB	336
REG_BR_PROB_BASE, BB_FREQ_BASE, count	355
REG_BR_PROB_BASE, EDGE_FREQUENCY	354
REG_CALL_NOCF_CHECK	336
REG_CLASS_CONTENTS	564
REG_CLASS_NAMES	564
REG_DEAD	334
REG_DEAD, REG_UNUSED	357
REG_DEP_ANTI	336
REG_DEP_OUTPUT	336
REG_DEP_TRUE	336
REG_EH_REGION, EDGE_ABNORMAL_CALL	352
REG_EQUAL	335
REG_EQUIV	335
REG_EXPR	288
REG_FRAME_RELATED_EXPR	336
REG_FUNCTION_VALUE_P	291
REG_INC	334
REG_LABEL_OPERAND	334
REG_LABEL_TARGET	334
REG_NONNEG	334
REG_NOTE_KIND	334
REG_NOTES	333
REG_OFFSET	288
REG_OK_STRICT	618
REG_PARM_STACK_SPACE	588
REG_PARM_STACK_SPACE, and TARGET_FUNCTION_ARG	590
REG_POINTER	291
REG_SETJMP	334
REG_UNUSED	334
REG_USERVAR_P	291
REG_VALUE_IN_UNWIND_CONTEXT	585
REG_WORDS_BIG_ENDIAN	541
register allocation order	559
register class definitions	563
register class preference constraints	390
register pairs	560
Register Transfer Language (RTL)	283
register usage	556
register_operand	381
REGISTER_MOVE_COST	631
REGISTER_NAMES	673
REGISTER_PREFIX	675

REGISTER_TARGET_PRAGMAS	708	rotate	315
registers arguments	589	rotatert	315
registers in constraints	386	rotl _m 3 instruction pattern	446
REGMODE_NATURAL_SIZE	309, 310, 560	rotr _m 3 instruction pattern	446
REGNO_MODE_CODE_OK_FOR_BASE_P	566	ROUND_DIV_EXPR	195
REGNO_MODE_OK_FOR_BASE_P	566	ROUND_MOD_EXPR	195
REGNO_MODE_OK_FOR_REG_BASE_P	566	ROUND_TYPE_ALIGN	549
REGNO_OK_FOR_BASE_P	566	round _m 2 instruction pattern	450
REGNO_OK_FOR_INDEX_P	566	RPO	339
REGNO_OK_FOR_INSN_BASE_P	566	RROTATE_EXPR	195
REGNO_REG_CLASS	565	RSHIFT_EXPR	195
regs_ever_live	602	rsqrt _m 2 instruction pattern	447
regular expressions	508, 510	rtl_ssa::access_info	340
regular IPA passes	148	rtl_ssa::bb_info	339
relative costs	631	rtl_ssa::clobber_info	340
RELATIVE_PREFIX_NOT_LINKDIR	534	rtl_ssa::def_info	340
reload_completed	467	rtl_ssa::ebb_info	339
reload_in instruction pattern	427	rtl_ssa::insn_change	343
reload_in_progress	426	rtl_ssa::insn_info	339
reload_out instruction pattern	427	rtl_ssa::phi_info	340, 341
RELOAD_ELIMINABLE_REGS	587	rtl_ssa::set_info	340
reloading	159	rtl_ssa::use_info	340
remainder	314	RTL addition	312
remainder _m 3 instruction pattern	447	RTL addition with signed saturation	312
reorder	741	RTL addition with unsigned saturation	312
representation of RTL	283	RTL classes	284
reservation delays	508	RTL comparison	313
rest_of_decl_compilation	145	RTL comparison operations	316
rest_of_type_compilation	145	RTL constant expression types	302
restore_stack_block instruction pattern	469	RTL constants	302
restore_stack_function		RTL declarations	321
instruction pattern	469	RTL difference	312
restore_stack_nonlocal		RTL expression	283
instruction pattern	469	RTL expressions for arithmetic	312
RESULT_DECL	186	RTL format	285
return	322	RTL format characters	285
return instruction pattern	467	RTL function-call insns	337
return values in registers	598	RTL insn template	371
return_val	294	RTL integers	283
return_val, in call_insn	291	RTL memory expressions	306
return_val, in reg	291	RTL object types	283
return_val, in symbol_ref	293	RTL postdecrement	326
RETURN_ADDR_IN_PREVIOUS_FRAME	576	RTL postincrement	326
RETURN_ADDR_OFFSET	580	RTL predecrement	326
RETURN_ADDR_RTX	576	RTL preincrement	326
RETURN_ADDRESS_POINTER_REGNUM	584	RTL register expressions	306
RETURN_EXPR	226	RTL representation	283
RETURN_STMT	226	RTL side effect expressions	321
returning aggregate values	600	RTL SSA	338
reverse postorder	339	RTL strings	283
reverse probability	355	RTL structure sharing assumptions	347
REVERSE_CONDITION	630	RTL subtraction	312
REVERSIBLE_CC_MODE	630	RTL subtraction with signed saturation	312
right rotate	315	RTL subtraction with unsigned saturation	312
right shift	315	RTL sum	312
rint _m 2 instruction pattern	450	RTL vectors	283
RISC	508, 512	RTL_CONST_CALL_P	291
roots, marking	745	RTL_CONST_OR_PURE_CALL_P	292

RTL_LOOPING_CONST_OR_PURE_CALL_P	292
RTL_PURE_CALL_P	291
RTX (See RTL)	283
RTX codes, classes of	284
RTX_FRAME_RELATED_P	292
run-time target specification	537

S

‘s’ in constraint	386
sabdm3 instruction pattern	446
SAD_EXPR	203
same_type_p	183
SAmode	297
sat_fract	320
satfractmn2 instruction pattern	458
satfractunsmn2 instruction pattern	459
satisfies_constraint_m	425
save_stack_block instruction pattern	469
save_stack_function instruction pattern	469
save_stack_nonlocal instruction pattern	469
SAVE_EXPR	195
SBSS_SECTION_ASM_OP	648
Scalar evolutions	363
scalar_float_mode	299
scalar_int_mode	299
scalar_mode	299
scalars, returned as values	598
scalbm3 instruction pattern	447
scatter_storemn instruction pattern	431
SCHED_GROUP_P	292
SCmode	298
scratch	311
scratch operands	311
scratch, RTL sharing	347
scratch_operand	381
SDATA_SECTION_ASM_OP	648
sdiv_pow2m3 instruction pattern	441
SDmode	296
sdot_prodmn instruction pattern	440
search options	494
SECONDARY_INPUT_RELOAD_CLASS	570
SECONDARY_MEMORY_NEEDED_RTX	571
SECONDARY_OUTPUT_RELOAD_CLASS	570
SECONDARY_RELOAD_CLASS	570
select_vlmm instruction pattern	433
SELECT_CC_MODE	629
sequence	325
Sequence iterators	264
set	321
set and ‘/f’	292
set_attr	504
set_attr_alternative	504
set_bb_seq	264
set_optab_libfunc	614
set_thread_pointermode instruction pattern	479
SET_ASM_OP	667, 668

SET_DEST	322
SET_IS_RETURN_P	292
SET_LABEL_KIND	330
SET_RATIO	636
SET_SRC	322
SET_TYPE_STRUCTURAL_EQUALITY	182, 183
setmemm instruction pattern	456
SETUP_FRAME_ADDRESSES	576
SFmode	296
sharing of RTL components	347
shift	315
SHIFT_COUNT_TRUNCATED	703
SHLIB_SUFFIX	672
SHORT_ACCUM_TYPE_SIZE	552
SHORT_FRACT_TYPE_SIZE	552
SHORT_IMMEDIATES_SIGN_EXTEND	703
SHORT_TYPE_SIZE	551
shrink-wrapping separate components	606
sibcall_epilogue instruction pattern	472
sibling call	352
SIBLING_CALL_P	292
SIG_ATOMIC_TYPE	554
sign_extend	319
sign_extract	318
sign_extract, canonicalization of	485
signal-to-noise ratio (metaphorical usage for diagnostics)	785
signbitm2 instruction pattern	449
signed division	314
signed division with signed saturation	314
signed maximum	314
signed minimum	314
significandm2 instruction pattern	449
SImode	296
simple constraints	385
simple_return	322
simple_return instruction pattern	467
sincosm3 instruction pattern	448
sinm2 instruction pattern	447
SIZE_ASM_OP	661
SIZE_TYPE	553
SIZETYPE	553
skip	739
SLOW_BYTE_ACCESS	634
small IPA passes	147
smax	314
smin	314
sms, swing, software pipelining	158
smul_highpart	314
smulhrsm3 instruction pattern	441
smulhsm3 instruction pattern	441
smulm3_highpart instruction pattern	445
soft float library	9
source code, location information	787
spaceshipm4 instruction pattern	480
special	742
special predicates	380
SPECS	732

- speculation_barrier instruction pattern 473
- speed of instructions 631
- split_block 356
- splitting instructions 489
- SQmode 297
- sqrtd 315
- sqrtdm2 instruction pattern 447
- square root 315
- ss_abs 315
- ss_ashift 315
- ss_div 314
- ss_minus 312
- ss_mult 313
- ss_neg 313
- ss_plus 312
- ss_truncate 319
- ssaddm3 instruction pattern 436
- ssadm instruction pattern 440
- ssashl3 instruction pattern 446
- SSA 276
- SSA, RTL form 338
- SSA_NAME_DEF_STMT 279
- SSA_NAME_VERSION 279
- ssdivm3 instruction pattern 436
- ssmaddmn4 instruction pattern 445
- ssmsubmn4 instruction pattern 445
- ssmulm3 instruction pattern 436
- ssnegm2 instruction pattern 447
- sssubm3 instruction pattern 436
- sstruncmn2 instruction pattern 437
- stack arguments 587
- stack frame layout 574
- stack smashing protection 607
- stack_pointer_rtx 585
- stack_protect_combined_set
 - instruction pattern 479
- stack_protect_combined_test
 - instruction pattern 479
- stack_protect_set instruction pattern 479
- stack_protect_test instruction pattern 480
- STACK_ADDRESS_OFFSET 578
- STACK_ALIGNMENT_NEEDED 575
- STACK_BOUNDARY 544
- STACK_CHECK_BUILTIN 582
- STACK_CHECK_FIXED_FRAME_SIZE 582
- STACK_CHECK_MAX_FRAME_SIZE 582
- STACK_CHECK_MAX_VAR_SIZE 583
- STACK_CHECK_MOVING_SP 582
- STACK_CHECK_PROBE_INTERVAL_EXP 582
- STACK_CHECK_PROTECT 582
- STACK_CHECK_STATIC_BUILTIN 582
- STACK_DYNAMIC_OFFSET 575
- STACK_DYNAMIC_OFFSET and virtual registers . . 307
- STACK_GROWS_DOWNWARD 574
- STACK_PARMs_IN_REG_PARM_AREA 588
- STACK_POINTER_OFFSET 575
- STACK_POINTER_OFFSET and virtual registers . . 307
- STACK_POINTER_REGNUM 583
- STACK_POINTER_REGNUM and virtual registers . . 307
- STACK_PUSH_CODE 574
- STACK_REG_COVER_CLASS 563
- STACK_REGS 563
- STACK_SAVEAREA_MODE 549
- STACK_SIZE_MODE 549
- STACK_SLOT_ALIGNMENT 546
- standard pattern names 426
- STANDARD_STARTFILE_PREFIX 535
- STANDARD_STARTFILE_PREFIX_1 535
- STANDARD_STARTFILE_PREFIX_2 535
- STARTFILE_SPEC 532
- Statement and operand traversals 268
- Statement Sequences 209
- Statements 207
- statements 217, 226
- static analysis 773
- static analyzer 773
- static analyzer, debugging 781
- static analyzer, internals 773
- Static profile estimation 354
- static single assignment 276
- STATIC_CHAIN_INCOMING_REGNUM 584
- STATIC_CHAIN_REGNUM 584
- stdarg.h and register arguments 590
- STDC_0_IN_SYSTEM_HEADERS 707
- STMT_EXPR 195
- STMT_IS_FULL_EXPR_P 226
- storage layout 541
- 'store_multiple' instruction pattern 428
- STORE_FLAG_VALUE 705
- STORE_MAX_PIECES 635
- strcpy 546
- strict_low_part 321
- strict_memory_address_p 619
- STRICT_ALIGNMENT 547
- string_length 739
- STRING_CST 192
- STRING_POOL_ADDRESS_P 292
- strlenm instruction pattern 457
- structure value address 600
- STRUCTURE_SIZE_BOUNDARY 547
- subm3 instruction pattern 436
- SUBOBJECT 226
- SUBOBJECT_CLEANUP 226
- subreg 308
- subreg and '/s' 293
- subreg and '/u' 293
- subreg and '/u' and '/v' 293
- subreg, in strict_low_part 321
- SUBREG_BYTE 311
- SUBREG_PROMOTED_UNSIGNED_P 293
- SUBREG_PROMOTED_UNSIGNED_SET 293
- SUBREG_PROMOTED_VAR_P 293
- SUBREG_REG 311
- subst iterators in .md files 525
- subvm4 instruction pattern 437
- SUCCESS_EXIT_CODE 727

support for nested functions.....	611
SUPPORTS_INIT_PRIORITY.....	671
SUPPORTS_ONE_ONLY.....	665
SUPPORTS_WEAK.....	665
SWITCH_BODY.....	226
SWITCH_COND.....	226
SWITCH_STMT.....	226
SWITCHABLE_TARGET.....	540
symbol_ref.....	305
symbol_ref and ‘/f’.....	292
symbol_ref and ‘/i’.....	293
symbol_ref and ‘/u’.....	290
symbol_ref and ‘/v’.....	293
symbol_ref, RTL sharing.....	347
SYMBOL_FLAG_ANCHOR.....	289
SYMBOL_FLAG_EXTERNAL.....	289
SYMBOL_FLAG_FUNCTION.....	289
SYMBOL_FLAG_HAS_BLOCK_INFO.....	289
SYMBOL_FLAG_LOCAL.....	289
SYMBOL_FLAG_SMALL.....	289
SYMBOL_FLAG_TLS_SHIFT.....	289
SYMBOL_REF_ANCHOR_P.....	289
SYMBOL_REF_BLOCK.....	289
SYMBOL_REF_BLOCK_OFFSET.....	289
SYMBOL_REF_CONSTANT.....	288
SYMBOL_REF_DATA.....	289
SYMBOL_REF_DECL.....	288
SYMBOL_REF_EXTERNAL_P.....	289
SYMBOL_REF_FLAG.....	293
SYMBOL_REF_FLAG, in	
TARGET_ENCODE_SECTION_INFO.....	651
SYMBOL_REF_FLAGS.....	289
SYMBOL_REF_FUNCTION_P.....	289
SYMBOL_REF_HAS_BLOCK_INFO_P.....	289
SYMBOL_REF_LOCAL_P.....	289
SYMBOL_REF_SMALL_P.....	289
SYMBOL_REF_TLS_MODEL.....	289
SYMBOL_REF_USED.....	293
SYMBOL_REF_WEAK.....	293
symbolic label.....	347
sync_addmode instruction pattern.....	474
sync_andmode instruction pattern.....	474
instruction pattern.....	474
sync_iormode instruction pattern.....	474
sync_lock_releasemode instruction pattern...	475
sync_lock_test_and_setmode	
instruction pattern.....	475
sync_nandmode instruction pattern.....	474
sync_new_addmode instruction pattern.....	475
sync_new_andmode instruction pattern.....	475
sync_new_iormode instruction pattern.....	475
sync_new_nandmode instruction pattern.....	475
sync_new_submode instruction pattern.....	475
sync_new_xormode instruction pattern.....	475
sync_old_addmode instruction pattern.....	475
sync_old_andmode instruction pattern.....	475
sync_old_iormode instruction pattern.....	475

sync_old_nandmode instruction pattern.....	475
sync_old_submode instruction pattern.....	475
sync_old_xormode instruction pattern.....	475
sync_submode instruction pattern.....	474
sync_xormode instruction pattern.....	474
SYSROOT_HEADERS_SUFFIX_SPEC.....	533
SYSROOT_SUFFIX_SPEC.....	533
SYSTEM_IMPLICIT_EXTERN_C.....	708

T

t-target.....	729
table jump.....	350
tablejump instruction pattern.....	468
tag.....	740
tag_memory instruction pattern.....	480
tagging insns.....	503
tail calls.....	605
TAmode.....	297
tanm2 instruction pattern.....	448
target attributes.....	688
target description macros.....	529
target functions.....	529
target hooks.....	529
target makefile fragment.....	729
target specifications.....	537
target_flags.....	537
TARGET_ABSOLUTE_BIGGEST_ALIGNMENT.....	544
TARGET_ADDITIONAL_ALLOCNO_CLASS_P.....	574
TARGET_ADDR_SPACE_ADDRESS_MODE.....	699
TARGET_ADDR_SPACE_CONVERT.....	700
TARGET_ADDR_SPACE_DEBUG.....	700
TARGET_ADDR_SPACE_DIAGNOSE_USAGE.....	701
TARGET_ADDR_SPACE_FOR_	
ARTIFICIAL_RODATA.....	701
TARGET_ADDR_SPACE_LEGITIMATE_ADDRESS_P...	700
TARGET_ADDR_SPACE_LEGITIMIZE_ADDRESS.....	700
TARGET_ADDR_SPACE_POINTER_MODE.....	699
TARGET_ADDR_SPACE_SUBSET_P.....	700
TARGET_ADDR_SPACE_VALID_POINTER_MODE.....	700
TARGET_ADDR_SPACE_ZERO_ADDRESS_VALID.....	700
TARGET_ADDRESS_COST.....	637
TARGET_ALIGN_ANON_BITFIELD.....	548
TARGET_ALLOCATE_INITIAL_VALUE.....	715
TARGET_ALLOCATE_STACK_SLOTS_FOR_ARGS.....	720
TARGET_ALWAYS_STRIP_DOTDOT.....	534
TARGET_ARG_PARTIAL_BYTES.....	591
TARGET_ARM_EABI_UNWINDER.....	680
TARGET_ARRAY_MODE.....	596
TARGET_ARRAY_MODE_SUPPORTED_P.....	596
TARGET_ASAN_DYNAMIC_SHADOW_OFFSET_P.....	720
TARGET_ASAN_SHADOW_OFFSET.....	720
TARGET_ASM_ALIGNED_DI_OP.....	656
TARGET_ASM_ALIGNED_HI_OP.....	656
TARGET_ASM_ALIGNED_PDI_OP.....	656
TARGET_ASM_ALIGNED_PSI_OP.....	656
TARGET_ASM_ALIGNED_PTI_OP.....	656
TARGET_ASM_ALIGNED_SI_OP.....	656

TARGET_ASM_ALIGNED_TI_OP	656	TARGET_ASM_TRAMPOLINE_TEMPLATE	613
TARGET_ASM_ASSEMBLE_UNDEFINED_DECL	664	TARGET_ASM_TTYPE	680
TARGET_ASM_ASSEMBLE_VISIBILITY	665	TARGET_ASM_UNALIGNED_DI_OP	656
TARGET_ASM_BYTE_OP	656	TARGET_ASM_UNALIGNED_HI_OP	656
TARGET_ASM_CAN_OUTPUT_MI_THUNK	605	TARGET_ASM_UNALIGNED_PDI_OP	656
TARGET_ASM_CLOSE_PAREN	658	TARGET_ASM_UNALIGNED_PSI_OP	656
TARGET_ASM_CODE_END	654	TARGET_ASM_UNALIGNED_PTI_OP	656
TARGET_ASM_CONSTRUCTOR	671	TARGET_ASM_UNALIGNED_SI_OP	656
TARGET_ASM_DECL_END	657	TARGET_ASM_UNALIGNED_TI_OP	656
TARGET_ASM_DECLARE_CONSTANT_NAME	663	TARGET_ASM_UNIQUE_SECTION	650
TARGET_ASM_DESTRUCTOR	671	TARGET_ASM_UNWIND_EMIT	677
TARGET_ASM_ELF_FLAGS_NUMERIC	655	TARGET_ASM_UNWIND_EMIT_BEFORE_INSN	678
TARGET_ASM_EMIT_EXCEPT_PERSONALITY	677	TARGET_ATOMIC_ALIGN_FOR_MODE	721
TARGET_ASM_EMIT_EXCEPT_TABLE_LABEL	677	TARGET_ATOMIC_ASSIGN_EXPAND_FENV	721
TARGET_ASM_EMIT_UNWIND_LABEL	677	TARGET_ATOMIC_TEST_AND_SET_TRUEVAL	720
TARGET_ASM_EXTERNAL_LIBCALL	666	TARGET_ATTRIBUTE_TABLE	689
TARGET_ASM_FILE_END	653	TARGET_ATTRIBUTE_TAKES_IDENTIFIER_P	689
TARGET_ASM_FILE_START	653	TARGET_AVOID_STORE_FORWARDING_P	639
TARGET_ASM_FILE_START_APP_OFF	653	TARGET_BINDS_LOCAL_P	652
TARGET_ASM_FILE_START_FILE_DIRECTIVE	653	TARGET_BUILD_BUILTIN_VA_LIST	594
TARGET_ASM_FINAL_POSTSCAN_INSN	674	TARGET_BUILTIN_DECL	712
TARGET_ASM_FUNCTION_BEGIN_EPILOGUE	602	TARGET_BUILTIN_RECIPROCAL	621
TARGET_ASM_FUNCTION_END_PROLOGUE	602	TARGET_BUILTIN_SETJMP_FRAME_VALUE	576
TARGET_ASM_FUNCTION_EPILOGUE	602	TARGET_C_BITINT_TYPE_INFO	543
TARGET_ASM_FUNCTION_PROLOGUE	602	TARGET_C_EXCESS_PRECISION	543
TARGET_ASM_FUNCTION_RODATA_SECTION	650	TARGET_C_MODE_FOR_FLOATING_TYPE	543
TARGET_ASM_FUNCTION_SECTION	655	TARGET_C_PREINCLUDE	707
TARGET_ASM_FUNCTION_SWITCHED_		TARGET_CALL_ARGS	611
TEXT_SECTIONS	655	TARGET_CALL_FUSAGE_CONTAINS_NON_	
TARGET_ASM_GENERATE_PIC_ADDR_DIFF_VEC	650	CALLEE_CLOBBERS	608
TARGET_ASM_GLOBALIZE_DECL_NAME	664	TARGET_CALL_OFFSET_RETURN_LABEL	610
TARGET_ASM_GLOBALIZE_LABEL	664	TARGET_CALLEE_COPIES	591
TARGET_ASM_INIT_SECTIONS	649	TARGET_CALLEE_SAVE_COST	632
TARGET_ASM_INTEGER	657	TARGET_CAN_CHANGE_MODE_CLASS	572
TARGET_ASM_INTERNAL_LABEL	666	TARGET_CAN_CHANGE_MODE_CLASS and	
TARGET_ASM_LTO_END	654	subreg semantics	311
TARGET_ASM_LTO_START	654	TARGET_CAN_ELIMINATE	587
TARGET_ASM_MAKE_EH_SYMBOL_INDIRECT	677	TARGET_CAN_FOLLOW_JUMP	715
TARGET_ASM_MARK_DECL_PRESERVED	666	TARGET_CAN_INLINE_P	693
TARGET_ASM_MERGEABLE_RODATA_PREFIX	651	TARGET_CAN_USE_DOLOOP_P	714
TARGET_ASM_NAMED_SECTION	655	TARGET_CANNOT_FORCE_CONST_MEM	620
TARGET_ASM_OPEN_PAREN	658	TARGET_CANNOT_MODIFY_JUMPS_P	716
TARGET_ASM_OUTPUT_ADDR_CONST_EXTRA	657	TARGET_CANNOT_SUBSTITUTE_MEM_EQUIV_P	573
TARGET_ASM_OUTPUT_ANCHOR	628	TARGET_CANONICAL_VA_LIST_TYPE	594
TARGET_ASM_OUTPUT_DWARF_DTPREL	684	TARGET_CANONICALIZE_COMPARISON	629
TARGET_ASM_OUTPUT_IDENT	655	TARGET_CASE_VALUES_THRESHOLD	702
TARGET_ASM_OUTPUT_MI_THUNK	604	TARGET_CC_MODES_COMPATIBLE	631
TARGET_ASM_OUTPUT_SOURCE_FILENAME	654	TARGET_CHECK_BUILTIN_CALL	712
TARGET_ASM_POST_CFI_STARTPROC	677	TARGET_CHECK_PCH_TARGET_FLAGS	695
TARGET_ASM_PRINT_PATCHABLE_		TARGET_CHECK_STRING_OBJECT_FORMAT_ARG	539
FUNCTION_ENTRY	601	TARGET_CHECK_TARGET_CLONE_VERSION	713
TARGET_ASM_RECORD_GCC_SWITCHES	656	TARGET_CLASS_LIKELY_SPILLED_P	571
TARGET_ASM_RECORD_GCC_SWITCHES_SECTION	656	TARGET_CLASS_MAX_NREGS	572
TARGET_ASM_RELOC_RW_MASK	650	TARGET_CLONES_ATTR_SEPARATOR	691
TARGET_ASM_SELECT_RTX_SECTION	651	TARGET_COMMUTATIVE_P	715
TARGET_ASM_SELECT_SECTION	650	TARGET_COMP_TYPE_ATTRIBUTES	689
TARGET_ASM_SHOULD_RESTORE_CFA_STATE	678	TARGET_COMPARE_BY_PIECES_BRANCH_RATIO	635
TARGET_ASM_TM_CLONE_TABLE_SECTION	651	TARGET_COMPARE_VERSION_PRIORITY	713

TARGET_COMPATIBLE_VECTOR_TYPES_P	596	TARGET_DWARF_POLY_INDETERMINATE_VALUE....	577
TARGET_COMPUTE_FRAME_LAYOUT	587	TARGET_DWARF_REGISTER_SPAN	680
TARGET_COMPUTE_MULTILIB	539	TARGET_EDOM	615
TARGET_COMPUTE_PRESSURE_CLASSES	574	TARGET_EMIT_CALL_BUILTIN__CLEAR_CACHE...	613
TARGET_CONDITIONAL_REGISTER_USAGE	558	TARGET_EMIT_EPILOGUE_FOR_SIBCALL	711
TARGET_CONST_ANCHOR	720	TARGET_EMIT_SUPPORT_TINFOS	551
TARGET_CONST_NOT_OK_FOR_DEBUG_P	620	TARGET_EMPTY_RECORD_P	601
TARGET_CONSTANT_ALIGNMENT	546	TARGET_EMUTLS_DEBUG_FORM_TLS_ADDRESS	694
TARGET_CONVERT_TO_TYPE	719	TARGET_EMUTLS_GET_ADDRESS	694
TARGET_CPU_CPP_BUILTINS	537	TARGET_EMUTLS_REGISTER_COMMON	694
TARGET_CSTORE_MODE	574	TARGET_EMUTLS_TMPL_PREFIX	694
TARGET_CUSTOM_FUNCTION_DESCRIPTOR	612	TARGET_EMUTLS_TMPL_SECTION	694
TARGET_CXX_ADJUST_CDTOR_CALLABI_FNTYPE...	697	TARGET_EMUTLS_VAR_ALIGN_FIXED	694
TARGET_CXX_ADJUST_CLASS_AT_DEFINITION...	697	TARGET_EMUTLS_VAR_FIELDS	694
TARGET_CXX_CDTOR_RETURNS_THIS	696	TARGET_EMUTLS_VAR_INIT	694
TARGET_CXX_CLASS_DATA_ALWAYS_COMDAT	697	TARGET_EMUTLS_VAR_PREFIX	694
TARGET_CXX_COOKIE_HAS_SIZE	696	TARGET_EMUTLS_VAR_SECTION	694
TARGET_CXX_DECL_MANGLING_CONTEXT	697	TARGET_ENCODE_SECTION_INFO	651
TARGET_CXX_DETERMINE_CLASS_		TARGET_ENCODE_SECTION_INFO and	
DATA_VISIBILITY	696	address validation	618
TARGET_CXX_GET_COOKIE_SIZE	696	TARGET_ENCODE_SECTION_INFO usage	674
TARGET_CXX_GUARD_MASK_BIT	696	TARGET_END_CALL_ARGS	611
TARGET_CXX_GUARD_TYPE	696	TARGET_ENUM_VA_LIST_P	594
TARGET_CXX_IMPLICIT_EXTERN_C	707	TARGET_ESTIMATED_POLY_VALUE	639
TARGET_CXX_IMPORT_EXPORT_CLASS	696	TARGET_EXCEPT_UNWIND_INFO	679
TARGET_CXX_KEY_METHOD_MAY_BE_INLINE	696	TARGET_EXECUTABLE_SUFFIX	716
TARGET_CXX_LIBRARY_RTTI_COMDAT	697	TARGET_EXPAND_BUILTIN	712
TARGET_CXX_USE_AEABI_ATEXIT	697	TARGET_EXPAND_BUILTIN_SAVEREGS	609
TARGET_CXX_USE_ATEXIT_FOR_CXA_ATEXIT	697	TARGET_EXPAND_DIVMOD_LIBFUNC	647
TARGET_D_CPU_VERSIONS	697	TARGET_EXPAND_TO_RTL_HOOK	550
TARGET_D_HAS_STDCALL_CONVENTION	698	TARGET_EXPR	195
TARGET_D_MINFO_SECTION	698	TARGET_EXTRA_INCLUDES	717
TARGET_D_MINFO_SECTION_END	698	TARGET_EXTRA_LIVE_ON_ENTRY	606
TARGET_D_MINFO_SECTION_START	698	TARGET_EXTRA_PRE_INCLUDES	717
TARGET_D_OS_VERSIONS	697	TARGET_FIXED_CONDITION_CODE_REGS	630
TARGET_D_REGISTER_CPU_TARGET_INFO	698	TARGET_FIXED_POINT_SUPPORTED_P	550
TARGET_D_REGISTER_OS_TARGET_INFO	698	TARGET_FLAGS_REGNUM	631
TARGET_D_TEMPLATES_ALWAYS_COMDAT	698	TARGET_FLOAT_EXCEPTIONS_	
TARGET_DEBUG_UNWIND_INFO	683	ROUNDING_SUPPORTED_P	540
TARGET_DECIMAL_FLOAT_SUPPORTED_P	550	TARGET_FLOATN_BUILTIN_P	597
TARGET_DECLSPEC	690	TARGET_FLOATN_MODE	597
TARGET_DEFAULT_PACK_STRUCT	709	TARGET_FN_ABI_VA_LIST	594
TARGET_DEFAULT_SHORT_ENUMS	553	TARGET_FNTYPE_ABI	557
TARGET_DEFAULT_TARGET_FLAGS	538	TARGET_FOLD_BUILTIN	712
TARGET_DEFERRED_OUTPUT_DEFS	668	TARGET_FORMAT_TYPES	718
TARGET_DELAY_SCHED2	684	TARGET_FORTIFY_SOURCE_DEFAULT_LEVEL	616
TARGET_DELAY_VARTRACK	684	TARGET_FRAME_ALLOCATION_COST	633
TARGET_DELEGITIMIZE_ADDRESS	620	TARGET_FRAME_POINTER_REQUIRED	586
TARGET_DIFFERENT_ADDR_DISPLACEMENT_P	573	TARGET_FUNCTION_ARG	589
TARGET_DLLIMPORT_DECL_ATTRIBUTES	690	TARGET_FUNCTION_ARG_ADVANCE	592
TARGET_DOCUMENTATION_NAME	724	TARGET_FUNCTION_ARG_BOUNDARY	593
TARGET_DLOOP_COST_FOR_ADDRESS	714	TARGET_FUNCTION_ARG_OFFSET	593
TARGET_DLOOP_COST_FOR_GENERIC	713	TARGET_FUNCTION_ARG_PADDING	593
TARGET_DTORS_FROM_CXA_ATEXIT	671	TARGET_FUNCTION_ARG_ROUND_BOUNDARY	593
TARGET_DW_CFI_OPRND1_DESC	680	TARGET_FUNCTION_ATTRIBUTE_INLINABLE_P	691
TARGET_DWARF_CALLING_CONVENTION	683	TARGET_FUNCTION_INCOMING_ARG	590
TARGET_DWARF_FRAME_REG_MODE	680	TARGET_FUNCTION_OK_FOR_SIBCALL	605
TARGET_DWARF_HANDLE_FRAME_UNSPEC	577	TARGET_FUNCTION_VALUE	598

TARGET_FUNCTION_VALUE_REGNO_P	599	TARGET_INSTRUCTION_SELECTION	626
TARGET_GEN_CCMP_FIRST	716	TARGET_INVALID_ARG_FOR_UNPROTOTYPED_FN...	718
TARGET_GEN_CCMP_NEXT	716	TARGET_INVALID_BINARY_OP	718
TARGET_GENERATE_VERSION_ DISPATCHER_BODY	713	TARGET_INVALID_CONVERSION	718
TARGET_GET_DRAP_RTX	719	TARGET_INVALID_UNARY_OP	718
TARGET_GET_FUNCTION_ VERSIONS_DISPATCHER	713	TARGET_INVALID_WITHIN_DOLOOP	714
TARGET_GET_MULTILIB_ABI_NAME	558	TARGET_IRA_CHANGE_PSEUDO_ALLOCNO_CLASS...	572
TARGET_GET_PCH_VALIDITY	695	TARGET_JIT_REGISTER_CPU_TARGET_INFO	699
TARGET_GET_RAW_ARG_MODE	601	TARGET_KEEP_LEAF_WHEN_PROFILED	605
TARGET_GET_RAW_RESULT_MODE	601	TARGET_LEGITIMATE_ADDRESS_P	617
TARGET_GET_VALID_OPTION_VALUES	608	TARGET_LEGITIMATE_COMBINED_INSN	714
TARGET_GIMPLE_FOLD_BUILTIN	713	TARGET_LEGITIMATE_CONSTANT_P	620
TARGET_GIMPLIFY_VA_ARG_EXPR	594	TARGET_LEGITIMIZE_ADDRESS	618
TARGET_GOACC_ADJUST_PRIVATE_DECL	627	TARGET_LEGITIMIZE_ADDRESS_DISPLACEMENT...	573
TARGET_GOACC_CREATE_WORKER_ BROADCAST_RECORD	627	TARGET_LIB_INT_CMP_BIASED	615
TARGET_GOACC_DIM_LIMIT	626	TARGET_LIBC_HAS_FAST_FUNCTION	616
TARGET_GOACC_EXPAND_VAR_DECL	627	TARGET_LIBC_HAS_FUNCTION	615
TARGET_GOACC_FORK_JOIN	626	TARGET_LIBCALL_VALUE	599
TARGET_GOACC_REDUCTION	626	TARGET_LIBFUNC_GNU_PREFIX	615
TARGET_GOACC_SHARED_MEM_LAYOUT	627	TARGET_LIBGCC_CMP_RETURN_MODE	549
TARGET_GOACC_VALIDATE_DIMS	625	TARGET_LIBGCC_FLOATING_ MODE_SUPPORTED_P	596
TARGET_HANDLE_C_OPTION	538	TARGET_LIBGCC_SDATA_SECTION	649
TARGET_HANDLE_GENERIC_ATTRIBUTE	690	TARGET_LIBGCC_SHIFT_COUNT_MODE	549
TARGET_HANDLE_OPTION	538	TARGET_LIBM_FUNCTION_MAX_ERROR	616
TARGET_HARD_REGNO_CALL_PART_CLOBBERED...	557	TARGET_LOOP_UNROLL_ADJUST	717
TARGET_HARD_REGNO_MODE_OK	560	TARGET_LOWER_LOCAL_DECL_ALIGNMENT	545
TARGET_HARD_REGNO_MODE_OK and constraints	422	TARGET_LRA_P	573
TARGET_HARD_REGNO_NREGS	559	TARGET_MACHINE_DEPENDENT_REORG	711
TARGET_HARD_REGNO_SCRATCH_OK	561	TARGET_MANGLE_ASSEMBLER_NAME	666
TARGET_HAS_FMV_TARGET_ATTRIBUTE	691	TARGET_MANGLE_DECL_ASSEMBLER_NAME	651
TARGET_HAS_IFUNC_P	720	TARGET_MANGLE_TYPE	550
TARGET_HAS_NO_HW_DIVIDE	615	TARGET_MAX_ANCHOR_OFFSET	628
TARGET_HAVE_CCMP	717	TARGET_MAX_NOCE_IFCVT_SEQ_COST	638
TARGET_HAVE_CONDITIONAL_EXECUTION	716	TARGET_MD_ASM_ADJUST	709
TARGET_HAVE_COUNT_REG_DECR_P	713	TARGET_MEM_CONSTRAINT	618
TARGET_HAVE_CTORS_DTORS	671	TARGET_MEM_REF	194
TARGET_HAVE_LIBATOMIC	723	TARGET_MEMBER_TYPE_FORCES_BLK	549
TARGET_HAVE_NAMED_SECTIONS	655	TARGET_MEMMODEL_CHECK	720
TARGET_HAVE_SHADOW_CALL_STACK	723	TARGET_MEMORY_MOVE_COST	632
TARGET_HAVE_SPECULATION_SAFE_VALUE	722	TARGET_MEMTAG_ADD_TAG	723
TARGET_HAVE_SRODATA_SECTION	652	TARGET_MEMTAG_CAN_TAG_ADDRESSES	722
TARGET_HAVE_STRUB_SUPPORT_FOR	578	TARGET_MEMTAG_EXTRACT_TAG	723
TARGET_HAVE_SWITCHABLE_BSS_SECTIONS	655	TARGET_MEMTAG_GRANULE_SIZE	723
TARGET_HAVE_TLS	652	TARGET_MEMTAG_INSERT_RANDOM_TAG	723
TARGET_IFUNC_REF_LOCAL_OK	720	TARGET_MEMTAG_SET_TAG	723
TARGET_IN_SMALL_DATA_P	652	TARGET_MEMTAG_TAG_BITSIZE	722
TARGET_INIT_BUILTINS	711	TARGET_MEMTAG_UNTAGGED_POINTER	723
TARGET_INIT_DWARF_REG_SIZES_EXTRA	680	TARGET_MERGE_DECL_ATTRIBUTES	690
TARGET_INIT_LIBFUNCS	614	TARGET_MERGE_TYPE_ATTRIBUTES	689
TARGET_INIT_PIC_REG	591	TARGET_MIN_ANCHOR_OFFSET	628
TARGET_INSERT_ATTRIBUTES	690	TARGET_MIN_ARITHMETIC_PRECISION	702
TARGET_INSN_CALLEE_ABI	557	TARGET_MIN_DIVISIONS_FOR_RECIP_MUL	703
TARGET_INSN_COST	638	TARGET_MODE_AFTER	687
TARGET_INSTANTIATE_DECLS	550	TARGET_MODE_BACKPROP	688
		TARGET_MODE_CAN_TRANSFER_BITS	595
		TARGET_MODE_CONFLUENCE	687
		TARGET_MODE_DEPENDENT_ADDRESS_P	619

TARGET_MODE_EH_HANDLER	688	TARGET_PREFERRED_ELSE_VALUE	626
TARGET_MODE_EMIT	687	TARGET_PREFERRED_OUTPUT_RELOAD_CLASS	568
TARGET_MODE_ENTRY	688	TARGET_PREFERRED_RELOAD_CLASS	567
TARGET_MODE_EXIT	688	TARGET_PREFERRED_RENAME_CLASS	567
TARGET_MODE_NEEDED	687	TARGET_PREPARE_PCH_SAVE	695
TARGET_MODE_PRIORITY	688	TARGET_PRETEND_OUTGOING_VARARGS_NAMED	611
TARGET_MODE_REP_EXTENDED	704	TARGET_PROFILE_BEFORE_PROLOGUE	652
TARGET_MODES_TIEABLE_P	561	TARGET_PROMOTE_FUNCTION_MODE	543
TARGET_MS_BITFIELD_LAYOUT_P	550	TARGET_PROMOTE_PROTOTYPES	587
TARGET_MUST_PASS_IN_STACK	590	TARGET_PROMOTED_TYPE	718
TARGET_MUST_PASS_IN_STACK, and		TARGET_PTRMEMFUNC_VBIT_LOCATION	555
TARGET_FUNCTION_ARG	590	TARGET_PUSH_ARGUMENT	587
TARGET_N_FORMAT_TYPES	718	TARGET_RECORD_OFFLOAD_SYMBOL	721
TARGET_NARROW_VOLATILE_BITFIELD	548	TARGET_REDZONE_CLOBBER	595
TARGET_NEED_IPA_FN_TARGET_INFO	693	TARGET_REF_MAY_ALIAS_ERRNO	595
TARGET_NEW_ADDRESS_PROFITABLE_P	639	TARGET_REGISTER_MOVE_COST	631
TARGET_NO_REGISTER_ALLOCATION	684	TARGET_REGISTER_PRIORITY	573
TARGET_NO_SPECULATION_IN_DELAY_SLOTS_P	639	TARGET_REGISTER_USAGE_LEVELING_P	573
TARGET_NOCE_CONVERSION_PROFITABLE_P	638	TARGET_RELAYOUT_FUNCTION	693
TARGET_OBJS_CONSTRUCT_STRING_OBJECT	538	TARGET_RESET_LOCATION_VIEW	683
TARGET_OBJS_DECLARE_CLASS_DEFINITION	538	TARGET_RESOLVE_OVERLOADED_BUILTIN	712
TARGET_OBJS_DECLARE_UNRESOLVED_		TARGET_RETURN_IN_MEMORY	600
CLASS_REFERENCE	538	TARGET_RETURN_IN_MSB	599
TARGET_OBJECT_SUFFIX	716	TARGET_RETURN_POPS_ARGS	589
TARGET_OBJFMT_CPP_BUILTINS	537	TARGET_RTX_COSTS	637
TARGET_OFFLOAD_OPTIONS	721	TARGET_RUN_TARGET_SELFTESTS	722
TARGET_OMIT_STRUCT_RETURN_REG	599	TARGET_RUST_CPU_INFO	698
TARGET_OMP_DEVICE_KIND_ARCH_ISA	625	TARGET_RUST_OS_INFO	699
TARGET_OPTAB_SUPPORTED_P	637	TARGET_SCALAR_MODE_SUPPORTED_P	595
TARGET_OPTF	717	TARGET_SCHED_ADJUST_COST	640
TARGET_OPTION_FUNCTIONS_B_		TARGET_SCHED_ADJUST_PRIORITY	640
RESOLVABLE_FROM_A	692	TARGET_SCHED_ALLOC_SCHED_CONTEXT	644
TARGET_OPTION_INIT_STRUCT	539	TARGET_SCHED_CAN_SPECULATE_INSN	645
TARGET_OPTION_OPTIMIZATION_TABLE	539	TARGET_SCHED_CLEAR_SCHED_CONTEXT	644
TARGET_OPTION_OVERRIDE	692	TARGET_SCHED_DEPENDENCIES_	
TARGET_OPTION_POST_STREAM_IN	692	EVALUATION_HOOK	641
TARGET_OPTION_PRAGMA_PARSE	692	TARGET_SCHED_DFA_NEW_CYCLE	643
TARGET_OPTION_PRINT	692	TARGET_SCHED_DFA_POST_ADVANCE_CYCLE	642
TARGET_OPTION_RESTORE	691	TARGET_SCHED_DFA_POST_CYCLE_INSN	642
TARGET_OPTION_SAME_FUNCTION_VERSIONS	692	TARGET_SCHED_DFA_PRE_ADVANCE_CYCLE	642
TARGET_OPTION_SAVE	691	TARGET_SCHED_DFA_PRE_CYCLE_INSN	641
TARGET_OPTION_VALID_ATTRIBUTE_P	691	TARGET_SCHED_DISPATCH	645
TARGET_OPTION_VALID_		TARGET_SCHED_DISPATCH_DO	645
VERSION_ATTRIBUTE_P	691	TARGET_SCHED_EXPOSED_PIPELINE	645
TARGET_OS_CPP_BUILTINS	537	TARGET_SCHED_FINISH	641
TARGET_OUTPUT_CFI_DIRECTIVE	680	TARGET_SCHED_FINISH_GLOBAL	641
TARGET_OVERLAP_OP_BY_PIECES_P	635	TARGET_SCHED_FIRST_CYCLE_	
TARGET_OVERRIDE_OPTIONS_AFTER_CHANGE	539	MULTIPASS_BACKTRACK	643
TARGET_OVERRIDES_FORMAT_ATTRIBUTES	718	TARGET_SCHED_FIRST_CYCLE_	
TARGET_OVERRIDES_FORMAT_		MULTIPASS_BEGIN	643
ATTRIBUTES_COUNT	718	TARGET_SCHED_FIRST_CYCLE_	
TARGET_OVERRIDES_FORMAT_INIT	718	MULTIPASS_DFA_LOOKAHEAD	642
TARGET_PASS_BY_REFERENCE	591	TARGET_SCHED_FIRST_CYCLE_MULTIPASS_	
TARGET_PCH_VALID_P	695	DFA_LOOKAHEAD_GUARD	643
TARGET_POSIX_IO	710	TARGET_SCHED_FIRST_CYCLE_MULTIPASS_END	643
TARGET_PRECOMPUTE_TLS_P	620	TARGET_SCHED_FIRST_CYCLE_	
TARGET_PREDICT_DOLoop_P	713	MULTIPASS_FINI	643
TARGET_PREFERRED_DOLoop_MODE	714		

TARGET_SCHED_FIRST_CYCLE_		
MULTIPASS_INIT	643	
TARGET_SCHED_FIRST_CYCLE_		
MULTIPASS_ISSUE	643	
TARGET_SCHED_FREE_SCHED_CONTEXT	644	
TARGET_SCHED_FUSION_PRIORITY	646	
TARGET_SCHED_GEN_SPEC_CHECK	645	
TARGET_SCHED_H_I_D_EXTENDED	644	
TARGET_SCHED_INIT	641	
TARGET_SCHED_INIT_DFA_POST_CYCLE_INSN	642	
TARGET_SCHED_INIT_DFA_PRE_CYCLE_INSN	642	
TARGET_SCHED_INIT_GLOBAL	641	
TARGET_SCHED_INIT_SCHED_CONTEXT	644	
TARGET_SCHED_IS_COSTLY_DEPENDENCE	644	
TARGET_SCHED_ISSUE_RATE	639	
TARGET_SCHED_MACRO_FUSION_P	641	
TARGET_SCHED_MACRO_FUSION_PAIR_P	641	
TARGET_SCHED_NEEDS_BLOCK_P	645	
TARGET_SCHED_REASSOCIATION_WIDTH	646	
TARGET_SCHED_REORDER	640	
TARGET_SCHED_REORDER2	640	
TARGET_SCHED_SET_SCHED_CONTEXT	644	
TARGET_SCHED_SET_SCHED_FLAGS	645	
TARGET_SCHED_SMS_RES_MII	645	
TARGET_SCHED_SPECULATE_INSN	644	
TARGET_SCHED_VARIABLE_ISSUE	639	
TARGET_SECONDARY_MEMORY_NEEDED	570	
TARGET_SECONDARY_MEMORY_NEEDED_MODE	571	
TARGET_SECONDARY_RELOAD	568	
TARGET_SECTION_TYPE_FLAGS	656	
TARGET_SELECT_EARLY_REMAT_MODES	571	
TARGET_SET_CURRENT_FUNCTION	715	
TARGET_SET_DEFAULT_TYPE_ATTRIBUTES	689	
TARGET_SET_UP_BY_PROLOGUE	606	
TARGET_SETJMP_PRESERVES_		
NONVOLATILE_REGS_P	705	
TARGET_SETUP_INCOMING_VARARGS	609	
TARGET_SHIFT_TRUNCATION_MASK	704	
TARGET_SHRINK_WRAP_COMPONENTS_FOR_BB	607	
TARGET_SHRINK_WRAP_		
DISQUALIFY_COMPONENTS	607	
TARGET_SHRINK_WRAP_EMIT_		
EPILOGUE_COMPONENTS	607	
TARGET_SHRINK_WRAP_EMIT_		
PROLOGUE_COMPONENTS	607	
TARGET_SHRINK_WRAP_GET_		
SEPARATE_COMPONENTS	606	
TARGET_SHRINK_WRAP_SET_		
HANDLED_COMPONENTS	607	
TARGET_SIMD_CLONE_ADJUST	625	
TARGET_SIMD_CLONE_COMPUTE_		
VECSIZE_AND_SIMDLEN	625	
TARGET_SIMD_CLONE_USABLE	625	
TARGET_SIMT_VF	625	
TARGET_SLOW_UNALIGNED_ACCESS	634	
TARGET_SMALL_REGISTER_		
CLASSES_FOR_MODE_P	597	
TARGET_SPECULATION_SAFE_VALUE	722	
TARGET_SPILL_CLASS	574	
TARGET_SPLIT_COMPLEX_ARG	594	
TARGET_STACK_CLASH_PROTECTION_		
ALLOCA_PROBE_RANGE	583	
TARGET_STACK_PROTECT_FAIL	607	
TARGET_STACK_PROTECT_GUARD	607	
TARGET_STACK_PROTECT_RUNTIME_ENABLED_P	607	
TARGET_START_CALL_ARGS	610	
TARGET_STARTING_FRAME_OFFSET	575	
TARGET_STARTING_FRAME_OFFSET and		
virtual registers	307	
TARGET_STATIC_CHAIN	584	
TARGET_STATIC_RTX_ALIGNMENT	545	
TARGET_STRICT_ARGUMENT_NAMING	610	
TARGET_STRING_OBJECT_REF_TYPE_P	538	
TARGET_STRIP_NAME_ENCODING	652	
TARGET_STRUB_MAY_USE_MEMSET	579	
TARGET_STRUB_USE_DYNAMIC_ARRAY	579	
TARGET_STRUCT_VALUE_RTX	600	
TARGET_SUPPORTS_SPLIT_STACK	608	
TARGET_SUPPORTS_WEAK	665	
TARGET_SUPPORTS_WIDE_INT	721	
TARGET_TERMINATE_DW2_EH_FRAME_INFO	679	
TARGET_TRAMPOLINE_ADJUST_ADDRESS	613	
TARGET_TRAMPOLINE_INIT	613	
TARGET_TRANSLATE_MODE_ATTRIBUTE	595	
TARGET_TRULY_NOOP_TRUNCATION	704	
TARGET_UNSPEC_MAY_TRAP_P	715	
TARGET_UNWIND_TABLES_DEFAULT	679	
TARGET_UNWIND_WORD_MODE	550	
TARGET_UPDATE_IPA_FN_TARGET_INFO	693	
TARGET_UPDATE_STACK_BOUNDARY	719	
TARGET_USE_ANCHORS_FOR_SYMBOL_P	628	
TARGET_USE_BLOCKS_FOR_CONSTANT_P	620	
TARGET_USE_BLOCKS_FOR_DECL_P	620	
TARGET_USE_BY_PIECES_INFRASTRUCTURE_P	634	
TARGET_USE_LATE_PROLOGUE_EPILOGUE	711	
TARGET_USE_PSEUDO_PIC_REG	591	
TARGET_USES_WEAK_UNWIND_INFO	581	
TARGET_VALID_DLLIMPORT_ATTRIBUTE_P	690	
TARGET_VALID_POINTER_MODE	594	
TARGET_VECTOR_ALIGNMENT	546	
TARGET_VECTOR_MODE_		
SUPPORTED_ANY_TARGET_P	595	
TARGET_VECTOR_MODE_SUPPORTED_P	595	
TARGET_VECTORIZE_		
AUTOVECTORIZE_VECTOR_MODES	623	
TARGET_VECTORIZE_BUILTIN_GATHER	624	
TARGET_VECTORIZE_BUILTIN_MASK_FOR_LOAD	621	
TARGET_VECTORIZE_BUILTIN_MD_		
VECTORIZED_FUNCTION	622	
TARGET_VECTORIZE_BUILTIN_SCATTER	625	
TARGET_VECTORIZE_BUILTIN_		
VECTORIZATION_COST	621	
TARGET_VECTORIZE_BUILTIN_		
VECTORIZED_FUNCTION	622	
TARGET_VECTORIZE_CONDITIONAL_		
OPERATION_IS_EXPENSIVE	624	

TARGET_VECTORIZE_CREATE_COSTS	624	tree_fits_shwi_p	192
TARGET_VECTORIZE_EMPTY_		tree_fits_uhwi_p	192
MASK_IS_EXPENSIVE	624	tree_int_cst_equal	192
TARGET_VECTORIZE_GET_MASK_MODE	624	tree_int_cst_lt	192
TARGET_VECTORIZE_PREFER_GATHER_SCATTER...	625	tree_size	180
TARGET_VECTORIZE_PREFERRED_DIV_AS_		tree_to_shwi	192
SHIFTS_OVER_MULT	622	tree_to_uhwi	192
TARGET_VECTORIZE_PREFERRED_SIMD_MODE	623	TREE_CHAIN	180
TARGET_VECTORIZE_PREFERRED_		TREE_CODE	179
VECTOR_ALIGNMENT	621	TREE_INT_CST_ELT	192
TARGET_VECTORIZE_RELATED_MODE	623	TREE_INT_CST_LOW	192
TARGET_VECTORIZE_SPLIT_REDUCTION	623	TREE_INT_CST_NUNITS	192
TARGET_VECTORIZE_SUPPORT_		TREE_LIST	181
VECTOR_MISALIGNMENT	622	TREE_OPERAND	191
TARGET_VECTORIZE_VEC_PERM_CONST	622	TREE_PUBLIC	215, 217
TARGET_VECTORIZE_VECTOR_		TREE_PURPOSE	181
ALIGNMENT_REACHABLE	621	TREE_READONLY	217
TARGET_VERIFY_TYPE_CONTEXT	719	TREE_STATIC	217
TARGET_VTABLE_DATA_ENTRY_DISTANCE	556	TREE_STRING_LENGTH	192
TARGET_VTABLE_ENTRY_ALIGN	556	TREE_STRING_POINTER	192
TARGET_VTABLE_USES_DESCRIPTOR	555	TREE_THIS_VOLATILE	217
TARGET_WANT_DEBUG_PUB_SECTIONS	684	TREE_TYPE	180, 182, 186, 191, 216, 219
TARGET_WARN_FUNC_RETURN	606	TREE_VALUE	181
TARGET_WARN_PARAMETER_PASSING_ABI	601	TREE_VEC	181
TARGET_WEAK_NOT_IN_ARCHIVE_TOC	665	TREE_VEC_ELT	181
TARGET_ZERO_CALL_USED_REGS	719	TREE_VEC_LENGTH	181
targetm	529	true positive	785
targets, makefile	68	TRUNC_DIV_EXPR	195
tbranch_opmode3 instruction pattern	464	TRUNC_MOD_EXPR	195
TCmode	298	truncate	319
TDmode	296	truncmn2 instruction pattern	458
TEMPLATE_DECL	186	TRUTH_AND_EXPR	195
Temporaries	238	TRUTH_ANDIF_EXPR	195
termination routines	668	TRUTH_NOT_EXPR	195
testing constraints	425	TRUTH_OR_EXPR	195
TEXT_SECTION_ASM_OP	647	TRUTH_ORIF_EXPR	195
TFmode	296	TRUTH_XOR_EXPR	195
The Language	766	TRY_BLOCK	226
THEN_CLAUSE	226	TRY_HANDLERS	226
THREAD_MODEL_SPEC	533	TRY_STMTS	226
THROW_EXPR	195	Tuple specific accessors	245
THUNK_DECL	186	tuples	232
THUNK_DELTA	186	type	182
TImode	296	type declaration	186
TImode, in insn	333	TYPE_ALIGN	182, 219
TLS_COMMON_ASM_OP	648	TYPE_ARG_TYPES	182, 219
TLS_SECTION_ASM_FLAG	648	TYPE_ASM_OP	662
tm.h macros	529	TYPE_ATTRIBUTES	191
TQFmode	296	TYPE_BINFO	222
TQmode	297	TYPE_BUILT_IN	220
TRAMPOLINE_ALIGNMENT	613	TYPE_CANONICAL	182
TRAMPOLINE_SECTION	613	TYPE_CONTEXT	182, 219
TRAMPOLINE_SIZE	613	TYPE_DECL	186
trampolines for nested functions	611	TYPE_FIELDS	182, 219, 222
TRANSFER_FROM_TRAMPOLINE	614	TYPE_HAS_ARRAY_NEW_OPERATOR	223
trap instruction pattern	472	TYPE_HAS_DEFAULT_CONSTRUCTOR	223
tree	179, 180	TYPE_HAS_MUTABLE_P	223
Tree SSA	271	TYPE_HAS_NEW_OPERATOR	223

TYPE_MAIN_VARIANT 182, 219
 TYPE_MAX_VALUE 182
 TYPE_METHOD_BASETYPE 182, 219
 TYPE_MIN_VALUE 182
 TYPE_NAME 182, 219
 TYPE_NOTHROW_P 226
 TYPE_OFFSET_BASETYPE 182, 219
 TYPE_OPERAND_FMT 662
 TYPE_OVERLOADS_ARRAY_REF 223
 TYPE_OVERLOADS_ARROW 223
 TYPE_OVERLOADS_CALL_EXPR 223
 TYPE_POLYMORPHIC_P 223
 TYPE_PRECISION 182, 219
 TYPE_PTR_P 220
 TYPE_PTRDATAMEM_P 219, 220
 TYPE_PTRFN_P 220
 TYPE_PTROB_P 220
 TYPE_PTROBV_P 219
 TYPE_QUAL_CONST 182, 219
 TYPE_QUAL_RESTRICT 182, 219
 TYPE_QUAL_VOLATILE 182, 219
 TYPE_RAISES_EXCEPTIONS 226
 TYPE_SIZE 182, 219
 TYPE_STRUCTURAL_EQUALITY_P 182, 183
 TYPE_UNQUALIFIED 182, 219
 TYPE_VFIELD 222
 TYPENAME_TYPE 219
 TYPENAME_TYPE_FULLNAME 182, 219
 TYPEOF_TYPE 219

U

uabdm3 instruction pattern 446
 uaddcm5 instruction pattern 437
 uaddvm4 instruction pattern 437
 uavgm3_ceil instruction pattern 446
 uavgm3_floor instruction pattern 446
 UDAmode 297
 udiv 314
 udivm3 instruction pattern 436
 udivmodm4 instruction pattern 446
 udot_prodmn instruction pattern 440
 UDQmode 297
 UHAmode 297
 UHQmode 297
 UINT_FAST16_TYPE 555
 UINT_FAST32_TYPE 555
 UINT_FAST64_TYPE 555
 UINT_FAST8_TYPE 554
 UINT_LEAST16_TYPE 554
 UINT_LEAST32_TYPE 554
 UINT_LEAST64_TYPE 554
 UINT_LEAST8_TYPE 554
 UINT16_TYPE 554
 UINT32_TYPE 554
 UINT64_TYPE 554
 UINT8_TYPE 554
 UINTMAX_TYPE 554

UINTPTR_TYPE 555
 umaddmn4 instruction pattern 445
 umax 314
 umaxm3 instruction pattern 436
 umin 314
 uminm3 instruction pattern 436
 umod 314
 umodm3 instruction pattern 436
 umsubmn4 instruction pattern 445
 umul_highpart 314
 umulhisi3 instruction pattern 444
 umulhrsm3 instruction pattern 441
 umulhsm3 instruction pattern 441
 umulm3_highpart instruction pattern 445
 umulqih3 instruction pattern 444
 umulsidi3 instruction pattern 444
 umulvm4 instruction pattern 437
 unchanging 294
 unchanging, in call_insn 291
 unchanging, in jump_insn,
 call_insn and insn 290
 unchanging, in mem 291
 unchanging, in subreg 293
 unchanging, in symbol_ref 290
 UNEQ_EXPR 195
 UNGE_EXPR 195
 UNGT_EXPR 195
 UNION_TYPE 182, 222
 UNITS_PER_WORD 542
 UNKNOWN_TYPE 182, 219
 UNLE_EXPR 195
 UNLIKELY_EXECUTED_TEXT_SECTION_NAME 648
 UNLT_EXPR 195
 UNORDERED_EXPR 195
 unshare_all_rtl 348
 unsigned division 314
 unsigned division with unsigned saturation 314
 unsigned greater than 317
 unsigned less than 317
 unsigned minimum and maximum 314
 unsigned_fix 320
 unsigned_float 320
 unsigned_fract_convert 320
 unsigned_sat_fract 320
 unspec 325, 520
 unspec_volatile 325, 520
 untyped_call instruction pattern 466
 untyped_return instruction pattern 467
 update_ssa 278
 update_stmt 244, 271
 update_stmt_if_modified 244
 UPDATE_PATH_HOST_CANONICALIZE (path) 726
 UQQmode 297
 us_ashift 315
 us_minus 312
 us_mult 313
 us_neg 313
 us_plus 312

<code>us_truncate</code>	319
<code>usaddm3</code> instruction pattern.....	436
<code>usadm</code> instruction pattern.....	440
<code>USAmode</code>	297
<code>usashlm3</code> instruction pattern.....	446
<code>usdivm3</code> instruction pattern.....	436
<code>usdot_prodmn</code> instruction pattern.....	440
<code>use</code>	323
<code>USE_C_ALLOCA</code>	727
<code>USE_LD_AS_NEEDED</code>	532
<code>USE_LOAD_POST_DECREMENT</code>	636
<code>USE_LOAD_POST_INCREMENT</code>	636
<code>USE_LOAD_PRE_DECREMENT</code>	636
<code>USE_LOAD_PRE_INCREMENT</code>	636
<code>USE_SELECT_SECTION_FOR_FUNCTIONS</code>	650
<code>USE_STORE_POST_DECREMENT</code>	636
<code>USE_STORE_POST_INCREMENT</code>	636
<code>USE_STORE_PRE_DECREMENT</code>	636
<code>USE_STORE_PRE_INCREMENT</code>	636
<code>used</code>	295
<code>used</code> , in <code>symbol_ref</code>	293
<code>user</code>	742
user experience guidelines.....	785
<code>user gc</code>	743
<code>USER_LABEL_PREFIX</code>	675
<code>USING_STMT</code>	226
<code>usmaddmn4</code> instruction pattern.....	445
<code>usmsubmn4</code> instruction pattern.....	445
<code>usmulhisi3</code> instruction pattern.....	444
<code>usmulm3</code> instruction pattern.....	436
<code>usmulqih3</code> instruction pattern.....	444
<code>usmulsidi3</code> instruction pattern.....	444
<code>usnegm2</code> instruction pattern.....	447
<code>USQmode</code>	297
<code>ussubm3</code> instruction pattern.....	436
<code>ustruncmn2</code> instruction pattern.....	437
<code>usubcm5</code> instruction pattern.....	437
<code>usubvm4</code> instruction pattern.....	437
<code>UTAmode</code>	297
<code>UTQmode</code>	297

V

‘V’ in constraint.....	385
<code>VA_ARG_EXPR</code>	195
values, returned by functions.....	598
<code>var_location</code>	328
<code>VAR_DECL</code>	186
varargs implementation.....	608
variable.....	186
Variable Location Debug	
Information in RTL.....	328
<code>vashlm3</code> instruction pattern.....	446
<code>vashrm3</code> instruction pattern.....	446
<code>vcond_mask_len_mn</code> instruction pattern.....	434
<code>vcond_mask_mn</code> instruction pattern.....	434
<code>vec_addsubm3</code> instruction pattern.....	444
<code>vec_cbranch_allmode</code> instruction pattern.....	465

<code>vec_cbranch_anymode</code> instruction pattern.....	465
<code>vec_cmpeqmn</code> instruction pattern.....	433
<code>vec_cmpmn</code> instruction pattern.....	433
<code>vec_cmpumn</code> instruction pattern.....	433
<code>vec_concat</code>	318
<code>vec_duplicate</code>	319
<code>vec_duplicatem</code> instruction pattern.....	432
<code>vec_extractmn</code> instruction pattern.....	432
<code>vec_fmaddsubm4</code> instruction pattern.....	444
<code>vec_fmsubaddm4</code> instruction pattern.....	444
<code>vec_initmn</code> instruction pattern.....	432
<code>vec_load_lanesmn</code> instruction pattern.....	428
<code>vec_mask_len_load_lanesmn</code> instruction pattern.....	429
<code>vec_mask_len_store_lanesmn</code> instruction pattern.....	430
<code>vec_mask_load_lanesmn</code> instruction pattern...	429
<code>vec_mask_store_lanesmn</code> instruction pattern.....	429
<code>vec_merge</code>	318
<code>vec_merge</code> , canonicalization of.....	484
<code>vec_pack_sbool_trunc_m</code> instruction pattern.....	442
<code>vec_pack_sfix_trunc_m</code> instruction pattern...	442
<code>vec_pack_ssat_m</code> instruction pattern.....	442
<code>vec_pack_trunc_m</code> instruction pattern.....	441
<code>vec_pack_ufix_trunc_m</code> instruction pattern...	442
<code>vec_pack_usat_m</code> instruction pattern.....	442
<code>vec_packs_float_m</code> instruction pattern.....	442
<code>vec_packu_float_m</code> instruction pattern.....	442
<code>vec_permm</code> instruction pattern.....	436, 622
<code>vec_select</code>	318
<code>vec_series</code>	319
<code>vec_seriesm</code> instruction pattern.....	432
<code>vec_setm</code> instruction pattern.....	432
<code>vec_shl_insert_m</code> instruction pattern.....	441
<code>vec_shl_m</code> instruction pattern.....	441
<code>vec_shr_m</code> instruction pattern.....	441
<code>vec_store_lanesmn</code> instruction pattern.....	429
<code>vec_trunc_add_highm</code> instruction pattern.....	444
<code>vec_unpack_sfix_trunc_hi_m</code> instruction pattern.....	443
<code>vec_unpack_sfix_trunc_lo_m</code> instruction pattern.....	443
<code>vec_unpack_ufix_trunc_hi_m</code> instruction pattern.....	443
<code>vec_unpack_ufix_trunc_lo_m</code> instruction pattern.....	443
<code>vec_unpacks_float_hi_m</code> instruction pattern.....	443
<code>vec_unpacks_float_lo_m</code> instruction pattern.....	443
<code>vec_unpacks_hi_m</code> instruction pattern.....	442
<code>vec_unpacks_lo_m</code> instruction pattern.....	442
<code>vec_unpacks_sbool_hi_m</code> instruction pattern.....	442
<code>vec_unpacks_sbool_lo_m</code> instruction pattern.....	442

vec_unpacku_float_hi_m
 instruction pattern 443
 vec_unpacku_float_lo_m
 instruction pattern 443
 vec_unpacku_hi_m instruction pattern 442
 vec_unpacku_lo_m instruction pattern 442
 vec_widen_sabd_even_m instruction pattern... 443
 vec_widen_sabd_hi_m instruction pattern.... 443
 vec_widen_sabd_lo_m instruction pattern.... 443
 vec_widen_sabd_odd_m instruction pattern.... 443
 vec_widen_saddl_hi_m instruction pattern.... 443
 vec_widen_saddl_lo_m instruction pattern.... 443
 vec_widen_smult_even_m
 instruction pattern 443
 vec_widen_smult_hi_m instruction pattern.... 443
 vec_widen_smult_lo_m instruction pattern.... 443
 vec_widen_smult_odd_m instruction pattern... 443
 vec_widen_sshiftl_hi_m
 instruction pattern 443
 vec_widen_sshiftl_lo_m
 instruction pattern 443
 vec_widen_ssubl_hi_m instruction pattern.... 443
 vec_widen_ssubl_lo_m instruction pattern.... 443
 vec_widen_uabd_even_m instruction pattern... 443
 vec_widen_uabd_hi_m instruction pattern.... 443
 vec_widen_uabd_lo_m instruction pattern.... 443
 vec_widen_uabd_odd_m instruction pattern.... 443
 vec_widen_uaddl_hi_m instruction pattern.... 443
 vec_widen_uaddl_lo_m instruction pattern.... 443
 vec_widen_umult_even_m
 instruction pattern 443
 vec_widen_umult_hi_m instruction pattern.... 443
 vec_widen_umult_lo_m instruction pattern.... 443
 vec_widen_umult_odd_m instruction pattern... 443
 vec_widen_ushiftl_hi_m
 instruction pattern 443
 vec_widen_ushiftl_lo_m
 instruction pattern 443
 vec_widen_usubl_hi_m instruction pattern.... 443
 vec_widen_usubl_lo_m instruction pattern.... 443
 VEC_COND_EXPR 203
 VEC_DUPLICATE_EXPR 203
 VEC_LSHIFT_EXPR 203
 VEC_PACK_FIX_TRUNC_EXPR 203
 VEC_PACK_FLOAT_EXPR 203
 VEC_PACK_SAT_EXPR 203
 VEC_PACK_TRUNC_EXPR 203
 VEC_RSHIFT_EXPR 203
 VEC_SERIES_EXPR 203
 VEC_UNPACK_FIX_TRUNC_HI_EXPR 203
 VEC_UNPACK_FIX_TRUNC_LO_EXPR 203
 VEC_UNPACK_FLOAT_HI_EXPR 203
 VEC_UNPACK_FLOAT_LO_EXPR 203
 VEC_UNPACK_HI_EXPR 203
 VEC_UNPACK_LO_EXPR 203
 VEC_WIDEN_MULT_HI_EXPR 203
 VEC_WIDEN_MULT_LO_EXPR 203
 vector 181

vector operations 318
 VECTOR_CST 192
 VECTOR_STORE_FLAG_VALUE 706
 Vectorization 621
 verify_flow_info 357
 virtual operands 271
 VIRTUAL_INCOMING_ARGS_REGNUM 307
 VIRTUAL_OUTGOING_ARGS_REGNUM 307
 VIRTUAL_STACK_DYNAMIC_REGNUM 307
 VIRTUAL_STACK_VARS_REGNUM 307
 VLIW 508, 512
 vlshrm3 instruction pattern 446
 VMS 726
 VMS_DEBUGGING_INFO 685
 VOID_TYPE 182
 VOIDmode 298
 volatil 295
 volatil, in insn, call_insn, jump_insn,
 code_label, jump_table_data, barrier, and
 note 290
 volatil, in label_ref and reg_label 290
 volatil, in mem, asm_operands,
 and asm_input 291
 volatil, in reg 291
 volatil, in subreg 293
 volatil, in symbol_ref 293
 volatile memory references 295
 volatile, in prefetch 291
 voting between constraint alternatives 390
 vrotrm3 instruction pattern 446
 vrotrm3 instruction pattern 446

W

walk_dominator_tree 279
 walk_gimple_op 269
 walk_gimple_seq 269
 walk_gimple_stmt 268
 WCHAR_TYPE 553
 WCHAR_TYPE_SIZE 554
 which_alternative 377
 while_ultmn instruction pattern 432
 WHILE_BODY 226
 WHILE_COND 226
 WHILE_STMT 226
 whopr 757
 widen_ssumnm3 instruction pattern 440
 widen_usumnm3 instruction pattern 440
 WIDEN_MULT_EXPR 195
 WIDEST_HARDWARE_FP_SIZE 553
 window_save instruction pattern 472
 WINT_TYPE 554
 word_mode 302
 WORD_REGISTER_OPERATIONS 702
 WORDS_BIG_ENDIAN 541
 WORDS_BIG_ENDIAN, effect on subreg 310
 wpa 757

X

<code>x-host</code>	732
'X' in constraint	386
<code>XCmode</code>	298
<code>XEXP</code>	286
<code>XFmode</code>	296
<code>XImode</code>	296
<code>XINT</code>	286
<code>xm-machine.h</code>	726, 727
<code>xor</code>	315
<code>xor</code> , canonicalization of	485
<code>xorm3</code> instruction pattern	436

<code>xorsignm3</code> instruction pattern	451
<code>XSTR</code>	286
<code>XVEC</code>	287
<code>XVECEXP</code>	287
<code>XVECLEN</code>	287
<code>XWINT</code>	286

Z

<code>zero_extend</code>	319
<code>zero_extendmn2</code> instruction pattern	458
<code>zero_extract</code>	318
<code>zero_extract</code> , canonicalization of	485