RISC-V State Enable Extension

Smstateen

Version 1.0.0-a28bfae, 2022-02-03: Ratified

Table of Contents

Preamble

Document Source Information

See github.com/riscv/riscv-state-enable for document source. The document version includes the source control tag or commit hash used to produce this document.

Motivation

The implementation of optional RISC-V extensions has the potential to open covert channels between separate user threads, or between separate guest OSes running under a hypervisor. The problem occurs when an extension adds processor state---usually explicit registers, but possibly other forms of state---that the main OS or hypervisor is unaware of (and hence won't context-switch) but that can be modified/written by one user thread or guest OS and perceived/examined/read by another.

For example, the proposed Advanced Interrupt Architecture (AIA) for RISC-V adds to a hart as many as ten supervisor-level CSRs (`siselect`, `sireg`, `stopi`, `sseteipnum`, `sclreipnum`, `sseteienum`, `sclreienum`, `sclaimei`, `sieh`, and `siph`) and provides also the option for hardware to be backward-compatible with older, pre-AIA software. Because an older hypervisor that is oblivious to the AIA will not know to swap any of the AIA's new CSRs on context switches, the registers may then be used as a covert channel between multiple guest OSes that run atop this hypervisor. Although traditional practices might consider such a communication channel harmless, the intense focus on security today argues that a means be offered to plug such channels.

The `f` registers of the RISC-V floating-point extensions and the `v` registers of the vector extension would similarly be potential covert channels between user threads, except for the existence of the FS and VS fields in the `sstatus` register. Even if an OS is unaware of, say, the vector extension and its `v` registers, access to those registers is blocked when the VS field is initialized to zero, either at machine level or by the OS itself initializing `sstatus`.

Obviously, one way to prevent the use of new user-level CSRs as covert channels would be to add to `mstatus` or `sstatus` an "XS" field for each relevant extension, paralleling the V extension's VS field. However, this is not considered a general solution to the problem due to the number of potential future extensions that may add small amounts of state. Even with a 64-bit `sstatus` (necessitating adding `sstatush` for RV32), it is not certain there are enough remaining bits in `sstatus` to accommodate all future user-level extensions. In any event, there is no need to strain `sstatus` (and add `sstatush`) for this purpose. The "enable" flags that are needed to plug covert channels are not generally expected to require swapping on context switches of user threads, making them a less-than-compelling candidate for inclusion in `sstatus`. Hence, a new place is proposed for them instead.

Chapter 1. Proposal

For RV64 harts, this extension adds four new 64-bit CSRs at machine level, listed with their CSR addresses:

`0x30C mstateen0` (Machine State Enable 0)

`0x30D mstateen1`

`0x30E mstateen2`

`0x30F mstateen3`

If supervisor mode is implemented, another four CSRs are defined at supervisor level:

`0x10C sstateen0`

`0x10D sstateen1`

`0x10E sstateen2`

`0x10F sstateen3`

And if the hypervisor extension is implemented, another set of CSRs is added:

`0x60C hstateen0`

`0x60D hstateen1`

`0x60E hstateen2`

`0x60F hstateen3`

For RV32, the registers listed above are 32-bit, and for the machine-level and hypervisor CSRs there is a corresponding set of high-half CSRs for the upper 32 bits of each register:

`0x31C mstateen0h`

`0x31D mstateen1h`

`0x31E mstateen2h`

`0x31F mstateen3h`

`0x61C hstateen0h`

`0x61D hstateen1h`

`0x61E hstateen2h`

`0x61F hstateen3h`

For the supervisor-level `sstateen` registers, high-half CSRs are not added at this time because it is expected the upper 32 bits of these registers will always be zeros, as explained later below.

Each bit of a `stateen` CSR controls less-privileged access to an extension's state, for an extension that was not deemed "worthy" of a full XS field in `sstatus` like the FS and VS fields for the F and V extensions. The number of registers provided at each level is four because it is believed that 4 * 64 = 256 bits for machine and hypervisor levels, and 4 * 32 = 128 bits for supervisor level, will be adequate for many years to come, perhaps for as long as the RISC-V ISA is in use. The exact number four is an attempted compromise between providing too few bits on the one hand and going overboard with CSRs that will never be used on the other. A possible future doubling of the number of `stateen` CSRs is covered later.

The `stateen` registers at each level control access to state at all less-privileged levels, but not at its own level. This is analogous to how the existing `counteren` CSRs control access to performance counter registers. Just as with the `counteren` CSRs, when a `stateen` CSR prevents access to state by less-privileged levels, an attempt in one of those privilege modes to execute an instruction that would read or write the protected state raises an illegal instruction exception, or, if executing in VS or VU mode and the circumstances for a virtual instruction exception apply, raises a virtual instruction exception instead of an illegal instruction exception.

When this extension is not implemented, all state added by an extension is accessible as defined by that extension.

When a `stateen` CSR prevents access to state for a privilege mode, attempting to execute in that privilege mode an instruction that *implicitly* updates the state without reading it may or may not raise an illegal instruction or virtual instruction exception. Such cases must be disambiguated by being explicitly specified one way or the other.

In some cases, the bits of the `stateen` CSRs will have a dual purpose as enables for the ISA extensions that introduce the controlled state.

Each bit of a supervisor-level `sstateen` CSR controls user-level access (from U-mode or VU-mode) to an extension's state. The intention is to allocate the bits of `sstateen` CSRs starting at the least-significant end, bit 0, through to bit 31, and then on to the next-higher-numbered `sstateen` CSR.

For every bit with a defined purpose in an `sstateen` CSR, the same bit is defined in the matching `mstateen` CSR to control access below machine level to the same state. The upper 32 bits of an `mstateen` CSR (or for RV32, the corresponding high-half CSR) control access to state that is inherently inaccessible to user level, so no corresponding enable bits in the supervisor-level `sstateen` CSR are applicable. The intention is to allocate bits for this purpose starting at the most-significant end, bit 63, through to bit 32, and then on to the next-higher `mstateen` CSR. If the rate that bits are being allocated from the least-significant end for `sstateen` CSRs is sufficiently low, allocation from the most-significant end of `mstateen` CSRs may be allowed to encroach on the lower 32 bits before jumping to the next-higher `mstateen` CSR. In that case, the bit positions of "encroaching" bits will remain forever read-only zeros in the matching `sstateen` CSRs.

With the hypervisor extension, the `hstateen` CSRs have identical encodings to the `mstateen` CSRs, except controlling accesses for a virtual machine (from VS and VU modes).

Each standard-defined bit of a `stateen` CSR is WARL and may be read-only zero or one, subject to the following conditions.

Bits in any `stateen` CSR that are defined to control state that a hart doesn't implement are read-only zeros for that hart. Likewise, all reserved bits not yet given a defined meaning are also read-only zeros. For every bit in an `mstateen` CSR that is zero (whether read-only zero or set to zero), the same bit appears as read-only zero in the matching `hstateen` and `sstateen` CSRs. For every bit in an `hstateen` CSR that is zero (whether read-only zero or set to zero), the same bit appears as read-only zero in `sstateen` when accessed in VS-mode.

A bit in a supervisor-level `sstateen` CSR cannot be read-only one unless the same bit is read-only one in the matching `mstateen` CSR and, if it exists, in the matching `hstateen` CSR. A bit in an `hstateen` CSR cannot be read-only one unless the same bit is read-only one in the matching `mstateen` CSR.

On reset, all writable `mstateen` bits are initialized by the hardware to zeros. If machine-level software changes these values, it is responsible for initializing the corresponding writable bits of the hstateen and sstateen CSRs to zeros too. Software at each privilege level should set its respective stateen CSRs to indicate the state it is prepared to allow less-privileged software to access. For OSes and hypervisors, this usually means the state that the OS or hypervisor is prepared to swap on a context switch, or to manage in some other way.

For each mstateen CSR, bit 63 is defined to control access to the matching sstateen and hstateen CSRs. That is, bit 63 of mstateen0 controls access to sstateen0 and hstateen0; bit 63 of mstateen1 controls access to sstateen1 and hstateen1; etc. Likewise, bit 63 of each hstateen correspondingly controls access to the matching sstateen CSR. A hypervisor may need this control over accesses to the sstateen CSRs if it ever must emulate for a virtual machine an extension that is supposed to be affected by a bit in an sstateen CSR. (Even if such emulation is uncommon, it should not be excluded.) Machine-level software needs identical control to be able to emulate the hypervisor extension. (That is, machine level needs control over accesses to the supervisor-level sstateen CSRs in order to emulate the hstateen CSRs, which have such control.)

Bit 63 of each mstateen CSR may be read-only zero only if the hypervisor extension is not implemented and the matching supervisor-level sstateen CSR is all read-only zeros. In that case, machine-level software should emulate attempts to access the affected sstateen CSR from S-mode, ignoring writes and returning zero for reads. Bit 63 of each hstateen CSR is always writable (not read-only).

Initially, the following bits are defined in mstateen0, hstateen0, and sstateen0:

bit 0 - Custom state

bit 1 - fcsr for Zfinx and related extensions (Zdinx, etc.)

Bit 0 controls access to any and all custom state.

(Bit 0 of these registers is not custom state itself; it is a standard field of a standard CSR, either mstateen0, hstateen0, or sstateen0. The requirements that non-standard extensions must meet to be *conforming* are not relaxed due solely to changes in the value of this bit. In particular, if software sets this bit but does not execute any custom instructions or access any custom state, the software must continue to execute as specified by all relevant RISC-V standards, or the hardware is not standard-conforming.)

Bit 1 applies only for the case when floating-point instructions operate on x registers instead of f registers. Whenever misa.F = 1, bit 1 of mstateen0 is read-only zero (and hence read-only zero in hstateen0 and sstateen0 too). For convenience, when the stateen CSRs are implemented and misa.F = 0, then if bit 1 of a controlling stateen0 CSR is zero, *all* floating-point instructions cause an illegal instruction trap (or virtual instruction trap, if relevant), as though they all access fcsr, regardless of whether they really do.

In addition to the bits listed above for user-accessible state, the following are also defined initially for mstateen0:

bit 57 - hcontext, scontext

bits 60:58 - Reserved for the RISC-V Advanced Interrupt Architecture

bit 61 - Reserved for possible henvcfg2/henvcfg2h, senvcfg2

bit 62 - henvcfg/henvcfgh, senvcfg

bit 63 - hstateen0/hstateen0h, sstateen0

The bits defined initially for hstateen0 are the same as those for mstateen0 except applying only to state that is accessible in VS-mode:

bit 57 - scontext

bits 60:58 - Reserved for the RISC-V Advanced Interrupt Architecture

bit 61 - Reserved for a possible `senvcfg2`

bit 62 - `senvcfg`

bit 63 - `sstateen0`

(Setting `hstateen0` bit 58 to zero prevents a virtual machine from accessing the hart's IMSIC the same as setting `hstatus`.VGEIN = 0.)

Chapter 2. Usage

After the writable bits of the machine-level `mstateen` CSRs are initialized to zeros on reset, machine-level software can set bits in these registers to enable less-privileged access to the controlled state. This may be either because machine-level software knows how to swap the state or, more likely, because machine-level software isn't swapping supervisor-level environments. (Recall that the main reason the `mstateen` CSRs must exist is so machine level can emulate the hypervisor extension. When machine level isn't emulating the hypervisor extension, it is likely there will be no need to keep any implemented `mstateen` bits zero.)

If machine level sets any writable `mstateen` bits to nonzero, it must initialize the matching `hstateen` CSRs, if they exist, by writing zeros to them. And if any `mstateen` bits that are set to one have matching bits in the `sstateen` CSRs, machine-level software must also initialize those `sstateen` CSRs by writing zeros to them. Ordinarily, machine-level software will want to set bit 63 of all `mstateen` CSRs, necessitating that it write zero to all `hstateen` CSRs.

Software should ensure that all writable bits of `sstateen` CSRs are initialized to zeros when an OS at supervisor level is first entered. The OS can then set bits in these registers to enable user-level access to the controlled state, presumably because it knows how to context-swap the state.

For the `sstateen` CSRs whose access by a guest OS is permitted by bit 63 of the corresponding `hstateen` CSRs, a hypervisor must include the `sstateen` CSRs in the context it swaps for a guest OS. When it starts a new guest OS, it must ensure the writable bits of those `sstateen` CSRs are initialized to zeros, and it must emulate accesses to any other `sstateen` CSRs.

If software at any privilege level does not support multiple contexts for less-privilege levels, then it may choose to maximize less-privileged access to all state by writing a value of all ones to the `stateen` CSRs at its level (the `mstateen` CSRs for machine level, the `sstateen` CSRs for an OS, and the `hstateen` CSRs for a hypervisor), without knowing all the state to which it is granting access. This is justified because there is no risk of a covert channel between execution contexts at the less-privileged level when only one context exists at that level. This situation is expected to be common for machine level, and it might also arise, for example, for a type-1 hypervisor that hosts only a single guest virtual machine.

Chapter 3. Possible expansion

If a need is anticipated, the set of `stateen` CSRs could in the future be doubled by adding these:

`0x38C mstateen4 0x39C mstateen4h`

`0x38D mstateen5 0x39D mstateen5h`

`0x38E mstateen6 0x39E mstateen6h`

`0x38F mstateen7 0x39F mstateen7h`

`0x18C sstateen4`

`0x18D sstateen5`

`0x18E sstateen6`

`0x18F sstateen7`

`0x68C hstateen4 0x69C hstateen4h`

`0x68D hstateen5 0x69D hstateen5h`

`0x68E hstateen6 0x69E hstateen6h`

`0x68F hstateen7 0x69F hstateen7h`

These additional CSRs are not a definite part of the original proposal because it is unclear whether they will ever be needed, and it is believed the rate of consumption of bits in the first group, registers numbered 0-3, will be slow enough that any looming shortage will be perceptible many years in advance. At the moment, it is not known even how many years it may take to exhaust just `mstateen0`, `sstateen0`, and `hstateen0`.

Index

Bibliography

RISC-V