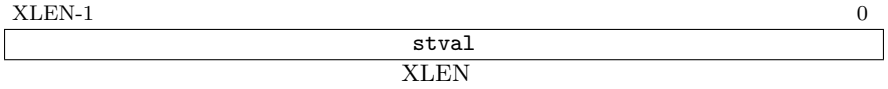


Figure 4.10: Supervisor Trap Value register.

tions, `stval` will point to the portion of the instruction that caused the fault while `sepc` will point to the beginning of the instruction.

The `stval` register can optionally also be used to return the faulting instruction bits on an illegal instruction exception (`sepc` points to the faulting instruction in memory).

If this feature is not provided, then `stval` is set to zero on an illegal instruction fault.

If the feature is provided, after an illegal instruction trap, `stval` will contain the entire faulting instruction provided the instruction is no longer than XLEN bits. If the instruction is less than XLEN bits long, the upper bits of `stval` are cleared to zero. If the instruction is more than XLEN bits long, `stval` will contain the first XLEN bits of the instruction.

`stval` is a **WARL** register that must be able to hold all valid physical and virtual addresses and the value 0. It need not be capable of holding all possible invalid addresses. Implementations may convert some invalid address patterns into other invalid addresses prior to writing them to `stval`. If the feature to return the faulting instruction bits is implemented, `stval` must also be able to hold all values less than 2^N, where N is the smaller of XLEN and the width of the longest supported instruction.

4.1.12 Supervisor Address Translation and Protection (`satp`) Register

The `satp` register is an XLEN-bit read/write register, formatted as shown in Figure 4.11 for RV32 and Figure 4.12, which controls supervisor-mode address translation and protection. This register holds the physical page number (PPN) of the root page table, i.e., its supervisor physical address divided by 4 KiB; an address space identifier (ASID), which facilitates address-translation fences on a per-address-space basis; and the MODE field, which selects the current address-translation scheme.

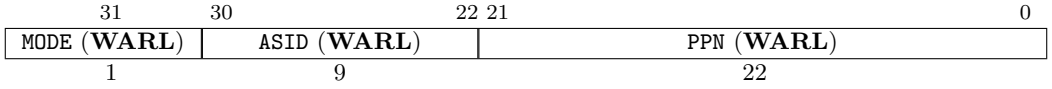


Figure 4.11: RV32 Supervisor address translation and protection register `satp`.

Storing a PPN in `satp`*, rather than a physical address, supports a physical address space larger than 4 GiB for RV32.*

We store the ASID and the page table base address in the same CSR to allow the pair to be changed atomically on a context switch. Swapping them non-atomically could pollute the old virtual address space with new translations, or vice-versa. This approach also slightly reduces the cost of a context switch.

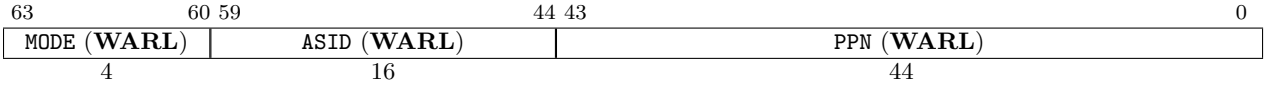


Figure 4.12: RV64 Supervisor address translation and protection register `satp`, for MODE values Sv39 and Sv48.

Table 4.3 shows the encodings of the MODE field for RV32 and RV64. When MODE=Bare, supervisor virtual addresses are equal to supervisor physical addresses, and there is no additional memory protection beyond the physical memory protection scheme described in Section 3.6. In this case, the remaining fields in `satp` have no effect.

For RV32, the only other valid setting for MODE is Sv32, a paged virtual-memory scheme described in Section 4.3.

For RV64, two paged virtual-memory schemes are defined: Sv39 and Sv48, described in Sections 4.4 and 4.5, respectively. Two additional schemes, Sv57 and Sv64, will be defined in a later version of this specification. The remaining MODE settings are reserved for future use and may define different interpretations of the other fields in `satp`.

Implementations are not required to support all MODE settings, and if `satp` is written with an unsupported MODE, the entire write has no effect; no fields in `satp` are modified.

RV32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32	Page-based 32-bit virtual addressing.

RV64		
Value	Name	Description
0	Bare	No translation or protection.
1–7	—	*Reserved*
8	Sv39	Page-based 39-bit virtual addressing.
9	Sv48	Page-based 48-bit virtual addressing.
10	*Sv57*	*Reserved for page-based 57-bit virtual addressing.*
11	*Sv64*	*Reserved for page-based 64-bit virtual addressing.*
12–15	—	*Reserved*

Table 4.3: Encoding of `satp` MODE field.

The number of supervisor physical address bits is implementation-defined; any unimplemented address bits are hardwired to zero in the `satp` register. The number of ASID bits is also implementation-defined and may be zero. The number of implemented ASID bits, termed *ASIDLEN*, may be determined by writing one to every bit position in the ASID field, then reading back the value in `satp` to see which bit positions in the ASID field hold a one. The least-significant bits of ASID are implemented first: that is, if ASIDLEN > 0, ASID[ASIDLEN-1:0] is writable. The maximal value of ASIDLEN, termed ASIDMAX, is 9 for Sv32 or 16 for Sv39 and Sv48

For many applications, the choice of page size has a substantial performance impact. A large

page size increases TLB reach and loosens the associativity constraints on virtually-indexed, physically-tagged caches. At the same time, large pages exacerbate internal fragmentation, wasting physical memory and possibly cache capacity.

After much deliberation, we have settled on a conventional page size of 4 KiB for both RV32 and RV64. We expect this decision to ease the porting of low-level runtime software and device drivers. The TLB reach problem is ameliorated by transparent superpage support in modern operating systems [2]. Additionally, multi-level TLB hierarchies are quite inexpensive relative to the multi-level cache hierarchies whose address space they map.

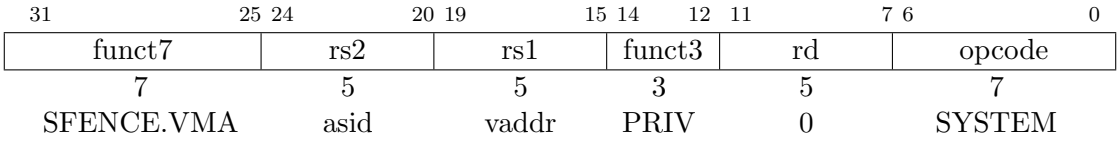
Note that writing `satp` does not imply any ordering constraints between page-table updates and subsequent address translations. If the new address space's page tables have been modified, it may be necessary to execute an SFENCE.VMA instruction (see Section 4.2.1) prior to writing `satp`.

Not imposing upon implementations to flush address-translation caches upon `satp` *writes reduces the cost of context switches, provided a sufficiently large ASID space.*

4.2 Supervisor Instructions

In addition to the SRET instruction defined in Section 3.2.2, one new supervisor-level instruction is provided.

4.2.1 Supervisor Memory-Management Fence Instruction



The supervisor memory-management fence instruction SFENCE.VMA is used to synchronize updates to in-memory memory-management data structures with current execution. Instruction execution causes implicit reads and writes to these data structures; however, these implicit references are ordinarily not ordered with respect to loads and stores in the instruction stream. Executing an SFENCE.VMA instruction guarantees that any stores in the instruction stream prior to the SFENCE.VMA are ordered before all implicit references subsequent to the SFENCE.VMA.

The SFENCE.VMA is used to flush any local hardware caches related to address translation. It is specified as a fence rather than a TLB flush to provide cleaner semantics with respect to which instructions are affected by the flush operation and to support a wider variety of dynamic caching structures and memory-management schemes. SFENCE.VMA is also used by higher privilege levels to synchronize page table writes and the address translation hardware.

Note the instruction has no effect on the translations of other RISC-V threads, which must be notified separately. One approach is to use 1) a local data fence to ensure local writes are visible globally, then 2) an interprocessor interrupt to the other thread, then 3) a local SFENCE.VMA in the interrupt handler of the remote thread, and finally 4) signal back to originating thread that operation is complete. This is, of course, the RISC-V analog to a TLB shootdown. Alternatively, implementations might provide direct hardware support for remote TLB invalidation. TLB shootdowns are handled by an SBI call to hide implementation details.

For the common case that the translation data structures have only been modified for a single address mapping (i.e., one page or superpage), *rs1* can specify a virtual address within that mapping to effect a translation fence for that mapping only. Furthermore, for the common case that the translation data structures have only been modified for a single address-space identifier, *rs2* can specify the address space. The behavior of SFENCE.VMA depends on *rs1* and *rs2* as follows:

- If *rs1*=`x0` and *rs2*=`x0`, the fence orders all reads and writes made to any level of the page tables, for all address spaces.
- If *rs1*=`x0` and *rs2*≠`x0`, the fence orders all reads and writes made to any level of the page tables, but only for the address space identified by integer register *rs2*. Accesses to *global* mappings (see Section 4.3.1) are not ordered.
- If *rs1*≠`x0` and *rs2*=`x0`, the fence orders only reads and writes made to the leaf page table entry corresponding to the virtual address in *rs1*, for all address spaces.
- If *rs1*≠`x0` and *rs2*≠`x0`, the fence orders only reads and writes made to the leaf page table entry corresponding to the virtual address in *rs1*, for the address space identified by integer register *rs2*. Accesses to global mappings are not ordered.

When *rs2*≠`x0`, bits XLEN-1:ASIDMAX of the value held in *rs2* are reserved for future use and should be zeroed by software and ignored by current implementations. Furthermore, if ASIDLEN < ASIDMAX, the implementation shall ignore bits ASIDMAX-1:ASIDLEN of the value held in *rs2*.

Simpler implementations can ignore the virtual address in rs1 *and the ASID value in* rs2 *and always perform a global fence.*

4.3 Sv32: Page-Based 32-bit Virtual-Memory Systems

When Sv32 is written to the MODE field in the `satp` register (see Section 4.1.12), the supervisor operates in a 32-bit paged virtual-memory system. Sv32 is supported on RV32 systems and is designed to include mechanisms sufficient for supporting modern Unix-based operating systems.

The initial RISC-V paged virtual-memory architectures have been designed as straightforward implementations to support existing operating systems. We have architected page table layouts to support a hardware page-table walker. Software TLB refills are a performance bottleneck on high-performance systems, and are especially troublesome with decoupled specialized coprocessors. An implementation can choose to implement software TLB refills using a machine-mode trap handler as an extension to M-mode.

4.3.1 Addressing and Memory Protection

Sv32 implementations support a 32-bit virtual address space, divided into 4 KiB pages. An Sv32 virtual address is partitioned into a virtual page number (VPN) and page offset, as shown in

Figure 4.13. When Sv32 virtual memory mode is selected in the MODE field of the `satp` register, supervisor virtual addresses are translated into supervisor physical addresses via a two-level page table. The 20-bit VPN is translated into a 22-bit physical page number (PPN), while the 12-bit page offset is untranslated. The resulting supervisor-level physical addresses are then checked using any physical memory protection structures (Sections 3.6), before being directly converted to machine-level physical addresses.

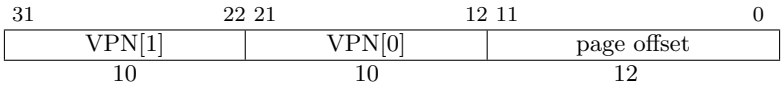


Figure 4.13: Sv32 virtual address.

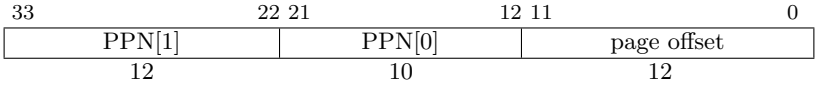


Figure 4.14: Sv32 physical address.

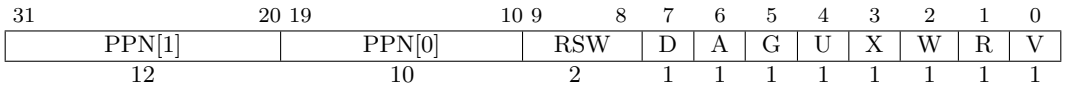


Figure 4.15: Sv32 page table entry.

Sv32 page tables consist of 2^{10} page-table entries (PTEs), each of four bytes. A page table is exactly the size of a page and must always be aligned to a page boundary. The physical page number of the root page table is stored in the `satp` register.

The PTE format for Sv32 is shown in Figures 4.15. The V bit indicates whether the PTE is valid; if it is 0, bits 31–1 of the PTE are don't-cares and may be used freely by software. The permission bits, R, W, and X, indicate whether the page is readable, writable, and executable, respectively. When all three are zero, the PTE is a pointer to the next level of the page table; otherwise, it is a leaf PTE. Writable pages must also be marked readable; the contrary combinations are reserved for future use. Table 4.4 summarizes the encoding of the permission bits.

X	W	R	Meaning
0	0	0	Pointer to next level of page table.
0	0	1	Read-only page.
0	1	0	*Reserved for future use.*
0	1	1	Read-write page.
1	0	0	Execute-only page.
1	0	1	Read-execute page.
1	1	0	*Reserved for future use.*
1	1	1	Read-write-execute page.

Table 4.4: Encoding of PTE R/W/X fields.

The U bit indicates whether the page is accessible to user mode. U-mode software may only access the page when U=1. If the SUM bit in the `sstatus` register is set, supervisor mode software may

also access pages with U=1. However, supervisor code normally operates with the SUM bit clear, in which case, supervisor code will fault on accesses to user-mode pages.

An alternative PTE format would support different permissions for supervisor and user. We omitted this feature because it would be largely redundant with the SUM mechanism (see Section 4.1.3) and would require more encoding space in the PTE.

The G bit designates a *global* mapping. Global mappings are those that exist in all address spaces. For non-leaf PTEs, the global setting implies that all mappings in the subsequent levels of the page table are global. Note that failing to mark a global mapping as global merely reduces performance, whereas marking a non-global mapping as global is an error.

Global mappings need not be stored redundantly in address-translation caches for multiple ASIDs. Additionally, they need not be flushed from local address-translation caches when an SFENCE.VMA instruction is executed with rs2≠`x0`.

The RSW field is reserved for use by supervisor software; the implementation shall ignore this field.

Each leaf PTE contains an accessed (A) and dirty (D) bit. The A bit indicates the virtual page has been read, written, or fetched from since the last time the A bit was cleared. The D bit indicates the virtual page has been written since the last time the D bit was cleared.

Two schemes to manage the A and D bits are permitted:

- When a virtual page is accessed and the A bit is clear, or is written and the D bit is clear, the implementation sets the corresponding bit in the PTE. The PTE update must be atomic with respect to other accesses to the PTE, and must atomically check that the PTE is valid and grants sufficient permissions. The PTE update must be exact (i.e., not speculative), and observed in program order by the local hart. The ordering on loads and stores provided by FENCE instructions and the acquire/release bits on atomic instructions also orders the PTE updates associated with those loads and stores as observed by remote harts.
- When a virtual page is accessed and the A bit is clear, or is written and the D bit is clear, a page-fault exception is raised.

Standard supervisor software should be written to assume either or both PTE update schemes may be in effect.

Mandating that the PTE updates to be exact, atomic, and in program order simplifies the specification, and makes the feature more useful for system software. Simple implementations may instead generate page-fault exceptions.

The A and D bits are never cleared by the implementation. If the supervisor software does not rely on accessed and/or dirty bits, e.g. if it does not swap memory pages to secondary storage or if the pages are being used to map I/O space, it should always set them to 1 in the PTE to improve performance.

Any level of PTE may be a leaf PTE, so in addition to 4 KiB pages, Sv32 supports 4 MiB *megapages*. A megapage must be virtually and physically aligned to a 4 MiB boundary; a page-fault exception is raised if the physical address is insufficiently aligned.

For non-leaf PTEs, the D, A, and U bits are reserved for future use and must be cleared by software for forward compatibility.

4.3.2 Virtual Address Translation Process

A virtual address *va* is translated into a physical address *pa* as follows:

1. Let a be `satp`.$ppn \times$ PAGESIZE, and let $i =$ LEVELS $- 1$. (For Sv32, PAGESIZE=2^{12} and LEVELS=2.)
2. Let *pte* be the value of the PTE at address $a+va.vpn[i]\times$PTESIZE. (For Sv32, PTESIZE=4.) If accessing *pte* violates a PMA or PMP check, raise an access exception.
3. If $pte.v = 0$, or if $pte.r = 0$ and $pte.w = 1$, stop and raise a page-fault exception.
4. Otherwise, the PTE is valid. If $pte.r = 1$ or $pte.x = 1$, go to step 5. Otherwise, this PTE is a pointer to the next level of the page table. Let $i = i - 1$. If $i < 0$, stop and raise a page-fault exception. Otherwise, let $a = pte.ppn \times$ PAGESIZE and go to step 2.
5. A leaf PTE has been found. Determine if the requested memory access is allowed by the *pte.r*, *pte.w*, *pte.x*, and *pte.u* bits, given the current privilege mode and the value of the SUM and MXR fields of the `mstatus` register. If not, stop and raise a page-fault exception.
6. If $i > 0$ and $pa.ppn[i-1:0] \neq 0$, this is a misaligned superpage; stop and raise a page-fault exception.
7. If $pte.a = 0$, or if the memory access is a store and $pte.d = 0$, either raise a page-fault exception or:
   - Set *pte.a* to 1 and, if the memory access is a store, also set *pte.d* to 1.
   - If this access violates a PMA or PMP check, raise an access exception.
   - This update and the loading of *pte* in step 2 must be atomic; in particular, no intervening store to the PTE may be perceived to have occurred in-between.
8. The translation is successful. The translated physical address is given as follows:
   - $pa.pgoff = va.pgoff$.
   - If $i > 0$, then this is a superpage translation and $pa.ppn[i-1:0] = va.vpn[i-1:0]$.
   - $pa.ppn[\text{LEVELS}-1:i] = pte.ppn[\text{LEVELS}-1:i]$.

4.4 Sv39: Page-Based 39-bit Virtual-Memory System

This section describes a simple paged virtual-memory system designed for RV64 systems, which supports 39-bit virtual address spaces. The design of Sv39 follows the overall scheme of Sv32, and this section details only the differences between the schemes.

We specified multiple virtual memory systems for RV64 to relieve the tension between providing a large address space and minimizing address-translation cost. For many systems, 512 GiB of virtual-address space is ample, and so Sv39 suffices. Sv48 increases the virtual address space to 256 TiB, but increases the physical memory capacity dedicated to page tables, the latency of page-table traversals, and the size of hardware structures that store virtual addresses.