Next: RL78 Options, Previous: PRU Options, Up: Machine-Dependent Options [Contents][Index]

### 3.19.40 RISC-V Options

These command-line options are defined for RISC-V targets:

`-mbranch-cost=`*n*

Set the cost of branches to roughly *n* instructions.

`-mplt`
`-mno-plt`

When generating PIC code, do or don't allow the use of PLTs. Ignored for non-PIC. The default is `-mplt`.

`-mabi=`*ABI-string*

Specify integer and floating-point calling convention. *ABI-string* contains two parts: the size of integer types and the registers used for floating-point types. For example '`-march=rv64ifd -mabi=lp64d`' means that '`long`' and pointers are 64-bit (implicitly defining '`int`' to be 32-bit), and that floating-point values up to 64 bits wide are passed in F registers. Contrast this with '`-march=rv64ifd -mabi=lp64f`', which still allows the compiler to generate code that uses the F and D extensions but only allows floating-point values up to 32 bits long to be passed in registers; or '`-march=rv64ifd -mabi=lp64`', in which no floating-point arguments will be passed in registers.

The default for this argument is system dependent, users who want a specific calling convention should specify one explicitly. The valid calling conventions are: '`ilp32`', '`ilp32f`', '`ilp32d`', '`lp64`', '`lp64f`', and '`lp64d`'. Some calling conventions are impossible to implement on some ISAs: for example, '`-march=rv32if -mabi=ilp32d`' is invalid because the ABI requires 64-bit values be passed in F registers, but F registers are only 32 bits wide. There are also the '`ilp32e`' ABI that can only be used with the '`rv32e`' architecture and the '`lp64e`' ABI that can only be used with the '`rv64e`'. Those ABIs are not well specified at present, and are subject to change.

`-mfdiv`
`-mno-fdiv`

Do or don't use hardware floating-point divide and square root instructions. This requires the F or D extensions for floating-point registers. The default is to use them if the specified architecture has these instructions.

`-mfence-tso`
`-mno-fence-tso`

Do or don't use the '`fence.tso`' instruction, which is unimplemented on some processors (including those from T-Head). If the '`fence.tso`' instruction is not availiable then a stronger fence will be used instead.

`-mdiv`
`-mno-div`

Do or don't use hardware instructions for integer division. This requires the M extension. The default is to use them if the specified architecture has these instructions.

`-misa-spec=`*`ISA-spec-string`*

Specify the version of the RISC-V Unprivileged (formerly User-Level) ISA specification to produce code conforming to. The possibilities for *ISA-spec-string* are:

`2.2`

Produce code conforming to version 2.2.

`20190608`

Produce code conforming to version 20190608.

`20191213`

Produce code conforming to version 20191213.

The default is `-misa-spec=20191213` unless GCC has been configured with `--with-isa-spec=` specifying a different default version.

`-march=`*`ISA-string`*

Generate code for given RISC-V ISA (e.g. ‘`rv64im`’). ISA strings must be lower-case. Examples include ‘`rv64i`’, ‘`rv32g`’, ‘`rv32e`’, and ‘`rv32imaf`’. Additionally, a special value `help` (`-march=help`) is accepted to list all supported extensions.

The syntax of the ISA string is defined as follows:

`The string must start with ‘rv32’ or ‘rv64’, followed by`

‘`i`’, ‘`e`’, or ‘`g`’, referred to as the base ISA.

`The subsequent part of the string is a list of extension names. Extension`

names can be categorized as multi-letter (e.g. ‘`zba`’) and single-letter (e.g. ‘`v`’). Single-letter extensions can appear consecutively, but multi-letter extensions must be separated by underscores.

`An underscore can appear anywhere after the base ISA. It has no specific`

effect but is used to improve readability and can act as a separator.

`Extension names may include an optional version number, following the`

syntax ‘`<major>p<minor>`’ or ‘`<major>`’, (e.g. ‘`m2p1`’ or ‘`m2`’).

Supported extension are listed below:

| Extension Name | Supported Version | Description |
|---|---|---|
| i | 2.0, 2.1 | Base integer extension. |
| e | 2.0 | Reduced base integer extension. |
| g | - | General-purpose computing base extension, ‘`g`’ will expand to ‘`i`’, ‘`m`’, ‘`a`’, ‘`f`’, ‘`d`’, ‘`zicsr`’ and ‘`zifencei`’. |
| m | 2.0 | Integer multiplication and division extension. |

| Extension Name | Supported Version | Description |
|---|---|---|
| a | 2.0, 2.1 | Atomic extension. |
| f | 2.0, 2.2 | Single-precision floating-point extension. |
| d | 2.0, 2.2 | Double-precision floating-point extension. |
| c | 2.0 | Compressed extension. |
| h | 1.0 | Hypervisor extension. |
| v | 1.0 | Vector extension. |
| zicsr | 2.0 | Control and status register access extension. |
| zifencei | 2.0 | Instruction-fetch fence extension. |
| zicond | 1.0 | Integer conditional operations extension. |
| za64rs | 1.0 | Reservation set size of 64 bytes. |
| za128rs | 1.0 | Reservation set size of 128 bytes. |
| zawrs | 1.0 | Wait-on-reservation-set extension. |
| zba | 1.0 | Address calculation extension. |
| zbb | 1.0 | Basic bit manipulation extension. |
| zbc | 1.0 | Carry-less multiplication extension. |
| zbs | 1.0 | Single-bit operation extension. |
| zfinx | 1.0 | Single-precision floating-point in integer registers extension. |
| zdinx | 1.0 | Double-precision floating-point in integer registers extension. |
| zhinx | 1.0 | Half-precision floating-point in integer registers extension. |
| zhinxmin | 1.0 | Minimal half-precision floating-point in integer registers extension. |
| zbkb | 1.0 | Cryptography bit-manipulation extension. |
| zbkc | 1.0 | Cryptography carry-less multiply extension. |
| zbkx | 1.0 | Cryptography crossbar permutation extension. |
| zkne | 1.0 | AES Encryption extension. |
| zknd | 1.0 | AES Decryption extension. |
| zknh | 1.0 | Hash function extension. |
| zkr | 1.0 | Entropy source extension. |
| zksed | 1.0 | SM4 block cipher extension. |
| zksh | 1.0 | SM3 hash function extension. |
| zkt | 1.0 | Data independent execution latency extension. |
| zk | 1.0 | Standard scalar cryptography extension. |
| zkn | 1.0 | NIST algorithm suite extension. |
| zks | 1.0 | ShangMi algorithm suite extension. |
| zihintntl | 1.0 | Non-temporal locality hints extension. |

| Extension Name | Supported Version | Description |
|---|---|---|
| zihintpause | 1.0 | Pause hint extension. |
| zicboz | 1.0 | Cache-block zero extension. |
| zicbom | 1.0 | Cache-block management extension. |
| zicbop | 1.0 | Cache-block prefetch extension. |
| zic64b | 1.0 | Cache block size isf 64 bytes. |
| ziccamoa | 1.0 | Main memory supports all atomics in A. |
| ziccif | 1.0 | Main memory supports instruction fetch with atomicity requirement. |
| zicclsm | 1.0 | Main memory supports misaligned loads/stores. |
| ziccrse | 1.0 | Main memory supports forward progress on LR/SC sequences. |
| zicntr | 2.0 | Standard extension for base counters and timers. |
| zihpm | 2.0 | Standard extension for hardware performance counters. |
| ztso | 1.0 | Total store ordering extension. |
| zve32x | 1.0 | Vector extensions for embedded processors. |
| zve32f | 1.0 | Vector extensions for embedded processors. |
| zve64x | 1.0 | Vector extensions for embedded processors. |
| zve64f | 1.0 | Vector extensions for embedded processors. |
| zve64d | 1.0 | Vector extensions for embedded processors. |
| zvl32b | 1.0 | Minimum vector length standard extensions |
| zvl64b | 1.0 | Minimum vector length standard extensions |
| zvl128b | 1.0 | Minimum vector length standard extensions |
| zvl256b | 1.0 | Minimum vector length standard extensions |
| zvl512b | 1.0 | Minimum vector length standard extensions |
| zvl1024b | 1.0 | Minimum vector length standard extensions |
| zvl2048b | 1.0 | Minimum vector length standard extensions |
| zvl4096b | 1.0 | Minimum vector length standard extensions |
| zvbb | 1.0 | Vector basic bit-manipulation extension. |
| zvbc | 1.0 | Vector carryless multiplication extension. |
| zvkb | 1.0 | Vector cryptography bit-manipulation extension. |
| zvkg | 1.0 | Vector GCM/GMAC extension. |
| zvkned | 1.0 | Vector AES block cipher extension. |
| zvknha | 1.0 | Vector SHA-2 secure hash extension. |
| zvknhb | 1.0 | Vector SHA-2 secure hash extension. |
| zvksed | 1.0 | Vector SM4 Block Cipher extension. |
| zvksh | 1.0 | Vector SM3 Secure Hash extension. |

| Extension Name | Supported Version | Description |
|---|---|---|
| zvkn | 1.0 | Vector NIST Algorithm Suite extension, ‘zvkn’ will expand to ‘zvkned’, ‘zvknhb’, ‘zvkb’ and ‘zvkt’. |
| zvknc | 1.0 | Vector NIST Algorithm Suite with carryless multiply extension, ‘zvknc’ will expand to ‘zvkn’ and ‘zvbc’. |
| zvkng | 1.0 | Vector NIST Algorithm Suite with GCM extension, ‘zvkng’ will expand to ‘zvkn’ and ‘zvkg’. |
| zvks | 1.0 | Vector ShangMi algorithm suite extension, ‘zvks’ will expand to ‘zvksed’, ‘zvksh’, ‘zvkb’ and ‘zvkt’. |
| zvksc | 1.0 | Vector ShangMi algorithm suite with carryless multiplication extension, ‘zvksc’ will expand to ‘zvks’ and ‘zvbc’. |
| zvksg | 1.0 | Vector ShangMi algorithm suite with GCM extension, ‘zvksg’ will expand to ‘zvks’ and ‘zvkg’. |
| zvkt | 1.0 | Vector data independent execution latency extension. |
| zfh | 1.0 | Half-precision floating-point extension. |
| zfhmin | 1.0 | Minimal half-precision floating-point extension. |
| zvfh | 1.0 | Vector half-precision floating-point extension. |
| zvfhmin | 1.0 | Vector minimal half-precision floating-point extension. |
| zvfbfmin | 1.0 | Vector BF16 converts extension. |
| zfa | 1.0 | Additional floating-point extension. |
| zmmul | 1.0 | Integer multiplication extension. |
| zca | 1.0 | Integer compressed instruction extension. |
| zcf | 1.0 | Compressed single-precision floating point loads and stores extension. |
| zcd | 1.0 | Compressed double-precision floating point loads and stores extension. |
| zcb | 1.0 | Simple compressed instruction extension. |
| zce | 1.0 | Compressed instruction extensions for embedded processors. |
| zcmp | 1.0 | Compressed push pop extension. |
| zcmt | 1.0 | Table jump instruction extension. |
| smaia | 1.0 | Advanced interrupt architecture extension. |
| smepmp | 1.0 | PMP Enhancements for memory access and execution prevention on Machine mode. |
| smstateen | 1.0 | State enable extension. |
| ssaia | 1.0 | Advanced interrupt architecture extension for supervisor-mode. |
| sscofpmf | 1.0 | Count overflow & filtering extension. |
| ssstateen | 1.0 | State-enable extension for supervisor-mode. |
| sstc | 1.0 | Supervisor-mode timer interrupts extension. |
| svinval | 1.0 | Fine-grained address-translation cache invalidation extension. |

| Extension Name | Supported Version | Description |
|---|---|---|
| svnapot | 1.0 | NAPOT translation contiguity extension. |
| svpbmt | 1.0 | Page-based memory types extension. |
| xcvmac | 1.0 | Core-V multiply-accumulate extension. |
| xcvalu | 1.0 | Core-V miscellaneous ALU extension. |
| xcvelw | 1.0 | Core-V event load word extension. |
| xtheadba | 1.0 | T-head address calculation extension. |
| xtheadbb | 1.0 | T-head basic bit-manipulation extension. |
| xtheadbs | 1.0 | T-head single-bit instructions extension. |
| xtheadcmo | 1.0 | T-head cache management operations extension. |
| xtheadcondmov | 1.0 | T-head conditional move extension. |
| xtheadfmemidx | 1.0 | T-head indexed memory operations for floating-point registers extension. |
| xtheadfmv | 1.0 | T-head double floating-point high-bit data transmission extension. |
| xtheadint | 1.0 | T-head acceleration interruption extension. |
| xtheadmac | 1.0 | T-head multiply-accumulate extension. |
| xtheadmemidx | 1.0 | T-head indexed memory operation extension. |
| xtheadmempair | 1.0 | T-head two-GPR memory operation extension. |
| xtheadsync | 1.0 | T-head multi-core synchronization extension. |
| xventanacondops | 1.0 | Ventana integer conditional operations extension. |

When `-march=` is not specified, use the setting from `-mcpu`.

If both `-march` and `-mcpu=` are not specified, the default for this argument is system dependent, users who want a specific architecture extensions should specify one explicitly.

`-mcpu=`*`processor-string`*

Use architecture of and optimize the output for the given processor, specified by particular CPU name. Permissible values for this option are: ‘`sifive-e20`’, ‘`sifive-e21`’, ‘`sifive-e24`’, ‘`sifive-e31`’, ‘`sifive-e34`’, ‘`sifive-e76`’, ‘`sifive-s21`’, ‘`sifive-s51`’, ‘`sifive-s54`’, ‘`sifive-s76`’, ‘`sifive-u54`’, ‘`sifive-u74`’, ‘`sifive-x280`’, ‘`sifive-xp450`’, ‘`sifive-x670`’.

Note that `-mcpu` does not override `-march` or `-mtune`.

`-mtune=`*`processor-string`*

Optimize the output for the given processor, specified by microarchitecture or particular CPU name. Permissible values for this option are: ‘`rocket`’, ‘`sifive-3-series`’, ‘`sifive-5-series`’, ‘`sifive-7-series`’, ‘`thead-c906`’, ‘`size`’, ‘`sifive-p400-series`’, ‘`sifive-p600-series`’, and all valid options for `-mcpu=`.

When `-mtune=` is not specified, use the setting from `-mcpu`, the default is ‘`rocket`’ if both are not specified.

The ‘size’ choice is not intended for use by end-users. This is used when `-Os` is specified. It overrides the instruction cost info provided by `-mtune=`, but does not override the pipeline info. This helps reduce code size while still giving good performance.

`-mpreferred-stack-boundary=`*num*

Attempt to keep the stack boundary aligned to a 2 raised to *num* byte boundary. If `-mpreferred-stack-boundary` is not specified, the default is 4 (16 bytes or 128-bits).

**Warning:** If you use this switch, then you must build all modules with the same value, including any libraries. This includes the system libraries and startup modules.

`-msmall-data-limit=`*n*

Put global and static data smaller than *n* bytes into a special section (on some targets).

`-msave-restore`
`-mno-save-restore`

Do or don’t use smaller but slower prologue and epilogue code that uses library function calls. The default is to use fast inline prologues and epilogues.

`-mmovcc`
`-mno-movcc`

Do or don’t produce branchless conditional-move code sequences even with targets that do not have specific instructions for conditional operations. If enabled, sequences of ALU operations are produced using base integer ISA instructions where profitable.

`-minline-atomics`
`-mno-inline-atomics`

Do or don’t use smaller but slower subword atomic emulation code that uses libatomic function calls. The default is to use fast inline subword atomics that do not require libatomic.

`-minline-strlen`
`-mno-inline-strlen`

Do or do not attempt to inline strlen calls if possible. Inlining will only be done if the string is properly aligned and instructions for accelerated processing are available. The default is to not inline strlen calls.

`-minline-strcmp`
`-mno-inline-strcmp`

Do or do not attempt to inline strcmp calls if possible. Inlining will only be done if the strings are properly aligned and instructions for accelerated processing are available. The default is to not inline strcmp calls.

The `--param riscv-strcmp-inline-limit=`*n* parameter controls the maximum number of bytes compared by the inlined code. The default value is 64.

`-minline-strncmp`
`-mno-inline-strncmp`

Do or do not attempt to inline strncmp calls if possible. Inlining will only be done if the strings are properly aligned and instructions for accelerated processing are available. The default is to not inline strncmp calls.

The `--param riscv-strcmp-inline-limit=`*n* parameter controls the maximum number of bytes compared by the inlined code. The default value is 64.

`-mshorten-memrefs`
`-mno-shorten-memrefs`

Do or do not attempt to make more use of compressed load/store instructions by replacing a load/store of 'base register + large offset' with a new load/store of 'new base + small offset'. If the new base gets stored in a compressed register, then the new load/store can be compressed. Currently targets 32-bit integer load/stores only.

`-mstrict-align`
`-mno-strict-align`

Do not or do generate unaligned memory accesses. The default is set depending on whether the processor we are optimizing for supports fast unaligned access or not.

`-mcmodel=medlow`

Generate code for the medium-low code model. The program and its statically defined symbols must lie within a single 2 GiB address range and must lie between absolute addresses −2 GiB and +2 GiB. Programs can be statically or dynamically linked. This is the default code model.

`-mcmodel=medany`

Generate code for the medium-any code model. The program and its statically defined symbols must be within any single 2 GiB address range. Programs can be statically or dynamically linked.

The code generated by the medium-any code model is position-independent, but is not guaranteed to function correctly when linked into position-independent executables or libraries.

`-mcmodel=large`

Generate code for a large code model, which has no restrictions on size or placement of symbols.

`-mexplicit-relocs`
`-mno-exlicit-relocs`

Use or do not use assembler relocation operators when dealing with symbolic addresses. The alternative is to use assembler macros instead, which may limit optimization.

`-mrelax`
`-mno-relax`

Take advantage of linker relaxations to reduce the number of instructions required to materialize symbol addresses. The default is to take advantage of linker relaxations.

`-mriscv-attribute`
`-mno-riscv-attribute`

Emit (do not emit) RISC-V attribute to record extra information into ELF objects. This feature requires at least binutils 2.32.

`-mcsr-check`
`-mno-csr-check`

Enables or disables the CSR checking.

`-malign-data=`*`type`*

Control how GCC aligns variables and constants of array, structure, or union types. Supported values for *type* are '`xlen`' which uses x register width as the alignment value, and '`natural`' which uses natural alignment. '`xlen`' is the default.

`-mbig-endian`

Generate big-endian code. This is the default when GCC is configured for a '`riscv64be-*-*`' or '`riscv32be-*-*`' target.

`-mlittle-endian`

Generate little-endian code. This is the default when GCC is configured for a '`riscv64-*-*`' or '`riscv32-*-*`' but not a '`riscv64be-*-*`' or '`riscv32be-*-*`' target.

`-mstack-protector-guard=`*`guard`*
`-mstack-protector-guard-reg=`*`reg`*
`-mstack-protector-guard-offset=`*`offset`*

Generate stack protection code using canary at *guard*. Supported locations are '`global`' for a global canary or '`tls`' for per-thread canary in the TLS block.

With the latter choice the options `-mstack-protector-guard-reg=`*`reg`* and `-mstack-protector-guard-offset=`*`offset`* furthermore specify which register to use as base register for reading the canary, and from what offset from that base register. There is no default register or offset as this is entirely for use within the Linux kernel.

`-mtls-dialect=desc`

Use TLS descriptors as the thread-local storage mechanism for dynamic accesses of TLS variables.

`-mtls-dialect=trad`

Use traditional TLS as the thread-local storage mechanism for dynamic accesses of TLS variables. This is the default.

---

Next: RL78 Options, Previous: PRU Options, Up: Machine-Dependent Options [Contents][Index]