RISC-V Indirect CSR Access (Smcsrind/Sscsrind)

Authors: Beeman Strong, John Hauser

Version 1.0.0, 2024-02-27: Ratified

Table of Contents

Preamble

This document is in the Ratified state

No changes are allowed. Any desired or needed changes can be the subject of a follow-on new extension. Ratified extensions are never revised.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

- Beeman Strong <beeman@rivosinc.com>
- John Hauser <jh.riscv@jhauser.us>
- Greg Favor <gfavor@ventanamicro.com>
- Krste Asanovic <krste@sifive.com>

Chapter 1. Introduction

Smcsrind/Sscsrind is an ISA extension that extends the indirect CSR access mechanism originally defined as part of the Smaia/Ssaia extensions, in order to make it available for use by other extensions without creating an unnecessary dependence on Smaia/Ssaia.

This extension confers two benefits:

1. It provides a means to access an array of registers via CSRs without requiring allocation of large chunks of the limited CSR address space.
2. It enables software to access each of an array of registers by index, without requiring a switch statement with a case for each register.

CSRs are accessed indirectly via this extension using select values, in contrast to being accessed directly using standard CSR numbers. A CSR accessible via one method may or may not be accessible via the other method. Select values are a separate address space from CSR numbers, and from tselect values in the Sdtrig extension. If a CSR is both directly and indirectly accessible, the CSR's select value is unrelated to its CSR number.

Further, Machine-level and Supervisor-level select values are separate address spaces from each other; however, Machine-level and Supervisor-level CSRs with the same select value may be defined by an extension as partial or full aliases with respect to each other. This typically would be done for CSRs that can be delegated from Machine-level to Supervisor-level.

The machine-level extension **Smcsrind** encompasses all added CSRs and all behavior modifications for a hart, over all privilege levels. For a supervisor-level environment, extension **Sscsrind** is essentially the same as Smcsrind except excluding the machine-level CSRs and behavior not directly visible to supervisor level.

Chapter 2. Machine-level CSRs

Number	Privilege	Width	Name	Description
0x350	MRW	XLEN	`miselect`	Machine indirect register select
0x351	MRW	XLEN	`mireg`	Machine indirect register alias
0x352	MRW	XLEN	`mireg2`	Machine indirect register alias 2
0x353	MRW	XLEN	`mireg3`	Machine indirect register alias 3
0x355	MRW	XLEN	`mireg4`	Machine indirect register alias 4
0x356	MRW	XLEN	`mireg5`	Machine indirect register alias 5
0x357	MRW	XLEN	`mireg6`	Machine indirect register alias 6

The `mireg*` CSR numbers are not consecutive because miph is CSR number 0x354.

The CSRs listed in the table above provide a window for accessing register state indirectly. The value of `miselect` determines which register is accessed upon read or write of each of the machine indirect alias CSRs (`mireg*`). `miselect` value ranges are allocated to dependent extensions, which specify the register state accessible via each `miregi` register, for each `miselect` value. `miselect` is a WARL register.

The `miselect` register implements at least enough bits to support all implemented `miselect` values (corresponding to the implemented extensions that utilize `miselect`/`mireg*` to indirectly access register state). The `miselect` register may be read-only zero if there are no extensions implemented that utilize it.

Values of `miselect` with the most-significant bit set (bit XLEN - 1 = 1) are designated only for custom use, presumably for accessing custom registers through the alias CSRs. Values of `miselect` with the most-significant bit clear are designated only for standard use and are reserved until allocated to a standard architecture extension. If XLEN is changed, the most-significant bit of `miselect` moves to the new position, retaining its value from before.

An implementation is not required to support any custom values for `miselect`.

The behavior upon accessing `mireg*` from M-mode, while `miselect` holds a value that is not implemented, is UNSPECIFIED.

It is expected that implementations will typically raise an illegal instruction exception for such accesses, so that, for example, they can be identified as software bugs. Platform specs, profile specs, and/or the Privileged ISA spec may place more restrictions on behavior for such accesses.

Attempts to access `mireg*` while `miselect` holds a number in an allocated and implemented range results in a specific behavior that, for each combination of `miselect` and `miregi`, is defined by the extension to which the `miselect` value is allocated.

Ordinarily, each `miregi` *will access register state, access read-only 0 state, or raise an illegal instruction exception.*

For RV32, if an extension defines an indirectly accessed register as 64 bits wide, it is recommended that the lower 32 bits of the register are accessed through one of `mireg`*,* `mireg2`*, or* `mireg3`*, while the upper 32 bits are accessed through* `mireg4`*,* `mireg5`*, or* `mireg6`*, respectively.*

Six `*ireg*` *registers are defined in order to ensure that the needs of extensions in development are covered, with some room for growth. For example, for an* `siselect` *value associated with counter X,* `sireg`*/*`sireg2` *could be used to access* `mhpmcounterX` */*`mhpmeventX`*, while* `sireg4`*/*`sireg5` *could access* `mhpmcounterXh`*/*`mhpmeventXh`*. Six* `*ireg*` *registers allows for accessing up to 3 CSR arrays per index (*`*iselect`*) with RV32-only CSRs, or up to 6 CSR arrays per index value without RV32-only CSRs.*

Chapter 3. Supervisor-level CSRs

Number	Privilege	Width	Name	Description
0x150	SRW	XLEN	`siselect`	Supervisor indirect register select
0x151	SRW	XLEN	`sireg`	Supervisor indirect register alias
0x152	SRW	XLEN	`sireg2`	Supervisor indirect register alias 2
0x153	SRW	XLEN	`sireg3`	Supervisor indirect register alias 3
0x155	SRW	XLEN	`sireg4`	Supervisor indirect register alias 4
0x156	SRW	XLEN	`sireg5`	Supervisor indirect register alias 5
0x157	SRW	XLEN	`sireg6`	Supervisor indirect register alias 6

The CSRs in the table above are required if S-mode is implemented.

The `siselect` register will support the value range 0..0xFFF at a minimum. A future extension may define a value range outside of this minimum range. Only if such an extension is implemented will `siselect` be required to support larger values.

Requiring a range of 0–0xFFF for `siselect`, even though most or all of the space may be reserved or inaccessible, permits M-mode to emulate indirectly accessed registers in this implemented range, including registers that may be standardized in the future.

Values of `siselect` with the most-significant bit set (bit XLEN - 1 = 1) are designated only for custom use, presumably for accessing custom registers through the alias CSRs. Values of `siselect` with the most-significant bit clear are designated only for standard use and are reserved until allocated to a standard architecture extension. If XLEN is changed, the most-significant bit of `siselect` moves to the new position, retaining its value from before.

The behavior upon accessing `sireg*` from M-mode or S-mode, while `siselect` holds a value that is not implemented at supervisor level, is UNSPECIFIED.

It is recommended that implementations raise an illegal instruction exception for such accesses, to facilitate possible emulation (by M-mode) of these accesses.

An extension is considered not to be implemented at supervisor level if machine level has disabled the extension for S-mode, such as by the settings of certain fields in CSR `menvcfg`, for example.

Otherwise, attempts to access `sireg*` from M-mode or S-mode while `siselect` holds a number in a standard-defined and implemented range result in specific behavior that, for each combination of `siselect` and `siregi`, is defined by the extension to which the `siselect` value is allocated.

Ordinarily, each `siregi` will access register state, access read-only 0 state, or, unless executing in a virtual machine (covered in the next section), raise an illegal instruction exception.

Note that the widths of `siselect` and `sireg*` are always the current XLEN rather than SXLEN. Hence, for example, if MXLEN = 64 and SXLEN = 32, then these registers are 64 bits when the current privilege mode is M (running RV64 code) but 32 bits when the privilege mode is S (RV32 code).

Chapter 4. Virtual Supervisor-level CSRs

Number	Privilege	Width	Name	Description
0x250	HRW	XLEN	vsiselect	Virtual supervisor indirect register select
0x251	HRW	XLEN	vsireg	Virtual supervisor indirect register alias
0x252	HRW	XLEN	vsireg2	Virtual supervisor indirect register alias 2
0x253	HRW	XLEN	vsireg3	Virtual supervisor indirect register alias 3
0x255	HRW	XLEN	vsireg4	Virtual supervisor indirect register alias 4
0x256	HRW	XLEN	vsireg5	Virtual supervisor indirect register alias 5
0x257	HRW	XLEN	vsireg6	Virtual supervisor indirect register alias 6

The CSRs in the table above are required if the hypervisor extension is implemented. These VS CSRs all match supervisor CSRs, and substitute for those supervisor CSRs when executing in a virtual machine (in VS-mode or VU-mode).

The vsiselect register will support the value range 0..0xFFF at a minimum. A future extension may define a value range outside of this minimum range. Only if such an extension is implemented will vsiselect be required to support larger values.

Requiring a range of 0–0xFFF for vsiselect, even though most or all of the space may be reserved or inaccessible, permits a hypervisor to emulate indirectly accessed registers in this implemented range, including registers that may be standardized in the future.

More generally it is recommended that vsiselect and siselect be implemented with the same number of bits. This also avoids creation of a virtualization hole due to observable differences between vsiselect and siselect widths.

Values of vsiselect with the most-significant bit set (bit XLEN - 1 = 1) are designated only for custom use, presumably for accessing custom registers through the alias CSRs. Values of vsiselect with the most-significant bit clear are designated only for standard use and are reserved until allocated to a standard architecture extension. If XLEN is changed, the most-significant bit of vsiselect moves to the new position, retaining its value from before.

For alias CSRs sireg* and vsireg*, the hypervisor extension's usual rules for when to raise a virtual instruction exception (based on whether an instruction is HS-qualified) are not applicable. The rules given in this section for sireg and vsireg apply instead, unless overridden by the requirements specified in the section below, which take precedence over this section when extension Smstateen is also implemented.

A virtual instruction exception is raised for attempts from VS-mode or VU-mode to directly access vsiselect or vsireg*, or attempts from VU-mode to access siselect or sireg*.

The behavior upon accessing vsireg* from M-mode or HS-mode, or accessing sireg* (really vsireg*) from VS-mode, while vsiselect holds a value that is not implemented at HS level, is UNSPECIFIED.

It is recommended that implementations raise an illegal instruction exception for such accesses, to facilitate possible emulation (by M-mode) of these accesses.

Otherwise, while `vsiselect` holds a number in a standard-defined and implemented range, attempts to access `vsireg*` from a sufficiently privileged mode, or to access `sireg*` (really `vsireg*`) from VS-mode, result in specific behavior that, for each combination of `vsiselect` and `vsiregi`, is defined by the extension to which the `vsiselect` value is allocated.

Ordinarily, each `vsiregi` will access register state, access read-only 0 state, or raise an exception (either an illegal instruction exception or, for select accesses from VS-mode, a virtual instruction exception). When `vsiselect` holds a value that is implemented at HS level but not at VS level, attempts to access `sireg*` (really `vsireg*`) from VS-mode will typically raise a virtual instruction exception. But there may be cases specific to an extension where different behavior is more appropriate.

Like `siselect` and `sireg*`, the widths of `vsiselect` and `vsireg*` are always the current XLEN rather than VSXLEN. Hence, for example, if HSXLEN = 64 and VSXLEN = 32, then these registers are 64 bits when accessed by a hypervisor in HS-mode (running RV64 code) but 32 bits for a guest OS in VS-mode (RV32 code).

Chapter 5. Access control by the state-enable CSRs

If extension Smstateen is implemented together with Smcsrind, bit 60 of state-enable register `mstateen0` controls access to `siselect`, `sireg*`, `vsiselect`, and `vsireg*`. When `mstateen0`[60]=0, an attempt to access one of these CSRs from a privilege mode less privileged than M-mode results in an illegal instruction exception. As always, the state-enable CSRs do not affect the accessibility of any state when in M-mode, only in less privileged modes. For more explanation, see the documentation for extension Smstateen.

Other extensions may specify that certain mstateen bits control access to registers accessed indirectly through `siselect` + `sireg*`, and/or `vsiselect` + `vsireg*`. However, regardless of any other mstateen bits, if `mstateen0`[60] = 1, a virtual instruction exception is raised as described in the previous section for all attempts from VS-mode or VU-mode to directly access `vsiselect` or `vsireg*`, and for all attempts from VU-mode to access `siselect` or `sireg*`.

If the hypervisor extension is implemented, the same bit is defined also in hypervisor CSR `hstateen0`, but controls access to only `siselect` and `sireg*` (really `vsiselect` and `vsireg*`), which is the state potentially accessible to a virtual machine executing in VS or VU-mode. When `hstateen0`[60]=0 and `mstateen0`[60]=1, all attempts from VS or VU-mode to access `siselect` or `sireg*` raise a virtual instruction exception, not an illegal instruction exception, regardless of the value of `vsiselect` or any other mstateen bit.

Extension Ssstateen is defined as the supervisor-level view of Smstateen. Therefore, the combination of Sscsrind and Ssstateen incorporates the bit defined above for `hstateen0` but not that for `mstateen0`, since machine-level CSRs are not visible to supervisor level.

> *CSR address space is reserved for a possible future "Sucsrind" extension that extends indirect CSR access to user mode.*