The RISC-V Instruction Set Manual

Volume II: Privileged Architecture

Document Version 20211203

Editors: Andrew Waterman[1], Krste Asanović[1,2], John Hauser

[1]SiFive Inc.,

[2]CS Division, EECS Department, University of California, Berkeley

waterman@eecs.berkeley.edu, krste@berkeley.edu, jh.riscv@jhauser.us

2022 年 11 月 25 日

Contributors to all versions of the spec in alphabetical order (please contact editors to suggest corrections): Krste Asanović, Peter Ashenden, Rimas Avižienis, Jacob Bachmeyer, Allen J. Baum, Jonathan Behrens, Paolo Bonzini, Ruslan Bukin, Christopher Celio, Chuanhua Chang, David Chisnall, Anthony Coulter, Palmer Dabbelt, Monte Dalrymple, Paul Donahue, Greg Favor, Dennis Ferguson, Marc Gauthier, Andy Glew, Gary Guo, Mike Frysinger, John Hauser, David Horner, Olof Johansson, David Kruckemyer, Yunsup Lee, Daniel Lustig, Andrew Lutomirski, Prashanth Mundkur, Jonathan Neuschäfer, Rishiyur Nikhil, Stefan O'Rear, Albert Ou, John Ousterhout, David Patterson, Dmitri Pavlov, Kade Phillips, Josh Scheid, Colin Schmidt, Michael Taylor, Wesley Terpstra, Matt Thomas, Tommy Thorn, Ray VanDeWalker, Megan Wachs, Steve Wallach, Andrew Waterman, Claire Wolf, and Reinoud Zandijk.

Please cite as: "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203", Editors Andrew Waterman, Krste Asanović, and John Hauser, RISC-V International, December 2021.

Preface

本文档描述了 RISC-V 特权体系结构。此版本 (20211203) 包含以下版本的 RISC-V ISA 模块

模块	版本	状态
Machine ISA	**1.12**	**已批准**
Supervisor ISA	**1.12**	**已批准**
Svnapot Extension	**1.0**	**已批准**
Svpbmt Extension	**1.0**	**已批准**
Svinval Extension	**1.0**	**已批准**
Hypervisor ISA	**1.0**	**已批准**

自 1.11 版本以来，已经做出了以下更改，虽然不是严格的向后兼容，但在实践中预计不会引起软件可移植性问题:

- 当离开 M 模式时，更改 MRET 和 SRET 来清除 `mstatus`.MPRV。
- 保留额外的 `satp` 模式供将来使用。
- 声明 `scause` 异常代码字段必须至少实现 bits 4–0。
- 已根据 RVWMO 规定了宽松的 I/O 区域。之前的规范暗示除了 fences 和 acquire/release annotations 以外的 PPO 规则不适用。
- 在使用基于页面的虚拟内存时，限制 LR/SC 预留集的大小和形状?。
- 任何实现了基于页面的虚拟内存的 hart 上，PMP 更改需要一个 SFENCE.VMA，即使 VM 目前没有启用。
- 允许对页表项 A 位进行推测性更新。
- 说明如果地址转换算法非推测性地到达一个 PTE，其中设置了一个为未来标准使用而保留的比特位，则必须引发缺页异常。

此外，自版本以来还进行了以下兼容更改 1.11:

- 删除 N 扩展
- 定义了强制性 RV32-only 的 CSR `mstatush`，它包含了与 RV64 的 `mstatus` 高 32 位相同的大部分字段。

- 定义了强制性 CSR`mconfigptr`，如果它不为零，则包含配置数据结构的地址。
- 定义了可选的 `mseccfg` 和 `mseccfgh` CSRs，它们控制计算机的安全配置。
- 定义了 `menvcfg`、`henvcfg` 和 `senvcfg` CSRs(和 RV32-only`menvcfgh` 和 `henvcfgh` CSRs)，它们控制执行环境的各种特性。
- 自定义使用 SYSTEM 主要操作码的指定部分。
- 允许对特权级较低的中断进行无条件授权。
- 增加了可选的 big-endian 和 bi-endian 支持。
- 相对于加载/存储/AMO 页面错误和访问错误异常，优先考虑实现定义的加载/存储/AMO 地址错误异常。
- PMP 重置值现在是平台定义的。
- 另外还定义了 48 个可选的 PMP 寄存器。
- 稍微放宽了实现执行对 A 和 D 位更新的原子性要求。
- 阐明地址转换高速缓存的体系结构行为
- 增加了 Sv57 和 Sv57x4 地址转换模式。
- 允许软件断点异常将 0 或 PC 写入 x`tval`。
- 阐明了裸 S 模式不需要支持 SFENCE.VMA 指令。
- 指定非幂等区域的隐式读取的宽松约束。
- 增加了 Svnapot 标准扩展，以及 Sv39、Sv48 和 Sv57 PTEs 中的 N 位。
- 增加了 Svpbmt 标准扩展，以及 Sv39、Sv48 和 Sv57 PTEs 中的 PBMT 位。
- 增加了 Svinval 标准扩展和相关指令。

最后，对管理程序体系结构提议进行了广泛修改。

Preface to Version 1.11

这是 RISC-V 特权架构的 1.11 版本。文档中包含的 RISC-V ISA 模块版本如下:

模块	版本	状态
Machine ISA	**1.11**	**已批准**
Supervisor ISA	**1.11**	**已批准**
Hypervisor ISA	*0.3*	草案

与 1.10 版本相比的变化包括:

- 将 Machine 和 Supervisor 规格转变到**已批准**状态。
- 对描述和注释的改进。
- 添加了 hypervisor 扩展的提案草案。
- 指定哪些中断源被保留用于标准使用。
- 为自定义使用分配了一些同步异常原因。
- 指定同步异常的优先级顺序。
- 增加规范，如果扩展存在，xRET 指令可以 (但不是必需) 清除 LR 保留。
- 虚拟内存系统不再允许管理模式执行来自用户页面的指令，无论 SUM 如何设置。
- 阐明了 ASIDs 对一个 hart 是私有的，并添加了关于未来 global-ASID 扩展的可能性的评论。
- SFENCE.VMA 的语义已经被阐明。
- 使 `mstatus`.MPP 字段是 **WARL**，而不是 **WLRL**。
- 使未使用的 *x*`ip` 字段是 **WPRI**，而不是 **WIRI**。
- 使未使用的 `misa` 字段 **WARL**，而不是 **WIRI**。
- 使未使用的 `pmpaddr` 和 `pmpcfg` 字段是 **WARL**，而不是 **WIRI**。
- 要求系统中的所有 harts 都使用相同的 PTE 更新方案。
- 修正了一个编辑错误，它描述了在异常情况下编写 `mstatus`.*x* 的机制。
- 描述了模拟失调 AMOs 的方案。
- 指定在可变 IALIGN 的系统中 `misa` 和 *x*`epc` 寄存器的行为。
- 指定将自相矛盾的值写入 `misa` 寄存器的行为。
- 定义了 `mcountinhibit` CSR，它阻止性能计数器增加以减少能耗。
- 为大于四个字节的 PMP 区域指定语义。
- 指定跨 XLEN 修改 CSRs 的内容。
- 将 PLIC 章节移动到自己的文档中。

Preface to Version 1.10

这是 RISC-V 特权架构提案的 1.10 版本。从 1.9.1 版本开始的更改包括:

- 本文档的前一个版本是由原作者在知识共享署名 4.0 国际许可下发布的，本文档和未来的版本将在相同的许可下发布。
- 为了回收 CSR 空间，对影子内阁的 CSR 地址的显式约定被移除。仍然可以根据需要添加影子内阁的 CSRs。
- `mvendorid` 寄存器现在包含核心提供者的 JEDEC 代码，而不是由基金会提供的代码。这避免了冗余并将减轻基金会的工作。
- 支持中断的堆栈规程已经简化。
- `mstatus` CSR 中增加了一种可选的机制来改变管理员和用户模式使用的 base ISA，并且为了一致性，之前在 `misa` 中称为 base 的字段被重命名为 `MXL`。
- 阐明了期望使用 XS 来总结 `mstatus` 中的其他扩展状态的状态字段。
- `mtvec` 和 `stvec` CSRs 中添加了可选的向量中断支持。
- `mip` CSR 中的 SEIP 和 UEIP 位被重新定义，以支持外部中断的软件注入。
- `mbadaddr` 寄存器已被一个更通用的`-mtval` 寄存器所包含，该寄存器现在可以捕获非法指令错误上的错误指令位，以加快指令仿真。
- 机器模式的基础边界转换和保护方案已经从规范中删除，作为将虚拟内存配置移动到 `sptbr`(现在是 `satp`) 的一部分。base 和 bound 方案的一些诱因现在已经由 PMP 寄存器覆盖，但 `mstatus` 中仍然有空间，如果认为有用，可以在以后添加这些寄存器。
- 在只支持 M 模式，或者同时支持 M 模式和 U 模式但不支持 U 模式陷阱的系统中，`medeleg` 和 `mideleg` 寄存器现在不存在，而以前它们返回 0。
- 虚拟内存页面错误现在具有不同于物理内存访问错误的 `mcause` 值。现在可以将页面错误异常授权到 S 模式，而无需授权由 PMA 和 PMP 检查生成的异常。
- 提出了一种可选的物理内存保护 (PMP) 方案。
- 管理员虚拟内存配置已经从 `mstatus` 寄存器移动到 `sptbr` 寄存器。因此，`sptbr` 寄存器被重命名为 `satp`(Supervisor Address Translation and Protection)，以反映其扩大的作用。
- SFENCE.VM 指令已被删除，取而代之的是改进后的 SFENCE.VMA 指令。
- `mstatus` 位 MXR 已通过 `sstatus` 暴露到 s 模式。
- `sstatus` 中的 PUM 位的极性被颠倒，以缩短涉及 MXR 的代码序列。位已重命名为 SUM。
- 页表条目访问位和脏位的硬件管理是可选的; 更简单的实现可能会让设置它们的软件陷入陷阱。
- 计数器使能方案已更改，因此 S 模式可以控制计数器对 U 模式的可用性。
- H 模式已经被删除，因为我们专注于 s 模式下的递归虚拟化支持。编码空间已被保留，以后可以重新使用。

- 添加了一种通过捕获 S 模式虚拟内存管理操作来提高虚拟化性能的机制。
- Supervisor Binary Interface (SBI) 章节已经被删除，因此它可以作为一个单独的规范来维护。

Preface to Version 1.9.1

这是 RISC-V 特权架构提案的 1.9.1 版本。从 1.9 版本开始的更改包括：

- 对说明部分进行了大量的补充和改进。
- 将配置字符串提案更改为使用支持各种格式的搜索过程，包括设备树字符串和扁平设备树。
- 使 `misa` 可选的写入，以支持修改基础和支持的 ISA 扩展。`misa` 的 CSR 地址已更改。
- 增加调试模式和调试 CSRs 的说明。
- 增加硬件性能监控方案。简化了现有硬件计数器的处理，删除了计数器的特权版本和相应的增量寄存器。
- 修正了用户级中断时 SPIE 的描述。

目录

第一章　介绍

本文描述了 RISC-V 特权指令架构，其覆盖了在非特权 ISA 基础上的 RISC-V 系统的各个方面，包括特权指令以及运行操作系统和连接外设需要的额外功能。

如果读者只关心规范本身，本段落的注释可以忽略。

在不改变非特权指令集，甚至不改变 *ABI* 的情况下，本文描述的整个特权层级设计可以完全用不同的特权层级设计取代。这种特权规范设计可以与当前流行的操作系统相适应，体现了传统的基于层级的保护模型。另外，特权规范可以体现其它更多的基于域保护的模型。为简洁起见，本文只以一种可能的特权架构进行撰写。

1.1　RISC-V 特权软件栈术语

本节对 RISC-V 的特权软件栈涉及的相关专业术语进行描述。

图 1.1展示了 RISC-V 架构支持的可能软件栈。左图显示一个简单的系统，这个系统只支持在应用执行环境（AEE）里面运行单一的程序。应用程序使用特定的应用程序接口（ABI）运行。ABI 包括支持的用户级 ISA，和一系列与 AEE 交互的 ABI 调用。从应用程序的角度看，ABI 隐藏了 AEE 的细节，这对于 AEE 的实现提供了更多的弹性。同样的 ABI 可能被不同的宿主操作系统本地化实现，或者被运行在不同原生 ISA 的机器上的用户模式模拟环境所支持。

我们通常用黑框白字表示抽象接口，将其与实现接口的具体实例区别。

中间的图显示了一个支持多程序执行的传统操作系统。每个应用通过 ABI 与 OS 进行交互，OS 为其提供 AEE。RISC-V 操作系统与特权执行环境（SEE）通过一个特权二进制接口（SBI）进行交互。SBI 由用户级和监管级 ISA 以及一套 SBI 函数调用组成。通过一个所有 SEE 实现都支持的 SBI，一个操作系统映像可以运行在任意的 SEE 上。SEE 可能是底层硬件平台的一个简单的启动加载器和 BIOS 风格的 IO 系统，也可能是一个高端服务器里面的提供支持超级监管器的虚拟机，或者是一个体系架构模拟环境里的宿主操作系统上的一个瘦中继层。

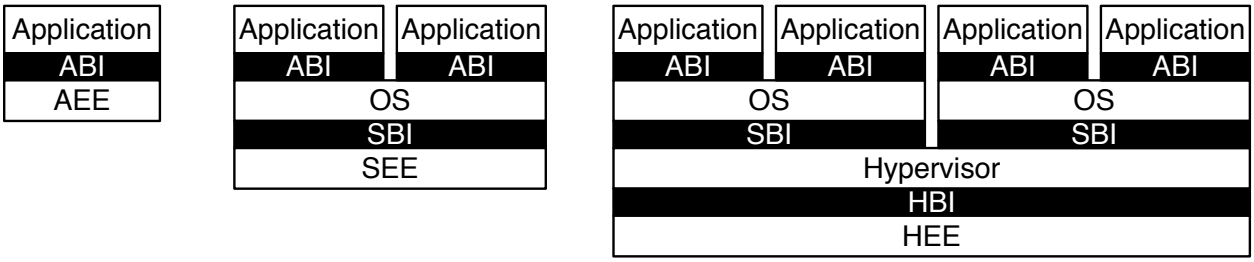


图 1.1: 支持各种特权执行形式的软件栈

大部分监管级 *ISA* 定义不会将 *SBI* 与可执行环境或/和硬件平台分开（那样将会使虚拟化复杂并产生新的硬件平台）

最右边的配置展示了一个虚拟机监视器的配置，在其中，一个单一的超级监管器支持多个多进程操作系统。每个操作系统通过 SBI 与提供 SEE 的超级监管器进行交互。超级管理进程以 HBI 接口的形式与 HEE 进行交互，从而将超级监管器与具体的硬件平台解耦。

ABI、*SBI*、和 *HBI* 规范依然在演进，我们优先支持第二种类型的超级进程，这种模式里面，*SBI* 由 *S* 模式（监管器模式）操作系统递归提供支持。

为了支持不同的执行环境（AEE，SEE，HEE），RISC-V ISA 的硬件实现需要在特权 ISA 的基础上增加额外的特征。

1.2 特权级别

在任意时刻，一个 RISC-V 硬件线程（*hart*）运行在某个特权层级（以一个或多个 CSR 进行编码）。表 1.1定义了三种 RISC-V 特权层级。

级别	编码	名称	缩写
0	00	用户（User）/应用（Application）	U
1	01	监管器（Supervisor）	S
2	10	保留的	
3	11	（机器）Machine	M

表 1.1: RISC-V 特权级别。

特权级别为软件栈的不同组件提供保护，试图在当前特权模式执行不允许的操作将导致抛出异常。这种异常一般会导致陷入下一层的执行环境。

在本文的描述中，我们试图将写代码的特权级别与其运行的特权模式相区分，虽然二者通常紧密联系在一起。例如，一个监管级操作系统可以三种特权模式运行在系统的监管器模式，它也可以在经典虚拟机下以用户模式运行在具有两个或多个特权模式的系统上运行。在这两种情况下，同一个超级层级操作系统二进制代码可以被编码为监管级 *SBI*，从而能够使用监管级特权指令和相应的 *CSR*。当在用户模式运行客户操作系统时，所有的监管级行为被捕获并被更高特权级运行的 *SEE* 模拟。

机器层级拥有最高的特权，它是 RISC-V 硬件平台的唯一强制性特权层级。在机器模式下运行的代码被认为是天然可信的，它可以直接访问计算机底层硬件。机器模式用于管理 RISC-V 的安全可执行环境。用户模式和监管器模式分别适用于传统的应用程序和操作系统。

每一个特权层级拥有一套核心特权 ISA 及其可选的扩展和变体。例如，机器模式针对内存保护支持可选的标准扩展。此外，监管器模式可以扩展以支持 type-2 类超级监管器执行，如第 八中所述。

具体实现可以提供编号为从 1 到 3 的特权模式，以降低底层实现之间的解耦成本。如表 1.2所示。

级别编号	支持的模式	预期的用途
1	M	简单嵌入式系统
2	M, U	安全嵌入式系统
3	M, S, U	运行类 Unix 操作系统的系统

表 1.2: 支持的特权模式组合

所有硬件实现必须提供机器模式，因为它是唯一的可以不受限制地访问整个机器的模式。最简单的 RISC-V 实现可以只提供机器模式，但是这样就会缺少对导致错误访问或恶意代码的保护。

可选 *PMP* 的锁特性能为仅仅提供机器模式的实现提供有限的保护。

为了对除应用程序代码的其余部分提供保护，许多 RISC-V 系统的实现至少支持用户模式（U-模式）。同时，为了保证监管级操作系统与 SEE 的隔离，监管器模式（S-模式）也可以添加到 RISC-V 的实现中。

在 hart 里，应用代码通常运行在用户模式下，当捕获到某个陷阱时（例如：一个监管器调用或一个时钟中断），hart 会切换到特权级别更高的陷阱处理程序。这个时候，hart 会运行陷阱处理程序，在陷阱处理程序执行完之后，它将返回到用户模式中产生陷阱的那条指令，或下一条指令继续执行。增加特权层级的陷阱称之为垂直 陷阱，保持原有特权层级的陷阱称之为水平陷阱。RISC-V 特权架构为不同的特权层级提供陷阱灵活的路由。

水平陷阱也可以通过在低特权即模式下返回一个水平陷阱的控制去实现垂直陷阱。

1.3 调试模式

RISC-V 的某种实现还可能包括一个调试模式，去支持片外的调试模式调试和/或制造测试。调试模式（D 模式）可以认为是一种附加的特权模式，它甚至具有比机器模式更多的访问权限。单独的调试规范提案描述了调试模式下的 RISC-V hart 的操作行为。调试模式只保留一些能在 D 模式下访问的 CSR 地址，也保留了平台上物理地址空间上的一些部分。

第二章　控制和状态寄存器（CSR）

在 RISC-V 指令集中，系统主操作码（STSTEM）编码所有特权指令。这些特权指令可以分为两类，一类为 Zicsr 扩展中定义的可以自动读-修改-写的控制和状态寄存器；另一类为之外的其它特权指令。特权架构需要 Zicsr 扩展支持，其它特权指令依赖于特权架构特征集。

除了在本手册第一卷中描述的非特权指令集外，具体实现可能包含一些额外的 CSR，这些控制和状态寄存器可通过本手册第一卷所描述的控制状态寄存器指令所规定的特权级别进行访问。本章主要描述控制寄存器的地址空间，后续章节根据特权级别描述每个状态控制寄存器的功能，以及与某个具体的特权级别相关的一些特权指令。请注意，虽然 CSR 与指令与一个特权级别相关联，但是它们也可以被更高的特权级别访问

标准的控制寄存器在读写操作时需要注意，读操作是没有副作用的，而写操作是有副作用的。

2.1　CSR 地址映射约定

标准 RISC-V 指令集预置了 12 比特的编码空间 (csr[11:0])，可以编码 4096 个控制寄存器。通常，高 4 位 (csr[11:8]) 用于对表格2.1中所示的特权级别访问控制寄存器的读写操作进行编码。其中的高两位 (csr[11:10]) 指示寄存器是读/写 (`00`, `01`, or `10` 或只读 (`11`))。低 2 位 (csr[9:8]) 对访问控制寄存器的最低特权级别进行编码。

CSR 地址约定使用 CSR 的高位比特编码默认的访问权限。这种方式简化了硬件的错误检测并提供了更大的 CSR 空间，但是限制了 CSR 地址空间的映射方式。

在具体实现中，可能允许特权更高的级别捕获特权较低的级别本来允许的 CSR 访问，以允许拦截这些访问。这些改变对低特权级别的软件应该透明。

试图访问一个不存在 CSR 会引发一个不合法的指令异常。以不恰当的特权级别访问 CSR 或写一个只读寄存器也会导致非法指令异常。一个读/写寄存器中的某些位可能是只读的，在这种情况下，针对只读位的写操作应该被忽略。

表 2.1约定了 CSR 地址空间的标准用法和自定义用法。分配给自定义使用的 CSR 空间在将来的标准中将不再重新定义。

机器模式标准读-写 CSR `0x7A0`–`0x7BF` 是为系统调试预留的。在这些 CSR 中，`0x7A0`–`0x7AF` 在机器模式中访问，`0x7B0`–`0x7BF` 只在调试模式中可见。具体的实现中，机器模式访问后者的寄存器时，应该报指令异常错误。

在一个虚拟机环境中，需要尽可能多的指令在本地运行，然而任何特权访问都会陷入到虚拟监视器中 *[1]*。如果在低特权级别只读的 *CSR* 在高特权级别是可读-写的，可以将低特权级别的 *CSR* 隐藏在单独的 *CSR* 空间中。这样可以避免捕获低级别的访问，但是依然会捕获一些不合法的访问。当前，计数器是唯一的隐藏 *CSR*。

2.2 CSR 列表

表 2.2–2.6列出了当前已被分配的 CSR 地址。计时器，计数器和浮点 CSR 都是非特权 CSR。其它寄存器都是特权编码，后续章节会详细描述。请注意，并非所有的寄存器在实现时都是必须的。

CSR 地址			16 进制	用法和访问权限
[11:10]	[9:8]	[7:4]		
非特权和用户级别的 CSR				
00	00	XXXX	0x000-0x0FF	标准读/写
01	00	XXXX	0x400-0x4FF	标准读/写
10	00	XXXX	0x800-0x8FF	自定义读/写
11	00	0XXX	0xC00-0xC7F	标准只读
11	00	10XX	0xC80-0xCBF	自定义只读
11	00	11XX	0xCC0-0xCFF	自定义只读
特权级别的 CSR				
00	01	XXXX	0x100-0x1FF	标准读/写
01	01	0XXX	0x500-0x57F	标准读/写
01	01	10XX	0x580-0x5BF	标准读/写
01	01	11XX	0x5C0-0x5FF	自定义读/写
10	01	0XXX	0x900-0x97F	标准读/写
10	01	10XX	0x980-0x9BF	标准读/写
10	01	11XX	0x9C0-0x9FF	自定义读/写
11	01	0XXX	0xD00-0xD7F	自定义只读
11	01	10XX	0xD80-0xDBF	自定义只读
11	01	11XX	0xDC0-0xDFF	自定义只读
超级监管模式和虚拟监管模式（VS）的 CSR				
00	10	XXXX	0x200-0x2FF	标准读/写
01	10	0XXX	0x600-0x67F	标准读/写
01	10	10XX	0x680-0x6BF	标准读/写
01	10	11XX	0x6C0-0x6FF	自定义读/写
10	10	0XXX	0xA00-0xA7F	标准读/写
10	10	10XX	0xA80-0xABF	标准读/写
10	10	11XX	0xAC0-0xAFF	自定义读/写
11	10	0XXX	0xE00-0xE7F	自定义只读
11	10	10XX	0xE80-0xEBF	自定义只读
11	10	11XX	0xEC0-0xEFF	自定义只读
机器级的 CSR				
00	11	XXXX	0x300-0x3FF	标准读/写
01	11	0XXX	0x700-0x77F	标准读/写
01	11	100X	0x780-0x79F	标准读/写
01	11	1010	0x7A0-0x7AF	标准读/写调试 CSR
01	11	1011	0x7B0-0x7BF	调试模式专用 CSR
01	11	11XX	0x7C0-0x7FF	自定义读/写
10	11	0XXX	0xB00-0xB7F	标准读/写
10	11	10XX	0xB80-0xBBF	标准读/写
10	11	11XX	0xBC0-0xBFF	自定义读/写

编号	特权	名称	描述
非特权级别的浮点 CSR			
0x001	URW	fflags	浮点应计异常。(Floating-Point Accrued Exceptions)
0x002	URW	frm	浮点动态舍入模式。
0x003	URW	fcsr	浮点控制和状态寄存器（frm + fflags)。
非特权计数器/计时器			
0xC00	URO	cycle	RDCYCLE 指令的时钟周期计数器。
0xC01	URO	time	RDTIME 指令的计时器。
0xC02	URO	instret	RDINSTRET 指令的指令退休计数器。
0xC03	URO	hpmcounter3	性能监视计数器。
0xC04	URO	hpmcounter4	性能监视计数器。
		⋮	
0xC1F	URO	hpmcounter31	性能监视计数器。
0xC80	URO	cycleh	cycle 的高 32 位，仅用于 RV32。
0xC81	URO	timeh	time 的高 32 位，仅用于 RV32。
0xC82	URO	instreth	instret 的高 32 位，仅用于 RV32。
0xC83	URO	hpmcounter3h	hpmcounter3 的高 32 位，仅用于 RV32。
0xC84	URO	hpmcounter4h	hpmcounter4 的高 32 位，仅用于 RV32。
		⋮	
0xC9F	URO	hpmcounter31h	hpmcounter31 的高 32 位，仅用于 RV32。

表 2.2: 当前已分配的 RISC-V 非特权 CSR 地址。

编号	特权	名称	描述
监管级陷阱设置			
0x100	SRW	sstatus	监管级状态寄存器。
0x104	SRW	sie	监管级中断使能寄存器。
0x105	SRW	stvec	监管级陷阱处理程序基址。
0x106	SRW	scounteren	监管级计数器使能。
监管器配置			
0x10A	SRW	senvcfg	监管级环境配置寄存器。
监管级陷阱处理			
0x140	SRW	sscratch	监管级陷阱处理程序的暂存器。
0x141	SRW	sepc	监管级异常程序计数器。
0x142	SRW	scause	监管级陷阱原因。
0x143	SRW	stval	监管级发生错误的地址或指令。
0x144	SRW	sip	监管级中断挂起。
监管级保护和翻译			
0x180	SRW	satp	监管级地址转换和保护。
调试/跟踪寄存器			
0x5A8	SRW	scontext	监管模式上下文寄存器。

表 2.3: 当前已分配的 RISC-V 监管级 CSR 地址。

编号	特权	名称	描述
超级监管器陷阱设置			
0x600	HRW	hstatus	超级监管器状态寄存器。
0x602	HRW	hedeleg	超级监管器异常代理寄存器。
0x603	HRW	hideleg	超级监管器中断代理寄存器。
0x604	HRW	hie	超级监管器中断使能寄存器。
0x606	HRW	hcounteren	超级监管器计数器使能。
0x607	HRW	hgeie	超级监管器客户外部中断使能寄存器。
超级监管器陷阱处理程序			
0x643	HRW	htval	超级监管器错误物理地址。
0x644	HRW	hip	超级监管器中断挂起。
0x645	HRW	hvip	超级监管器虚拟中断挂起。
0x64A	HRW	htinst	超级监管器发生陷入指令（transformed）。
0xE12	HRO	hgeip	超级监管器客户外部中断挂起。
超级监管器配置			
0x60A	HRW	henvcfg	超级监管器环境配置寄存器。
0x61A	HRW	henvcfgh	额外的超级监管器环境、配置寄存器，只用于 RV32。
超级监管器保护和转换			
0x680	HRW	hgatp	超级监管器客户地址转换和保护。
调试/跟踪寄存器			
0x6A8	HRW	hcontext	超级监管器模式上下文寄存器
超级监管器计数器/时钟虚拟化寄存器			
0x605	HRW	htimedelta	Delta for VS/VU-mode timer.
0x615	HRW	htimedeltah	htimedelta 的高 32 位，仅用于 HSXLEN=32
虚拟监管器寄存器			
0x200	HRW	vsstatus	虚拟监管器状态寄存器。
0x204	HRW	vsie	虚拟监管器中断使能寄存器。
0x205	HRW	vstvec	虚拟监管器陷阱处理程序基地址。
0x240	HRW	vsscratch	虚拟监管器暂存寄存器。
0x241	HRW	vsepc	虚拟监管器异常程序指针。
0x242	HRW	vscause	虚拟监管器陷阱原因。
0x243	HRW	vstval	虚拟监管器错误地址或指令。
0x244	HRW	vsip	虚拟监管器中断挂起。
0x280	HRW	vsatp	虚拟监管器地址转换和保护。

表 2.4: 当前已分配的 RISC-V 超级监管器和 VS CSR 地址。

编号	特权	名称	描述
机器信息寄存器			
0xF11	MRO	mvendorid	供应商 ID。
0xF12	MRO	marchid	架构 ID。
0xF13	MRO	mimpid	实现 ID。
0xF14	MRO	mhartid	硬件线程 ID。
0xF15	MRO	mconfigptr	配置数据结构指针。
机器陷阱配置			
0x300	MRW	mstatus	机器状态寄存器。
0x301	MRW	misa	ISA 和拓展。
0x302	MRW	medeleg	机器异常代理寄存器。
0x303	MRW	mideleg	机器中断代理寄存器。
0x304	MRW	mie	机器中断使能寄存器。
0x305	MRW	mtvec	机器陷阱处理程序基地址。
0x306	MRW	mcounteren	机器计数器使能。
0x310	MRW	mstatush	额外的机器状态寄存器，仅用于 RV32。
机器陷阱处理程序			
0x340	MRW	mscratch	机器陷阱处理程序的暂存器。
0x341	MRW	mepc	机器异常程序计数器。
0x342	MRW	mcause	机器陷阱原因。
0x343	MRW	mtval	机器错误地址或指令。
0x344	MRW	mip	机器中断挂起。
0x34A	MRW	mtinst	机器陷阱指令 (transformed)。
0x34B	MRW	mtval2	机器错误客户物理地址。
机器配置			
0x30A	MRW	menvcfg	机器环境配置寄存器。
0x31A	MRW	menvcfgh	额外机器环境、配置寄存器，仅用于 RV32。
0x747	MRW	mseccfg	机器安全配置寄存器。
0x757	MRW	mseccfgh	额外的机器安全配置寄存器，仅用于 RV32。
机器内存保护			
0x3A0	MRW	pmpcfg0	物理内存保护配置。
0x3A1	MRW	pmpcfg1	物理内存保护配置，仅用于 RV32。
0x3A2	MRW	pmpcfg2	物理内存保护配置。
0x3A3	MRW	pmpcfg3	物理内存保护配置，仅用于 RV32。
		⋮	
0x3AE	MRW	pmpcfg14	物理内存保护配置。
0x3AF	MRW	pmpcfg15	物理内存保护配种，仅用于 RV32。
0x3B0	MRW	pmpaddr0	物理内存保护地址寄存器。
0x3B1	MRW	pmpaddr1	物理内存保护地址寄存器。
		⋮	
0x3EF	MRW	pmpaddr63	物理内存保护地址寄存器。

编号	特权	名称	描述
机器计数器/计时器			
0xB00	MRW	mcycle	机器时钟周期计数器。
0xB02	MRW	minstret	机器指令退休计数器。
0xB03	MRW	mhpmcounter3	机器性能监视计数器。
0xB04	MRW	mhpmcounter4	机器性能监视计数器
		⋮	
0xB1F	MRW	mhpmcounter31	机器性能监视计数器。
0xB80	MRW	mcycleh	mcycle 的高 32 位，仅用于 RV32。
0xB82	MRW	minstreth	minstret 的高 32 位，仅用于 RV32。
0xB83	MRW	mhpmcounter3h	mhpmcounter3 的高 32 位，仅用于 RV32。
0xB84	MRW	mhpmcounter4h	mhpmcounter4 的高 32 位，仅用于 RV32。
		⋮	
0xB9F	MRW	mhpmcounter31h	mhpmcounter31 的高 32 位，仅用于 RV32。
机器计数器配置			
0x320	MRW	mcountinhibit	机器屏蔽计数器寄存器。
0x323	MRW	mhpmevent3	机器性能监视活动选择器。
0x324	MRW	mhpmevent4	机器性能监视活动选择器。
		⋮	
0x33F	MRW	mhpmevent31	机器性能监视活动选择器。
调试/跟踪寄存器（与调试模式共享）			
0x7A0	MRW	tselect	调试/跟踪触发寄存器选择。
0x7A1	MRW	tdata1	第一个调试/跟踪触发数据寄存器。
0x7A2	MRW	tdata2	第二个调试/跟踪触发数据寄存器。
0x7A3	MRW	tdata3	第三个调试/跟踪触发数据寄存器。
0x7A8	MRW	mcontext	机器模式上下文寄存器。
调试模式寄存器			
0x7B0	DRW	dcsr	调试控制和状态寄存器。
0x7B1	DRW	dpc	调试程序计数器。
0x7B2	DRW	dscratch0	调试暂存寄存器 0。
0x7B3	DRW	dscratch1	调试暂存寄存器 1。

表 2.6: 当前已分配的 RISC-V 机器级 CSR 地址。

2.3 CSR 字段规范

下面的定义和简称说明 CSR 中各字段的行为。

保留的写保护值，读忽略值（WPRI）

某些读/写整个字段都保留为将来使用。软件应该忽略从这些字段中读出的值，同时，在写寄存器的其他字段时，应该保留这些字段的值。为了前向兼容，不提供字段的具体实现必须将这些字段置为只读的零。在寄存器说明中，这些字段标记为 **WPRI** 。

为了简化软件模型，*CSR* 中的预留字段在将来后向兼容的定义中，更新 *CSR* 其它字段时必须处理非原子的读/修改/写顺序。

或者，原始的 *CSR* 定义必须说明哪些子字段只能以原子步的形式更新，这通常需要两指令清除位/设置位，从而导致中间值不合法。

只写/读合法值（WLRL）

某些读/写 CSR 字段描述部分位编码的行为，其它编码保留。针对这些字段，软件应该只写入合法的值，并且不能假设一个读操作会得到一个合法的值，除非最后的写操作是一个合法的值，或者寄存器在被置为一个合法值之后没有其它的写操作。这些字段在寄存器描述中标记为 **WLRL**

硬件实现只需实现足够的状态位区分支持的值，但是读操作返回时，必须返回完整的任何支持值的指定位编码。

如果指令试图向 **WLRL** 字段写一个不支持的值，实现可以抛出一个非法指令异常。在读 **WLRL** 字段时，如果最后的写操作是一个不合法的值，实现可以返回任意的位模式，但是返回值应该明确依赖于不合法的写入值和写入之前的值。

写任意值，读合法值（WARL）

某些读/写操作 CSR 字段只定义部分编码，在能够保证读操作返回合法值的前提下，允许写入任何值。假设写入 CSR 没有副作用，可以通过写入预期的值然后查看是否保留来确认支持值的范围。这些字段在寄存器描述中标记为 **WARL** 。

针对 **WARL** 字段，写入不支持的值时，系统不会抛出异常。在读一个 **WARL** 字段时，当最后的写入是一个非法的值时，系统可以返回一个合法的值，但是返回的值应该由非法写入的值和 hart 的架构状态确定。

2.4 CSR 字段的模块化设计

如果对一个 CSR 的写入更改了第二个 CSR 的字段允许的合法值的集合，则除非另有说明，第二个 CSR 的字段会立即从其新的合法值中获取一个 未指定的 。即使写入前字段的值在写入后仍然合法，也是如此；由于写入控制 CSR，该字段的值可能会更改。

作为这种规则的特例，写入 *CSR* 的值可能控制另一个 *CSR* 的字段为可写或只读。控制 *CSR* 的一个写操作导致另一个 *CSR* 的字段从之前的只读状态变为当前的可写，这个字段将变为未指定的 的合法值，除非指定特定的值。

某些 *CSR* 可写字段被定义为其它 *CSR* 字段的别名。若 x 是一个这样的 *CSR* 字段，y 是 x 的别名，如果控制 *CSR* 上的一个写操作导致 x 从之前的只读状态变为当前的可写状态，则 x 的最新值不是未指定的 ，而是立即映射到已有别名 y 的值。

出于这种原因对 CSR 值的改变，不是受此影响 CSR 上的一个写操作，因此不会针对这个 CSR 产生特定的副作用。

2.5 CSR 的隐式读

具体实现有时候采用隐式读 CSR，例如，所有的 S-mode 指令隐式获取读 `satp`CSR 的结果。如果没有特殊说明，特权模式下的 CSR 访问指令，在 CSR 上的隐式读操作返回的值与在其上显示读操作返回的值是一致的，

2.6 CSR 宽度模块化设计

If the width of a CSR is changed (for example, by changing MXLEN or UXLEN, as described in Section 3.1.6.2), the values of the *writable* fields and bits of the new-width CSR are, unless specified otherwise, determined from the previous-width CSR as though by this algorithm:

如果 CSR 的宽度发生了改变（比如，通过修改 MXLEN 或 UXLEN，在第 3.1.6.2节描述），在没有特殊说明的情况下，CSR 中 *writable* 字段的值和具有新宽度的 CSR 依然由之前的宽度确定，就想下面这个算法描述的一样：

1. 宽度改变前的 CSR 值拷贝到具有同样宽度的临时寄存器中
2. 对于宽度改变前的 CSR 中的只读位，临时寄存器中的相应位置为 0

3. 临时寄存器的宽度根据新的宽度进行调整。如果新的宽度 W 比之前的宽度要小，则临时寄存器中的低 W 位被保留，而高位则被丢弃。如果新的宽度比之前的宽度大，则临时寄存器需要按照补零方式扩展至指定宽度。

4. 对于新宽度 CSR 中的每个可写字段，都采用临时寄存器中同样比特位作为其值。

改变 CSR 的宽度不是 CSR 的读或写操作，不会产生副作用。

第三章　机器级 ISA, 1.12 版本

本章介绍了机器模式 (M 模式) 下可用的机器级操作，这是 RISC-V 系统中权限最高的模式。M 模式用于对硬件平台的底层（低级?）访问，是重启时进入的第一个模式。M 模式还可以用于完成那些难以直接用硬件实现或成本昂贵的功能。RISC-V 的机器级 ISA 包含一个共同核心，其根据所支持的其它特权等级和硬件实现中的其它细节来进行扩展。

3.1　机器级 CSRs

除了本节所介绍的机器级 CSRs，M 模式的代码能够在低特权级别访问所有的 CSRs。

3.1.1　机器 ISA 寄存器 misa

misa CSR 是一个 **WARL** 读写寄存器，用于报告 hart 所支持的 ISA。该寄存器在任何实现当中都必须是可读的，但是可以返回一个 0 值来表示没有实现 misa 寄存器，这时需要通过独立的非标准机制来确定 CPU 的能力。

图 3.1: Machine ISA register (misa).

MXL 字段表示（编码?）本机基础整数型 ISA 宽度，如图 3.1所示。在支持多个基础 ISA 的实现中，MXL 字段能够是可写的。M 模式下的有效 XLEN 是 MXLEN，由 MXL 的设置给出，如果 misa 为零，则其为一个固定值。在复位时，总是将 MXL 字段设为所支持的最宽的 ISA 变体。

misa CSR 的位宽是 MXLEN bits。如果从 misa 中读取到的是非零值，那这个值中的 MXL 字段总是表示当前的 MXLEN。如果对 misa 的写入造成了 MXLEN 的改变，MXL 的位置将变化到在新位宽下 misa 的最高有效 2bits。

MXL	XLEN
1	32
2	64
3	128

表 3.1: `misa` 中的 MXL 字段编码

基本宽度可以通过 `misa` 返回值的符号的分支来快速确定，也可以通过符号左移一位和第二个分支来确定。可以在不知道机器的寄存器位宽（*XLEN*）的情况下用汇编语言来编写这些检查。基础宽度由 $XLEN = 2^{MXL+4}$ 给出。

当 `misa` 为零时，基础宽度可以通过将立即数 *4* 放入一个寄存器，然后将其左移 *32bits* 来得到。如果移位一次之后是零，则机器是 *RV32* 的；如果是两次移位后得到零，则机器是 *RV64* 的；否则就是 *RV128*。

Extensions 字段编码了目前存有的标准扩展，其每一位都对应了字母表中的一个字母（bit 0 对扩展 “A” 编码，bit 1 对扩展 “B” 编码，以此类推，直至 bit 25 编码 “Z”）。如果基础 ISA 是 RV32I、RV64I 或 RV128I，则置位 “I” bit，如果基础 ISA 是 RV32E，则置位 “E”bit。Extensions 字段是一个能包含可写位的 WARL 字段，需要（where?）实现允许修改所支持的 ISA。复位时，Extensions 字段应包含所支持的最大扩展集，如果 E 和 I 都可以，那么优先选择 I。

当通过清除 `misa` 中相应位（bit）来禁止一个标准扩展时，经扩展定义或修改的指令和 CSRs 会恢复到它们定义或保留的行为，就像扩展没有实现一样。

RV128 基础 ISA 的设计还没有完成，尽管本规格中剩余的大部分都期望适用于 RV128，但本版本的文档仅关注 RV32 和 RV64。

如果支持用户模式（user mode），则将“U”bit 置位；如果支持特权模式（supervisor mode），则将“S ”bit 置位。

如果存在任何非标准扩展（non-standard extensions），则将“X”bit 置位。

`misa` CSR 向机器模式的代码展示了 CPU 的基本功能目录。在机器模式下，可以通过探测其它机器寄存器，并且作为启动程序（boot process）的一部分，检查系统中其他的 ROM 存储器，来获得更广泛的信息。

我们要求较低特权级使用环境调用来确定每个特权级别的可用功能，而不是直接读取 CPU 寄存器。这使得虚拟层能够改变在任何级别上可观察到的 ISA，并支持更加丰富的命令接口，且不会带来额外的硬件设计的负担。

Bit	Character	Description
0	A	Atomic extension
1	B	*Tentatively reserved for Bit-Manipulation extension*
2	C	Compressed extension
3	D	Double-precision floating-point extension
4	E	RV32E base ISA
5	F	Single-precision floating-point extension
6	G	*Reserved*
7	H	Hypervisor extension
8	I	RV32I/64I/128I base ISA
9	J	*Tentatively reserved for Dynamically Translated Languages extension*
10	K	*Reserved*
11	L	*Reserved*
12	M	Integer Multiply/Divide extension
13	N	*Tentatively reserved for User-Level Interrupts extension*
14	O	*Reserved*
15	P	*Tentatively reserved for Packed-SIMD extension*
16	Q	Quad-precision floating-point extension
17	R	*Reserved*
18	S	Supervisor mode implemented
19	T	*Reserved*
20	U	User mode implemented
21	V	*Tentatively reserved for Vector extension*
22	W	*Reserved*
23	X	Non-standard extensions present
24	Y	*Reserved*
25	Z	*Reserved*

表 3.2: Encoding of Extensions field in `misa`. All bits that are reserved for future use must return zero when read.

bit E 是只读的。除非 `misa` 是只读全零，否则 “E” bit 总是 “I” bit 的补码。对于同时支持 RV32E 和 RV32I 的实现，可通过清除“I”bit 来选择 RV32E。

如果 ISA 功能 x 依赖于 ISA 功能 y，那么尝试在启用功能 x 的同时禁用功能 y，会导致两个功能都被禁用。例如，设置 “F”=0 和 “D”=1 会导致 “F” 和 “D” 都被清除。

一个实现可能会对两个或多个 `misa` 字段的集合设置施加额外的限制，在这种情况下，它们共同作为一个单独的 **WARL** 字段发挥作用。尝试写入不受支持的组合会导致这些位被设置成一些受支持的组合。

写入 `misa` 可能会增加 IALIGN，例如，通过禁用” C ” 扩展。如果要写入 `misa` 的指令增加 IALIGN，而后续指令的地址不是 IALIGN-bit 对齐的，则抑制对 `misa` 的写入，保持 `misa` 不变。

当软件启用以前禁用的扩展时，所有与该扩展唯一关联的状态都是未指定的，除非该扩展另有指定。

3.1.2 机器厂商 ID 寄存器 `mvendorid`

`mvendorid` CSR 是一个 32 位只读寄存器，提供核心提供者的 JEDEC 制造商 ID。该寄存器在任何实现中都必须是可读的，但是可以返回 0 值，表示该字段没有实现，或者这是一个非商业实现。

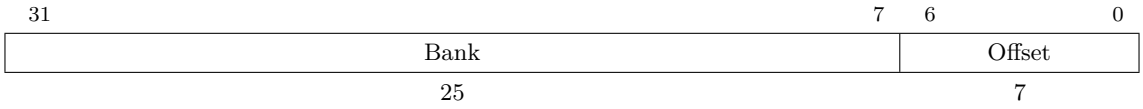


图 3.2: Vendor ID register (`mvendorid`).

JEDEC 制造商 ID 通常被编码为一个单字节延续码序列 `0x7f`，由一个不等于 `0x7f` 的单字节 ID 终止，每个字节的最高有效位有一个奇偶校验位。`mvendorid` 编码 Bank 字段中单字节延续码的数量，并编码 Offset 字段中的最后一个字节，丢弃校验位。例如，JEDEC 制造商 ID `0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x7f 0x8a`(十二个延续码，后面跟着 `0x8a`) 将被编码为 `0x60a`。

按照 JEDEC 的说法，银行号比延续代码的数量大 1; 因此，`mvendorid` *Bank 字段编码一个比 JEDEC 银行号小 1 的值。*

以前，供应商 ID 是由 RISC-V International 分配的号码，但这重复了 JEDEC 维持制造商 ID 标准的工作。在撰写本文时，向 JEDEC 注册一个制造商 ID 的一次性成本为 $500。

3.1.3 机器架构 ID 寄存器 `marchid`

CSR 是一个 mxlen 位只读寄存器，编码 hart 的基本微架构。该寄存器在任何实现中都必须是可读的，但是可以返回 0 值表示未实现该字段。`mvendorid` 和 `marchid` 的组合应该唯一地标识所实现的 hart 微架构的类型。

图 3.3: Machine Architecture ID register (`marchid`).

开源项目架构 id 由 RISC-V International 全局分配，并且具有非零架构 id，其中最有效位 (MSB) 为零。商业体系结构 id 由每个商业供应商独立分配，但必须设置 MSB，并且不能在剩余的 MXLEN-1 位中包含零。

架构 *ID* 用于代表与开发 *repo* 相关的微架构，而非特定组织。开源设计的商业制造商应（并且可能被 *lisense* 要求）保留原始架构 *ID*。有助于减少碎片化和工具支持成本，并提供归属。开源架构 *ID* 由 *RISC-V International* 管理，只分配给已发布且正在运行的开源项目。商业架构 *ID* 可由任何注册过的厂商独自管理，但诺厂商希望同时使用闭源和开源的微架构，则要拥有与开源架构不相干的 *ID*（*MSB set*），以防冲突。

接下来的实现字段中所采用的协定可用于区分同一架构设计的分支，包括按组织结构划分。`misa` 寄存器还有助于区分不同设计变体。

3.1.4 机器实现 ID 寄存器 `mimpid`

`mimid` CSR 提供处理器实现版本的唯一编码。该寄存器在任何实现中都必须是可读的，但是可以返回 0 值以指示未实现该字段。实现值应该反映 RISC-V 处理器本身的设计，而不是任何周边系统。

图 3.4: Machine Implementation ID register (`mimpid`).

该字段的格式留给体系结构源代码的提供者，但通常由标准工具打印为没有前导或后置零的十六进制字符串，因此可以对 Implementation 值进行左对齐 (即从最有效的啃入开始填充)，子字段按照啃入边界对齐，以方便人类的可读性。

3.1.5 硬件线程 ID 寄存器 mhartid

mhartid CSR 是一个 MXLEN 位只读寄存器，包含运行代码的硬件线程的整数 ID。这个寄存器在任何实现中都必须是可读的。在多处理器系统中，哈特 ID 不一定是连续编号的，但至少有一个哈特 ID 必须为 0。Hart id 在执行环境中必须是唯一的。

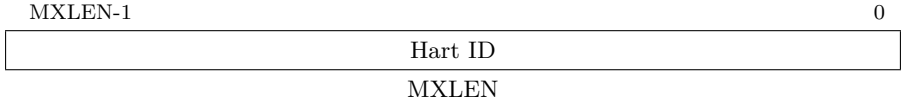


图 3.5: Hart ID register (mhartid).

在某些情况下，我们必须确保只有一个 *hart* 运行某些代码 *(*例如，在 *reset* 时*)*，因此要求一个 *hart* 的已知 *hart ID* 为 *0*。

为了提高效率，系统实现者应该致力于减少系统中使用的最大 *hart ID* 的大小

3.1.6 机器状态寄存器 (mstatus and mstatush)

mstatus 寄存器是一个 MXLEN-bit 的读/写寄存器，格式如图 **??**和图 3.7所示。mstatus 寄存器跟踪和控制 hart 的当前操作状态。mstatus 的受限视图作为 sstatus 寄存器出现在 s 级 ISA 中。

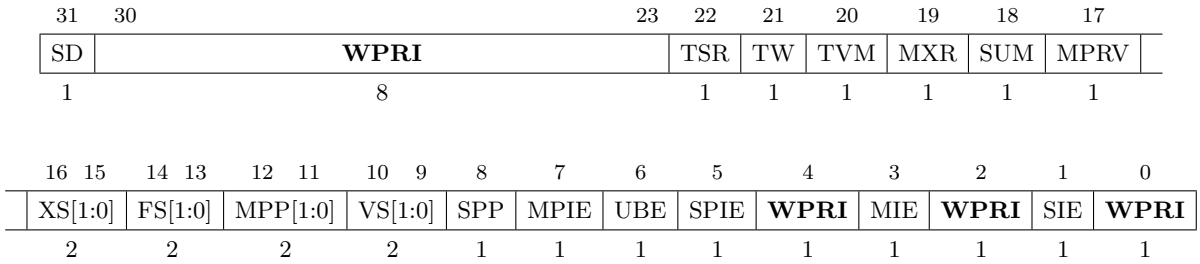


图 3.6: Machine-mode status register (mstatus) for RV32.

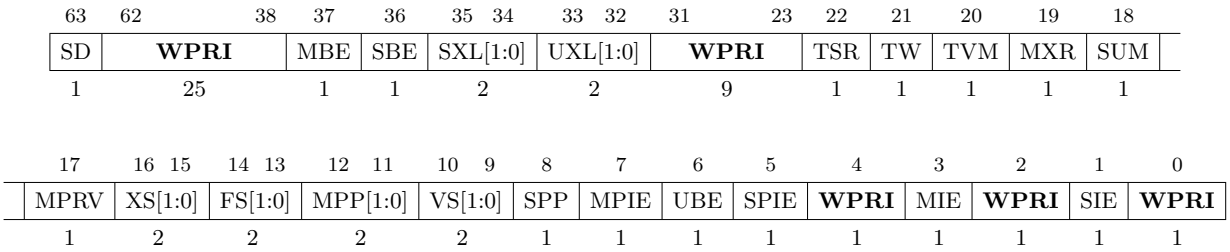


图 3.7: Machine-mode status register (mstatus) for RV64.

仅对于 RV32，`mstatush` 是一个 32 位的读/写寄存器，格式如图 3.8所示。`mstatus` 的第 30:4 位通常包含与 RV64 的 `mstatus` 第 62:36 位相同的字段。`mstatush` 中不存在 SD、SXL 和 UXL 字段。

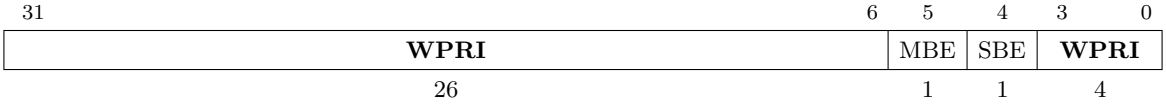


图 3.8: Additional machine-mode status register (`mstatush`) for RV32.

3.1.6.1 `mstatus` 寄存器中的特权和全局中断允许栈

对 m 模式和 s 模式分别提供了全局中断使能位 MIE 和 SIE。这些位主要用于保证与当前特权模式下的中断处理程序相关的原子性。

全局 x*IE* 位位于 `mstatus` 的低阶位中，允许通过单个 *CSR* 指令原子地设置或清除它们

当 hart 以特权模式 x 执行时，当 xIE=1 时全局启用中断，当 xIE=0 时全局禁用中断。低特权模式的中断，w<x，总是全局禁用的，不管低特权模式的任何全局 wIE 位的设置。高特权模式的中断，y>x 总是全局启用的，而不管高特权模式的全局 yIE 位的设置如何。高特权级别的代码可以使用单独的每个中断启用位，在将控制割让给低特权模式之前禁用选定的高特权模式中断。

高特权模式 y 可以在把控制权让给低特权模式之前禁用所有的中断，但这是不寻常的，因为它只会留下同步陷阱、不可屏蔽中断或重置作为重获 *hart* 控制的手段。

为了支持嵌套陷阱，每个可以响应中断的特权模式 x 都有一个两级的支持中断的位和特权模式堆栈。xPIE 保存在 trap 之前激活的中断使能位的值，xPP 保存之前的特权模式。xPP 字段只能持有不超过 x 的特权模式，因此 MPP 是两个比特宽，而 SPP 是一个比特宽。当一个 trap 从特权模式 y 进入特权模式 x 时，xPIE 被设置为 xIE;xIE 设置为 0;

对于较低的特权模式，任何 *trap(*同步或异步*)* 通常在较高的特权模式下执行，并在进入时禁用中断。高级的 *trap* 处理程序将服务于 *trap* 并使用堆叠的信息返回，或者，如果不立即返回到中断的上下文，则在重新启用中断之前保存特权堆栈，因此每个堆栈只需要一个条目。

MRET 或 SRET 指令分别用于从 m 模式或 s 模式的 trap 返回。当执行 xRET 指令时，假设 xPP 持有值 y，xIE 被设为 xPIE; 权限模式更改为 y;xPIE 设置为 1;xPP 设置为最不受特权支持的模式 (如果实现 U 模式则为 U，否则为 M)。如果 xPP≠M，xRET 也设置 MPRV=0。

将 x*PP* 设置为 x*RET* 上支持的特权最少的模式，有助于在两级特权模式堆栈的管理中识别软件错误

*x*PP 字段是 **WARL** 字段，只能持有权限模式 x 和任何低于 x 的实现的权限模式。如果特权模式 x 没有实现，那么 *x*PP 必须是只读的 0。

M 模式软件可以通过将特权模式写入 MPP，然后再将其读取回来，来确定是否实现了特权模式。

如果机器只提供 U 和 M 模式，那么只需要一个硬件存储位来表示 MPP 中的 00 或 11

3.1.6.2 Base ISA Control in `mstatus` Register

对于 RV64 系统，SXL 和 UXL 字段是分别控制 s 模式和 u 模式的 XLEN 值的 **WARL** 字段。这些字段的编码与表 3.1中 `misa` 的 MXL 字段相同。s 模式和 u 模式下的有效 XLEN 分别称为 *SXLEN* 和 *UXLEN*。

对于 RV32 系统，SXL 和 UXL 字段不存在，SXLEN=32 和 UXLEN=32。

对于 RV64 系统，如果不支持 s 模式，则 SXL 为只读零。否则，它是一个编码 SXLEN 当前值的 **WARL** 字段。具体来说，实现可以使 SXL 为只读字段，其值总是确保 SXLEN=MXLEN。

对于 RV64 系统，如果不支持 u 模式，则 UXL 为只读零。否则，它是一个编码 UXLEN 当前值的 **WARL** 字段。具体来说，实现可以使 UXL 为只读字段，其值总是确保 UXLEN=MXLEN 或 UXLEN=SXLEN。

每当将任何模式下的 XLEN 设置为小于最广支持的 XLEN 的值时，所有操作都必须忽略配置的 XLEN 之上的源操作数寄存器位，并且必须对结果进行符号扩展以填充目标寄存器中整个最广支持的 XLEN。类似地，XLEN 上面的 `pc` 位被忽略，当 `pc` 被写入时，它被符号扩展以填充支持最广的 XLEN

我们要求操作总是用定义的值填充整个底层硬件寄存器，以避免实现定义的行为

为了降低硬件复杂性，该体系结构不检查低特权模式的 *XLEN* 设置是否小于或等于下一个更高的特权模式。在实践中，这样的设置几乎总是软件错误，但即使在这种情况下，机器操作也是定义良好的

如果 MXLEN 从 32 更改为更宽的宽度，`mstatus` 字段 SXL 和 UXL，如果不限制为单个值，则得到对应于支持的最宽宽度的值，但不比新的 MXLEN 宽。

3.1.6.3 mstatus 寄存器中的内存特权

MPRV(修改特权) 位修改有效特权模式，即加载和存储执行时的特权级别。当 MPRV=0 时，加载和存储正常，使用当前特权模式的转换和保护机制。当 MPRV=1 时，加载和存储内存地址被转换和保护，并应用端序，就好像当前特权模式被设置为 MPP 一样。指令地址转换和保护不受 MPRV 设置的影响。如果不支持 u 模式，则 MPRV 为只读 0。

如果 MRET 或 SRET 指令将特权模式更改为特权低于 M 的模式，也会将 MPRV 设置为 0。

MXR(使可执行文件可读) 位修改负载访问虚拟内存的特权。当 MXR=0 时，只有从标记为可读的页面加载 (图 4.33中的 R=1) 才会成功。当 MXR=1 时，从标记为可读或可执行 (R=1 或 X=1) 的页面加载将成功。当基于页面的虚拟内存不生效时，MXR 没有作用。如果不支持 s 模式，则 MXR 为只读 0。

MPRV 和 MXR 机制的构想是为了提高 m 模式例程的效率，这些例程模拟了缺失的硬件特征，例如，错位的负载和存储。MPRV 消除了在软件中执行地址转换的需要。MXR 允许从标记为“仅执行”的页面加载指令字。

当前特权模式和 MPP 指定的特权模式可能具有不同的 XLEN 设置。当 MPRV=1 时，加载和存储内存地址就像当前的 XLEN 被设置为 MPP 的 XLEN 一样，遵循 Section **??** *中的规则。*

SUM(允许 Supervisor User Memory 访问) 位修改 s 模式加载和存储访问虚拟内存的特权。当 SUM=0 时，s 模式内存访问可通过 U 模式访问的页面 (图 4.33中的 U=1) 将出错。当 SUM=1 时，允许这些访问。当基于页面的虚拟内存无效时，SUM 将不起作用。注意，当不以 S 模式执行时，SUM 通常被忽略，但当 MPRV=1 和 MPP=S 时，SUM *is* 有效。如果不支持 s 模式或者 `satp`. SUM 为只读 0。模式是只读 0。

MXR 和 SUM 机制只影响对页表项中编码的权限的解释。特别是，它们对由于 pma 或 PMP 引发的访问故障异常没有影响。

3.1.6.4 mstatus 和 mstatush 寄存器中的端序控制

`mstatus` 和 `mstatush` 中的 MBE, SBE 和 UBE 位是 **WARL** 字段，控制内存访问的端序，而不是指令获取。指令获取总是小端式的。

MBE 控制从 m 模式 (假设 `mstatus`. mprv =0) 进行的非指令获取内存访问是小端式 (MBE=0) 还是大端式 (MBE=1)。

如果不支持 s 模式，则 SBE 只读 0。否则，SBE 控制从 s 模式进行的显式加载和存储内存访问是小端 (SBE=0) 还是大端 (SBE=1)。

如果不支持 u 模式，则 UBE 为只读 0。否则，UBE 控制从 u 模式进行的显式加载和存储内存访问是小端 (UBE=0) 还是大端 (UBE=1)。

对于隐式访问管理器级内存管理数据结构 (如页表)，字节顺序总是由 SBE 控制。由于更改 SBE 会改变对这些数据结构的实现解释，如果在更改到 SBE 期间仍在使用任何这样的数据结构，m 模式软件必须通过执行 SFENCE 来遵循此更改到 SBE。VMA 指令，*rs1*=x0 和 *rs2*=x0。

只有在人为设计的场景中，给定的内存管理数据结构才会被解释为小端和大端。在实践中，*SBE* 只会在运行时在世界交换机上更改，在这种情况下，新旧内存管理数据结构都不会以不同的端序重新解释。在本例中，不需要附加 *SFENCE*。*VMA* 是必要的，超出了世界切换的通常要求。

如果支持 s 模式，则实现可以使 SBE 成为 MBE 的只读副本。如果支持 u 模式，则实现可以使 UBE 成为 MBE 或 SBE 的只读副本。

如果字段 *MBE*、*SBE* 和 *UBE* 都是只读的 *0*，则实现只支持小端内存访问。如果 *MBE* 为只读 *1*，且 *SBE* 和 *UBE* 在支持 *s* 模式和 *u* 模式时都为只读 *1*，则实现只支持大端存储器访问 *(*不包括指令获取*)*。

卷 *I* 将 *hart* 的地址空间定义为在连续地址上由 2^{XLEN} 字节组成的循环序列。地址和字节位置之间的通信是固定的，不受任何端序模式的影响。相反，适用的端序模式决定了内存字节和多字节数量 *(*半字、字等*)* 之间的映射顺序。

标准的 *RISC-V abi* 被期望为纯粹的小端 *only* 或大端 *only*，不允许混合端数。然而，已经定义了端序控制以允许，例如，具有一个端序的操作系统可以执行相反端序的用户模式程序。还考虑了非标准使用的可能性，软件可以根据需要翻转内存访问的端序。

RISC-V 指令是统一的小端序，以将指令编码与当前的端序设置解耦，这对硬件和软件都有好处。否则，例如，*RISC-V* 汇编器或反汇编器将始终需要知道预期的活动端序，尽管端序模式可能在执行过程中动态更改。相反，通过给指令一个固定的端序，有时精心编写的软件即使在二进制形式中也可能是端序不可知的，就像位置无关的代码一样。

然而，对于编码或解码机器指令的 *RISC-V* 软件来说，只让指令具有小端序的选择确实会产生影响。在大端模式下，这样的软件必须考虑到显式加载和存储的端序与指令的端序相反，例如通过在加载后和存储前交换字节顺序。

3.1.6.5 mstatus 寄存器中的虚拟化支持

TVM (Trap 虚拟内存) 位是一个支持拦截主管虚拟内存管理操作的 **WARL** 字段。当 TVM=1 时，尝试读或写 `satp` CSR 或执行 SFENCE。影响或 SINVAL。VMA 指令在 s 模式下执行时会引发非法指令异常。当 TVM=0 时，在 s 模式下允许这些操作。当不支持 s 模式时，TVM 只读为 0。

TVM 机制允许客户操作系统以 s 模式执行，而不是传统的以 u 模式虚拟化它们，从而提高了虚拟化效率。这种方法不需要捕获对大多数 s 模式 csr 的访问。

捕获 `satp` 访问和 SFENCE。VMA 和 SINVAL。VMA 指令提供了延迟填充影子页表所需的挂钩。

TW(超时等待) 位是一个支持拦截 WFI 指令的 **WARL** 字段 (参见 **??**小节)。当 TW=0 时，WFI 指令可能以较低的特权模式执行，如果没有因其他原因而被阻止。当 TW=1 时，如果 WFI 以任何特权较低的模式执行，并且它没有在特定于实现的有限时间限制内完成，那么 WFI 指令将导致一个非法指令异常。时间限制可能总是 0，在这种情况下，当 TW=1 时，WFI 总是在特权较低的模式下引起非法指令异常。当没有低于 M 的模式时，TW 为只读 0。

捕获 WFI 指令可以触发世界切换到另一个客户操作系统，而不是浪费地在当前客户系统中闲置。

当实现 s 模式时，以 u 模式执行 WFI 会导致非法指令异常，除非它在特定于实现的有限时间内完成。该规范的未来修订版可能会添加一个特性，允许 s 模式有选择地允许 u 模式下的 WFI。这样的功能只有在 TW=0 时才会被激活。

TSR (Trap SRET) 位是一个支持拦截管理异常返回指令 SRET 的 **WARL** 字段。当 TSR=1 时，试图在 s 模式下执行 SRET 将引发非法指令异常。当 TSR=0 时，s 模式下允许该操作。当不支持 s 模式时，TSR 为只读 0。

捕获 SRET 对于在没有提供 SRET 的实现上模拟 hypervisor 扩展 (参见第 八章) 是必要的。

3.1.6.6 Extension Context Status in mstatus Register

支持大量的扩展是 RISC-V 的主要目标之一，因此我们定义了一个标准接口，以允许未更改的特权模式代码 (特别是管理员级操作系统) 支持任意的用户模式状态扩展。

到目前为止，V 扩展是唯一定义了浮点 CSR 和数据寄存器之外的附加状态的标准扩展。

FS[1:0] 和 VS[1:0] **WARL** 字段和 XS[1:0] 只读字段分别用于通过设置和跟踪浮点单元和任何其他用户模式扩展的当前状态来减少上下文保存和恢复的成本。FS 字段编码浮点单元状态的状态，包括浮点寄存器 `f0`——`f31` 和 csr `fcsr`，`frm` 和 `fflag`。VS 字段编码向量扩展状态的状态，包括向量寄存器 `v0`——`v31` 和 CSRsvcsr, `vxrm`, `vxsat`, `vstart`, `vl`, `vtype` 和 `vlenb`.。XS 字段编码其他用户模式扩展的状态和相关状态。上下文切换例程可以检查这些字段，以快速确定是否需要状态保存或恢复。如果需要保存或恢复，通常需要附加的指令和 csr 来实现和优化流程。

该设计预期大多数上下文开关不需要在浮点单元或其他扩展中的任何一个或两个中保存/恢复状态，因此通过 *SD* 位提供了快速检查。

FS、VS 和 XS 字段使用相同的状态编码，如表 3.3所示，四种可能的状态值分别为 Off、Initial、Clean 和 Dirty。

Status	FS and VS Meaning	XS Meaning
0	Off	All off
1	Initial	None dirty or clean, some on
2	Clean	None dirty, some clean
3	Dirty	Some dirty

表 3.3: Encoding of FS[1:0], VS[1:0], and XS[1:0] status fields.

如果实现了 F 扩展，FS 字段不能只读为零。

如果既没有实现 F 扩展也没有实现 s 模式，则 FS 为只读 0。如果实现了 s 模式，但 F 扩展没有实现，FS 可以选择为只读零。

允许但不需要 *F* 扩展的 *s* 模式实现将 *FS* 字段设为只读零。一些这样的实现将选择 not 使 *FS* 字段为只读零，以便通过不可见的陷阱模拟 *s* 模式和 *u* 模式的 *F* 扩展到 *m* 模式。

如果实现了 `v` 寄存器，VS 字段不应该是只读零。

如果既没有实现 `v` 寄存器也没有实现 s 模式，那么 VS 只读为零。如果实现了 s 模式，但 `v` 寄存器没有实现，VS 可以有选择地为只读零。

在没有需要新状态的额外用户扩展的系统中，XS 字段为只读零。每个附加的带有状态的扩展都提供一个 CSR 字段，该字段编码与 XS 状态等价的内容。XS 字段表示所有扩展状态的摘要，如表 3.3所示。

XS 字段有效地报告了所有用户扩展状态字段的最大状态值，尽管个别扩展可以使用与 *XS* 不同的编码。

SD 位是一个只读位，它总结 FS、VS 或 XS 字段是否表示存在某种脏状态，需要将扩展的用户上下文保存到内存中。如果 FS、XS 和 VS 都是只读零，那么 SD 也总是零。

当一个扩展的状态被设置为 Off 时，任何试图读或写相应状态的指令都会导致一个非法指令异常。当状态为 Initial 时，对应的状态应该有一个初始常数值。当状态为 Clean 时，对应的状态可能与初始值不同，但与存储在上下文交换中的最后一个值匹配。当状态为 Dirty 时，自上一次保存上下文以来，相应的状态可能已被修改。

在保存上下文期间，负责的特权代码只需要在状态为 Dirty 时写出相应的状态，然后可以将扩展的状态重置为 Clean。在上下文恢复期间，只有状态为 Clean 时才需要从内存中加载上下文 (在恢复时它不应该是 Dirty)。如果状态为 Initial，则必须在上下文恢复时将上下文设置为初始常量值，以避免安全漏洞，但这可以在不访问内存的情况下完成。例如，浮点寄存器都可以初始化为直接值 0。

FS 和 XS 字段在保存上下文之前由特权代码读取。FS 字段在恢复用户上下文时由特权代码直接设置，而 XS 字段则通过写入各个扩展的状态寄存器来间接设置。状态字段也将在指令执行期间更新，而不考虑特权模式。

用户模式 ISA 的扩展通常包括额外的用户模式状态，并且该状态可能比基本整数寄存器大得多。扩展可能只用于某些应用程序，或者可能只用于单个应用程序中的较短阶段。为了提高性能，用户模式扩展可以定义额外的指令，以允许用户模式软件将单元返回到初始状态，甚至关闭单元。

例如，协处理器可能需要在使用前进行配置，而在使用后可以“未配置”。未配置的状态将表示为用于上下文保存的 Initial 状态。如果同一个应用程序在 unconfigure 和下一个 configure 之间仍然在运行 (这会将状态设置为 Dirty)，那么实际上没有必要在 unconfigure 指令中重新初始化状态，因为所有的状态都是用户进程的本地状态，也就是说，Initial 状态可能只会导致协处理器状态在上下文恢复时被初始化为一个恒定值，而不是在每个 unconfigure 时。

执行一个用户模式指令来禁用一个单元并将其置于 Off 状态将导致一个非法指令异常，如果任何后续指令试图在单元被重新打开之前使用它。打开单元的用户模式指令还必须确保单元的状态已正确初始化，因为单元可能已被另一个上下文使用。

更改 FS 的设置对浮点寄存器状态的内容没有影响。特别地，设置 FS=Off 不会破坏状态，设置 FS=Initial 也不会清除内容。类似地，VS 的设置对向量寄存器状态的内容没有影响。但是，其他扩展在设置为 Off 时可能不会保存状态。

实现可能会选择不精确地跟踪浮点寄存器状态的脏性，即使它没有被修改，也会报告该状态是脏性的。在某些实现中，一些不改变浮点状态的指令可能导致状态从 Initial 或 Clean 转换为 Dirty。在其他实现中，脏程度可能根本不会被跟踪，在这种情况下，有效的 FS 状态是 Off 和 Dirty，试图将 FS 设置为 Initial 或 Clean 会导致它被设置为 Dirty。

FS 的这个定义并不禁止由于错误的推测而将 *FS* 设置为 *Dirty*。一些平台可能会选择不允许投机性写入 *FS* 来关闭潜在的侧通道。

如果一条指令显式或隐式地写入浮点寄存器或 `fcsr` 但不更改其内容，并且 FS=Initial 或 FS=Clean，则 FS 是否转换为 Dirty 由实现定义。

实现可以选择以类似的不精确的方式跟踪矢量寄存器状态的脏程度，包括可能在软件试图设置 VS=Initial 或 VS=Clean 时将 VS 设置为 Dirty。当 VS=Initial 或 VS=Clean 时，写入向量寄存器或向量 CSR 但不改变其内容的指令是否导致 VS 过渡到 Dirty 是由实现定义的。

表 3.4显示了 FS、VS 或 XS 状态位的所有可能的状态转换。注意，标准浮点和向量扩展不支持用户模式的取消配置或禁用/启用指令。

表 3.4显示了 FS、VS 或 XS 状态位的所有可能的状态转换。注意，标准浮点和向量扩展不支持用户模式的取消配置或禁用/启用指令。

许多协处理器扩展仅在有限的上下文中使用，这些上下文允许软件在完成后安全地取消配置甚至禁用单元。这减少了大型有状态协处理器的上下文切换开销。

我们将浮点状态与其他扩展状态分开，因为当浮点单元存在时，浮点寄存器是标准调用约定的一部分，因此用户模式软件无法知道何时禁用浮点单元是安全的。

XS 字段提供了所有添加的扩展状态的摘要，但是附加的微体系结构位可以被保持在扩展中，以进一步减少上下文保存和恢复开销。

SD 位为只读位，当 FS、VS 或 XS 位编码脏状态时置 1(即 SD= (FS==11) 或 (XS==11) 或 (VS==11))。这允许特权代码快速确定除整数寄存器组和 PC 之外何时不需要额外的上下文保存。

浮点单元状态总是使用标准指令 (F、D 和/或 Q) 进行初始化、保存和恢复，特权代码必须知道 FLEN，以确定为每个 `f` 寄存器保留的适当空间。

机器模式和管理模式共享 FS、VS 和 XS 位的单个副本。管理程序级软件通常直接使用 FS、VS 和 XS 位来记录与管理程序级保存的上下文相关的状态。机器级软件在相应版本的上下文中保存和恢复扩展状态时必须更加保守。

在任何合理的用例中，用户和管理员级别之间的上下文切换次数应该远远超过其他权限级别的上下文切换次数。注意，协处理器不应该要求保存和恢复它们的上下文来服务异步中断，除非中断导致用户级上下文交换。

Current State Action	Off	Initial	Clean	Dirty
At context save in privileged code				
Save state? Next state	No Off	No Initial	No Clean	Yes Clean
At context restore in privileged code				
Restore state? Next state	No Off	Yes, to initial Initial	Yes, from memory Clean	N/A N/A
Execute instruction to read state				
Action? Next state	Exception Off	Execute Initial	Execute Clean	Execute Dirty
Execute instruction that possibly modifies state, including configuration				
Action? Next state	Exception Off	Execute Dirty	Execute Dirty	Execute Dirty
Execute instruction to unconfigure unit				
Action? Next state	Exception Off	Execute Initial	Execute Initial	Execute Initial
Execute instruction to disable unit				
Action? Next state	Execute Off	Execute Off	Execute Off	Execute Off
Execute instruction to enable unit				
Action? Next state	Execute Initial	Execute Initial	Execute Initial	Execute Initial

表 3.4: FS, VS, and XS state transitions.

3.1.7 机器陷阱向量基地址寄存器 (`mtvec`)

`mtvec` 寄存器是一个 MXLEN 位 **WARL** 读/写寄存器，保存陷阱向量配置，包括向量基址 (base) 和向量模式 (mode)。

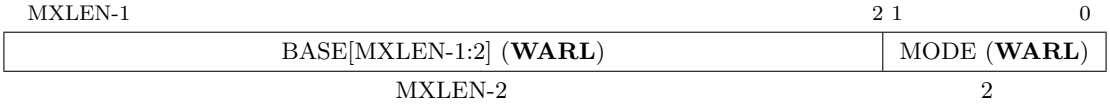


图 3.9: Machine trap-vector base-address register (`mtvec`).

mtvec 寄存器必须始终执行，但可以包含只读值。如果 mtvec 是可写的，则寄存器可能保存的一组值可能因实施而异。基本字段中的值必须始终在 4 字节边界上对齐，并且模式设置可能会对基本字段中的值施加额外的对齐约束。

我们允许在陷阱向量基地址的实现中有相当大的灵活性。一方面，我们不希望低端实现增加大量状态位的负担，但另一方面，我们希望允许更大系统的灵活性。

Value	Name	Description
0	Direct	All exceptions set pc to BASE.
1	Vectored	Asynchronous interrupts set pc to BASE+4×cause.
≥2	—	*Reserved*

表 3.5: Encoding of mtvec MODE field.

模式字段的编码如表 3.5所示。当 MODE=Direct 时，所有进入机器模式的陷阱会将 pc 设置为基址字段中的地址。当 MODE=Vectored 时，进入机器模式的所有同步异常导致 pc 被设置为基址域中的地址，而中断导致 pc 被设置为基址域中的地址加上四倍的中断原因号。例如，机器模式定时器中断 (参见第页的表 3.6) 导致 pc 被设置为 BASE+0x1c。

当向量中断使能时，中断原因 *0(对应于用户模式软件中断)* 被向量化到与同步异常相同的位置。这种模糊性在实践中不会出现，因为用户模式软件中断要么被禁用，要么被委托给用户模式。

对于不同的模式，实现可以具有不同的对准约束。特别是，MODE=Vectored 可能比 MODE=Direct 具有更严格的对齐约束。

在矢量化模式中允许较粗略的对准使得矢量化能够在没有硬件加法器电路的情况下实现。

平台规范中给出了复位和 *NMI* 向量的位置。

3.1.8 机器陷阱委托寄存器 (medeleg 和 mideleg)

默认情况下，任何特权级别的所有陷阱都在机器模式下处理，尽管机器模式处理程序可以使用 MRET 指令将陷阱重定向回适当级别 (第 3.3.2节)。为了提高性能，实现可以在 medeleg 和-mideleg 中提供单独的读/写位，以指示某些异常和中断应该由较低的特权级别直接处理。机器异常委托寄存器 (medeleg) 和机器中断委托寄存器 (mideleg) 是 MXLEN 位读/写寄存器。

在版本 1.9.1 及更早的版本中，这些寄存器存在，但仅在 M 模式或没有 N 个系统的 M/U 系统中硬连接到 0。在这些情况下，没有理由要求它们返回零，因为 misa 寄存器指示它们是否存在。

当陷阱被委派到 S 模式时，将使用陷阱原因写入 `scause` 寄存器；使用获取陷阱的指令的虚拟地址写入 `Sepc` 寄存器；使用特定于异常的数据写入 `stval` 寄存器；使用陷阱发生时的活动特权模式写入 `mstatus` 的 SPP 字段；使用陷阱发生时的 SIE 字段的值写入 `matus` 的 SPIE 字段；清除 `mStatus` 的 SIE 字段。未写入 `mason`、`mepc` 和 `mtval` 寄存器以及 `mstatus` 的 MPP 和 MPIE 字段。

实现可以选择将可委托陷阱设置为子集，通过向每个位位置写入一个来找到受支持的可委派位，然后读回 `medelig` 或 `mideleg` 中的值，以查看哪些位位置保存 1。

一个实现不应该有任何位的 `medelegg` 是只读的，即任何可以委托的同步陷阱必须支持不委托。类似地，实现不应将对应于机器级中断的任何位的 `mideleg` 固定为只读 (但可以对较低级别的中断这样做)。

版本 1.11 及更早版本禁止任何位的 `mideleg` 为只读。平台标准可能总是会添加这样的限制。

陷阱永远不会从特权较高的模式转换到特权较低的模式。例如，如果 M 模式已将非法指令异常委托给 S 模式，而 M 模式软件稍后执行了一条非法指令，则陷阱将在 M 模式中进行，而不是委托给 S 模式。相比之下，陷阱可以水平采取。使用相同的示例，如果 M 模式已将非法指令异常委托给 S 模式，并且 S 模式软件稍后执行了非法指令，则在 S 模式中进行陷阱。

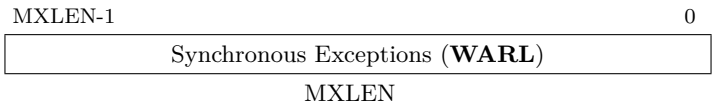


图 3.10: Machine Exception Delegation Register `medeleg`.

`medeleg` 为页面 **??** 上的 Table **??** 中所示的每个同步异常分配了一个位位置，位位置的索引等于 `mcause` 中返回的值 寄存器（即，设置位 8 允许将用户模式环境调用委托给较低权限的陷阱处理程序)。

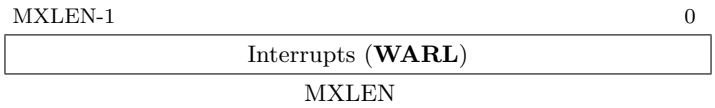


图 3.11: Machine Interrupt Delegation Register `mideleg`.

mideleg 保存各个中断的陷阱委托位，位布局与 mip 寄存器中的位布局相匹配 (即，STIP 中断委托控制位于第 5 位)。

对于在较低特权模式下不能发生的异常，相应的 medeleg 位应为只读零。特别是，medeleg[11] 是只读零。

3.1.9 机器中断寄存器 (mip 和 mie)

mip 寄存器是一个 MXLEN 位读/写寄存器，包含有关挂起中断的信息，而 mie 是相应的 MXLEN 位读/写寄存器，包含中断使能位。中断原因编号 i（如 CSR mcause，Section 3.1.15 中所述）与 mip 和 mie 中的 bit i 对应。位 15:0 仅分配给标准中断原因，而位 16 及更高位指定用于平台或自定义使用。

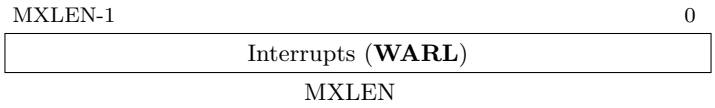


图 3.12: Machine Interrupt-Pending Register (mip).

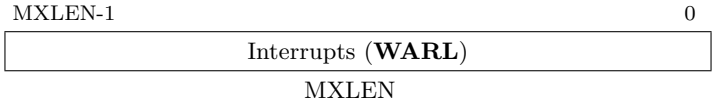


图 3.13: Machine Interrupt-Enable Register (mie).

如果满足以下所有条件，则中断 i 将陷入 M 模式（导致特权模式更改为 M 模式）：(a) 当前特权模式为 M 且 MIE 位在 mstatus 寄存器被设置，或者当前特权模式的特权低于 M-mode；(b) bit i 在 mip 和 mie 中都设置；(c) 如果寄存器 mideleg 存在，位 i 在 mideleg 中没有设置。

中断陷阱发生的这些条件必须在中断在 mip 中挂起或停止挂起时的有限时间内进行评估，并且还必须在执行 xRET 指令或对这些中断陷阱条件明确依赖的 CSR 的显式写入（包括 mip、mie、mstatus 和 mideleg）。

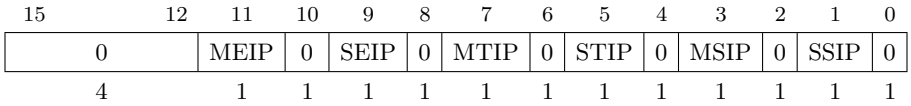


图 3.14: Standard portion (bits 15:0) of mip.

机器级中断寄存器处理一些根中断源，为简单起见，这些中断源被分配了固定的服务优先级，而单独的外部中断控制器可以在更大的一组中断上实现更复杂的优先级方案，然后将这些中断多路复用到机器级中断源。

15 12	11	10	9	8	7	6	5	4	3	2	1	0
0	MEIE	0	SEIE	0	MTIE	0	STIE	0	MSIE	0	SSIE	0
4	1	1	1	1	1	1	1	1	1	1	1	1

图 3.15: Standard portion (bits 15:0) of `mie`.

不可屏蔽中断通过 `mip` 寄存器不可见，因为在执行 *NMI* 陷阱处理程序时隐式知道它的存在。

位 `mip`.MEIP 和 `mie`.MEIE 是机器级外部中断的中断挂起和中断使能位。MEIP 在 `mip` 中是只读的，由特定于平台的中断控制器设置和清除。

位 `mip`.MTIP 和 `mie`.MTIE 是机器定时器中断的中断挂起和中断启用位。MTIP 在 `mip` 中是只读的，并通过写入内存映射机器模式定时器比较寄存器来清除。

位 `mip`.MSIP 和 `mie`.MSIE 是机器级软件中断的中断挂起和中断启用位。MSIP 在 `mip` 中是只读的，并且通过访问内存映射控制寄存器来写入，远程 hart 使用这些控制寄存器来提供机器级处理器间中断。hart 可以使用相同的内存映射控制寄存器写入自己的 MSIP 位。如果系统只有一个 hart，或者如果平台标准支持通过外部中断 (MEI) 传递机器级处理器间中断，则 `mip`.MSIP 和 `mie`.MSIE 都可以被只读零。

如果未实现主管模式，则 `mip` 的位 SEIP、STIP 和 SSIP 以及 `mie` 的 SEIE、STIE 和 SSIE 为只读零。

如果实现了监督模式，位 `mip`.SEIP 和 `mie`.SEIE 是监督级外部中断的中断挂起和中断使能位。SEIP 在 `mip` 中是可写的，并且可以由 M-mode 软件写入以向 S-mode 指示外部中断正在挂起。此外，平台级中断控制器可以产生监督级外部中断。基于软件可写 SEIP 位的逻辑或和来自外部中断控制器的信号，使管理级外部中断处于待处理状态。当使用 CSR 指令读取 `mip` 时，在 `rd` 目标寄存器中返回的 SEIP 位的值是软件可写位与来自中断控制器的中断信号的逻辑或，但来自中断控制器的信号不用于计算写入 SEIP 的值。只有软件可写 SEIP 位参与 CSRRS 或 CSRRC 指令的读-修改-写序列。

例如，如果我们将软件可写 *SEIP* 位 `B` 和来自外部中断控制器的信号命名为 `E`，那么如果执行 `csrrs t0, mip, t1`， *tt t0[9]* 用 `B || 编写 E`，则 `B` 写成 `B || t1[9]`。如果 `csrrw t0, mip, t1` 被执行，那么 `t0[9]` 写成 `B || E`，而 `B` 简单地写成 `t1[9]`。在这两种情况下，`B` 都不依赖于 `E`。*SEIP* 字段行为旨在允许更高特权层干净地模仿外部中断，而不会丢失任何真正的外部中断。因此，*CSR* 指令的行为与常规 *CSR* 访问相比略有修改。

如果实施监督模式，位 `mip`.STIP 和 `mie`.STIE 是监督级定时器中断的中断挂起和中断使能位。STIP 在 `mip` 中是可写的，并且可以由 M 模式软件编写以将定时器中断传递到 S 模式。

如果实现了监督模式，位 mip.SSIP 和 mie.SSIE 是监督级软件中断的中断挂起和中断使能位。SSIP 在 mip 中是可写的，也可以由特定于平台的中断控制器设置为 1。

以 M 模式为目标的多个同时中断按以下优先级降序处理：MEI、MSI、MTI、SEI、SSI、STI。

机器级中断固定优先级排序规则是根据以下原理开发的。

较高特权模式的中断必须在较低特权模式的中断之前服务以支持抢占。

位 *16* 及以上的平台特定机器级中断源具有特定于平台的优先级，但通常选择具有最高服务优先级以支持非常快速的本地向量中断。

外部中断在内部（定时器/软件）中断之前处理，因为外部中断通常由可能需要低中断服务时间的设备生成。

软件中断在内部定时器中断之前处理，因为内部定时器中断通常用于时间片，其中时间精度不太重要，而软件中断用于处理器间消息传递。当需要高精度定时时可以避免软件中断，或者可以通过不同的中断路径路由高精度定时器中断。软件中断位于 mip 的最低四位，因为它们通常由软件编写，并且该位置允许使用具有五位立即数的单个 *CSR* 指令。

mip 和 mie 寄存器的受限视图显示为主管级别的 sip 和 sie 寄存器。如果通过在 mideleg 寄存器中设置一个位将中断委托给 S 模式，则它在 sip 寄存器中变得可见，并且可以使用 sie 寄存器进行屏蔽。否则，sip 和 sie 中的对应位为只读零。

3.1.10 硬件性能监视器

M 模式包括一个基本的硬件性能监控工具。mcycle CSR 计算运行 hart 的处理器内核执行的时钟周期数。minstret CSR 计算 hart 已退出的指令数。mcycle 和 minstret 寄存器在所有 RV32 和 RV64 系统上都具有 64 位精度。

计数器寄存器在 hart 复位后具有任意值，并且可以写入给定值。任何 CSR 写操作在写指令完成后才生效。mcycle CSR 可以在同一内核上的 harts 之间共享，在这种情况下，对 mcycle 的写入将对这些 harts 可见。平台应提供一种机制来指示哪些 harts 共享一个 mcycle CSR。

硬件性能监视器包括 29 个额外的 64 位事件计数器，mhpmcounter3–mhpmcounter31。事件选择器 CSR，mhpmevent3–mhpmevent31，是 MXLEN 位 **WARL** 寄存器，用于控制哪个事件导致相应计数器递增。这些事件的含义由平台定义，但事件 0 被定义为“无事件”。所有计数器都应该实现，但合法的实现是使计数器及其对应的事件选择器都是只读的 0。

mhpmcounter 是 **WARL** 寄存器，在 RV32 和 RV64 上支持高达 64 位精度。

本规范的未来修订版将定义一种机制，以在硬件性能监视器计数器溢出时生成中断。

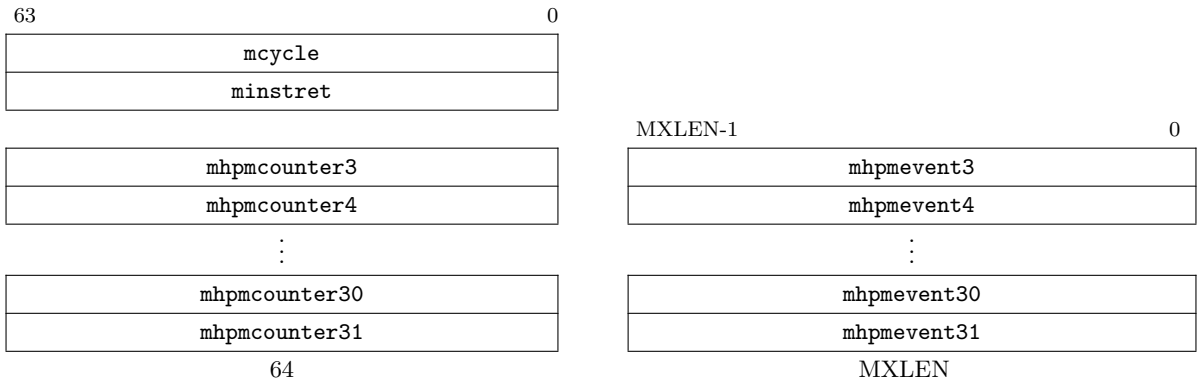


图 3.16: Hardware performance monitor counters.

当 MXLEN=32 时，`mcycle`、`minstret` 和 `mhpmcounter`*n* CSR 的读取返回相应计数器的位 31–0，而写入仅更改位 31- -0; `mcycleh`、`minstreth` 和 `mhpmcounter`*n*`h` CSR 的读取返回相应计数器的位 63–32，而写入仅更改位 63–32。

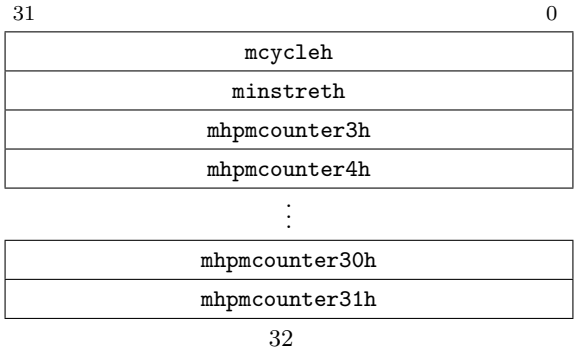


图 3.17: Upper 32 bits of hardware performance monitor counters, RV32 only.

3.1.11 机器计数器使能寄存器 (mcounteren)

计数器启用寄存器 `mcounteren` 是一个 32 位寄存器，用于控制硬件性能监视计数器在下一个最低特权模式下的可用性。

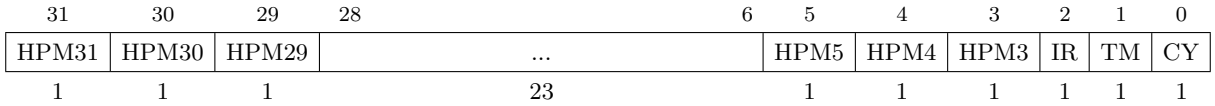


图 3.18: Counter-enable register (`mcounteren`).

此寄存器中的设置仅控制可访问性。读取或写入此寄存器的行为不会影响底层计数器，即使在不可访问时也会继续递增。

当 mcounteren 寄存器中的 CY、TM、IR 或 HPMn 位清零时，尝试读取 cycle、time、instret 或 hpmcounter$n$ 寄存器在 S-mode 或 U-mode 执行时会导致非法指令异常。当这些位中的一个被设置时，允许在下一个实现的特权模式下访问相应的寄存器（如果实现了 S-mode，否则为 U-mode）。

计数器启用位以最少的硬件支持两种常见用例。对于不需要高性能计时器和计数器的系统，机器模式软件可以捕获访问并在软件中实现所有功能。对于需要高性能计时器和计数器但不关心混淆底层硬件计数器的系统，计数器可以直接暴露于较低特权模式。

cycle、instret 和 hpmcounter$n$ CSR 是 mcycle、minstret 和 mhpmcounter em n，分别。time CSR 是内存映射 mtime 寄存器的只读影子。类似地，在 RV32I 上，cycleh、instreth 和 hpmcounter$n$h CSR 是 mcycleh、minstreth 和 mhpmcounter$n$h，分别。在 RV32I 上，timeh CSR 是内存映射 mtime 寄存器的高 32 位的只读遮蔽，而 time 仅隐藏 mtime 的低 32 位.

实现可以将 time 和 timeh *CSR* 的读取转换为对内存映射 mtime 寄存器的加载，或在 *M* 模式软件中模拟此功能。

在具有 U 模式的系统中，必须实现 mcounteren，但所有字段都是 **WARL** ，并且可能是只读零，表示读取相应计数器会导致在 less- 中执行时出现非法指令异常特权模式。在没有 U 模式的系统中，mcounteren 寄存器不应该存在。

3.1.12 机器计数器抑制寄存器 (mcountinhibit)

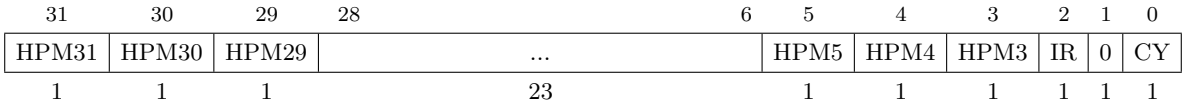


图 3.19: Counter-inhibit register mcountinhibit.

计数器禁止寄存器 mcountinhibit 是一个 32 位 **WARL** 寄存器，用于控制哪些硬件性能监控计数器递增。该寄存器中的设置仅控制计数器是否递增；它们的可访问性不受此寄存器设置的影响。

当 mcountinhibit 寄存器中的 CY、IR 或 HPMn 位清零时，cycle、instret 或 hpmcounter$n$ 寄存器像往常一样递增。当 CY、IR 或 HPMn 位置位时，相应的计数器不会增加。

mcycle CSR 可以在同一内核上的 harts 之间共享，在这种情况下 mcountinhibit.CY 字段也在这些 harts 之间共享，因此对 mcountinhibit.CY 的写入将是可见的对那些小鹿。

如果未实现 mcountinhibit 寄存器，则实现的行为就像寄存器设置为零一样。

当不需要 cycle 和 instret 计数器时，最好有条件地禁止它们以减少能量消耗。提供单个 *CSR* 来禁止所有计数器还允许对计数器进行原子采样。

因为 time 计数器可以在多个内核之间共享，所以不能用 mcountinhibit 机制来禁止它。

3.1.13 机器 scratch 寄存器 (mscratch)

mscratch 寄存器是一个 MXLEN 位读/写寄存器，专用于机器模式。通常，它用于保存指向机器模式 hart-local 上下文空间的指针，并在进入 M 模式陷阱处理程序时与用户寄存器交换。

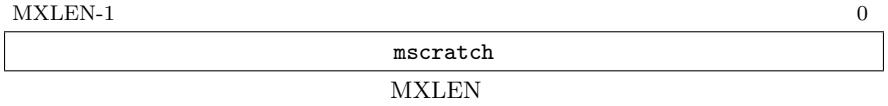


图 3.20: Machine-mode scratch register.

MIPS ISA 分配了两个用户寄存器 (k0/k1) 供操作系统使用。尽管 *MIPS* 方案提供了快速和简单的实现，但它也减少了可用的用户寄存器，并且无法扩展到更高的特权级别或嵌套陷阱。它还可以要求在返回用户级别之前清除两个寄存器，以避免潜在的安全漏洞并提供确定性的调试行为。

RISC-V 用户 *ISA* 旨在支持许多可能的特权系统环境，因此我们不想用任何依赖于操作系统的功能感染用户级 *ISA*。*RISC-V CSR* 交换指令可以快速将值保存/恢复到 mscratch 寄存器。与 *MIPS* 设计不同，操作系统可以依赖于在用户上下文运行时在 mscratch 寄存器中保存一个值。

3.1.14 机器异常程序计数器 (mepc)

mepc 是一个 MXLEN 位读/写寄存器，格式如图 3.21。mepc (mepc[0]) 的低位始终为零。在仅支持 IALIGN=32 的实现上，两个低位 (mepc[1:0]) 始终为零。

如果实现允许 IALIGN 为 16 或 32（例如，通过更改 CSR misa），则每当 IALIGN=32 时，位 mepc[1] 在读取时被屏蔽，因此它看起来为 0。此屏蔽也发生在 MRET 指令的隐式读取中。尽管被屏蔽，但当 IALIGN=32 时，mepc[1] 仍然是可写的。

mepc 是一个 **WARL** 寄存器，它必须能够保存所有有效的虚拟地址。它不需要能够保存所有可能的无效地址。在写入 mepc 之前，实现可能会将无效地址转换为 mepc 能够保存的其他一些无效地址。

当地址转换无效时，虚拟地址和物理地址是相等的。因此，地址集合 mepc 必须能够表示包括可以用作有效 pc 或有效地址的物理地址集合。

当陷阱进入 M 模式时，mepc 被写入被中断或遇到异常的指令的虚拟地址。否则，mepc 永远不会由实现编写，尽管它可能由软件显式编写。

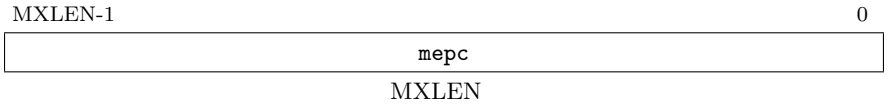


图 3.21: Machine exception program counter register.

3.1.15 机器成因寄存器 (mcause)

mcause 寄存器是一个 MXLEN 位读写寄存器，其格式如图 3.22 所示。当陷阱进入 M 模式时，会写入 mcause 代码，指示导致陷阱的事件。否则，mcause 永远不会由实现编写，尽管它可能由软件显式编写。

如果陷阱是由中断引起的，则设置 mcause 寄存器中的中断位。异常代码字段包含标识最后异常或中断的代码。表 3.6 列出了可能的机器级异常代码。异常代码是一个 **WLRL** 字段，因此只能保证包含支持的异常代码。

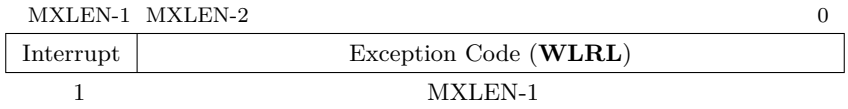


图 3.22: Machine Cause register mcause.

请注意，加载和保留加载指令会生成加载异常，而存储、条件存储和 AMO 指令会生成存储/AMO 异常。

可以通过 mcause 寄存器值的符号上的单个分支将中断与其他陷阱分开。左移可以移除中断位并缩放异常代码以索引到陷阱向量表中。

我们不区分特权指令异常和非法操作码异常。这简化了架构并隐藏了实现支持哪些更高权限指令的细节。为陷阱服务的特权级别可以实施关于是否需要区分这些的策略，如果需要，给定的操作码是否应该被视为非法或特权。

如果一条指令可能引发多个同步异常，则 Table 3.7 的优先级递减顺序指示在 mcause 中采取和报告哪个异常。任何自定义同步异常的优先级是实现定义的。

Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2	*Reserved*
1	3	Machine software interrupt
1	4	*Reserved*
1	5	Supervisor timer interrupt
1	6	*Reserved*
1	7	Machine timer interrupt
1	8	*Reserved*
1	9	Supervisor external interrupt
1	10	*Reserved*
1	11	Machine external interrupt
1	12–15	*Reserved*
1	≥16	*Designated for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	16–23	*Reserved*
0	24–31	*Designated for custom use*
0	32–47	*Reserved*
0	48–63	*Designated for custom use*
0	≥64	*Reserved*

表 3.6: Machine cause register (`mcause`) values after trap.

Priority	Exc. Code	Description
Highest	3	Instruction address breakpoint
		During instruction address translation:
	12, 1	First encountered page fault or access fault
		With physical address for instruction:
	1	Instruction access fault
	2	Illegal instruction
	0	Instruction address misaligned
	8, 9, 11	Environment call
	3	Environment break
	3	Load/store/AMO address breakpoint
		Optionally:
	4, 6	Load/store/AMO address misaligned
		During address translation for an explicit memory access:
	13, 15, 5, 7	First encountered page fault or access fault
		With physical address for an explicit memory access:
	5, 7	Load/store/AMO access fault
		If not higher priority:
Lowest	4, 6	Load/store/AMO address misaligned

表 3.7: Synchronous exception priority in decreasing priority order.

当一个虚拟地址被翻译成一个物理地址时，地址翻译算法决定了可能引发什么特定的异常。

加载/存储/AMO 地址未对齐异常的优先级可能高于或低于加载/存储/AMO 页面错误和访问错误异常。

加载/存储/*AMO* 地址未对齐和页面错误异常的相对优先级是实现定义的，以灵活地满足两个设计点。从不支持未对齐访问的实现可以无条件地引发未对齐地址异常，而无需执行地址转换或保护检查。仅支持对某些物理地址的未对齐访问的实现必须在确定未对齐访问是否可以继续之前转换并检查地址，在这种情况下引发页面错误异常或访问更合适。

指令地址断点与数据地址断点（也称为观察点）和环境中断异常（由 *EBREAK* 指令引发）具有相同的原因值，但优先级不同。

指令地址未对齐异常是由具有未对齐目标的控制流指令引发的，而不是通过获取指令的行为。因此，这些异常的优先级低于其他指令地址异常。

3.1.16 机器陷阱值寄存器 (mtval)

`mtval` 寄存器是一个 MXLEN 位的读写寄存器，格式如图 3.23。当陷阱进入 M 模式时，`mtval` 要么设置为零，要么写入特定于异常的信息，以帮助软件处理陷阱。否则，`mtval` 永远不会由实现编写，尽管它可能由软件显式编写。硬件平台将指定哪些异常必须设置 `mtval` 以提供信息，哪些可以无条件地将其设置为零。如果硬件平台指定没有异常将 `mtval` 设置为非零值，则 `mtval` 为只读零。

如果在指令获取、加载或存储时发生断点、地址未对齐、访问错误或页面错误异常时，使用非零值写入 `mtval`，则 `mtval` 将包含错误虚拟地址。

当启用基于页面的虚拟内存时，会使用错误的虚拟地址写入 `mtval`，即使对于物理内存访问错误异常也是如此。这种设计降低了大多数实现的数据路径成本，尤其是那些具有硬件页表遍历器的实现。

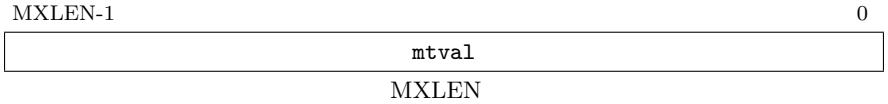


图 3.23: Machine Trap Value register.

如果在未对齐的加载或存储导致访问错误或页面错误异常时使用非零值写入 `mtval`，则 `mtval` 将包含导致错误的访问部分的虚拟地址。

如果在具有可变长度指令的系统上发生指令访问错误或页面错误异常时，mtval 以非零值写入，则-mtval 将包含导致错误的指令部分的虚拟地址，而 mepc 将指向指令的开头。

mtval 寄存器还可以可选地用于返回非法指令异常的错误指令位（mepc 指向内存中的错误指令）。如果 mtval 在发生非法指令异常时使用非零值写入，则 mtval 将包含以下最短值：

- 实际故障指令
- 故障指令的第一个 ILEN 位
- 故障指令的第一个 MXLEN 位

在非法指令异常上加载到 mtval 的值是右对齐的，所有未使用的上位都被清除为零。

捕获 mtval 中的错误指令减少了指令模拟的开销，在指令未对齐时可能避免多次部分指令加载，并且在使用加载将指令取到数据寄存器时可能会丢失数据缓存或减慢非缓存访问。如果另一个代理正在操作指令内存，也会出现原子性问题，就像在动态翻译系统中可能发生的那样。

一个要求是在捕获陷阱之前，将整个指令 *(*或者至少是第一个 *MXLEN* 位*)* 提取到 mtval 中。这不应该约束实现，实现通常会在尝试解码指令之前获取整个指令，并避免使软件处理程序复杂化。

mtval 中的值为零表示该特性不受支持，或者获取了非法的零指令。可以使用 mepc 指向的指令内存中的负载来区分这两种情况 *(*或者，可以查询系统配置信息，以便在运行时之前安装适当的 *trap* 处理*)*。

对于其他的陷阱，mtval 被设置为零，但是未来的标准可能会为其他的陷阱重新定义 mtval 的设置。

如果 mtval 不是只读零，它是一个 **WARL** 寄存器，必须能够保存所有有效的虚拟地址和值为零。它不需要能够保存所有可能的无效地址。在编写 mtval 之前，实现可能会将无效地址转换为 mtval 能够持有的其他一些无效地址。如果实现了返回错误指令位的特性，mtval 还必须能够保存所有小于 2^N 的值，其中 N 是 MXLEN 和 ILEN 中较小的一个。

3.1.17 机器配置指针计数器 (mconfigptr)

mconfigptr 是一个 mxlen 位只读 CSR，格式如图 3.24所示，它保存着配置数据结构的物理地址。软件可以遍历这个数据结构来发现关于 hart、平台及其配置的信息。

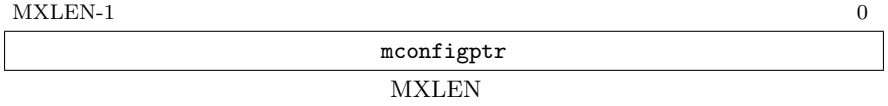


图 3.24: Machine Configuration Pointer register.

指针的位对齐必须不小于最大支持的 MXLEN: 例如，如果最大支持的 MXLEN 是 $8 * n$，那么 mconfigptr[log_n-1:0] 必须为零。

McOnfigptr 必须实现，但它可能为零，表示配置数据结构不存在，或者必须使用替代机制来定位它。

配置数据结构的格式和模式尚未标准化。

虽然在某些实现中 mconfigptr 将简单地硬连接,但其他实现可能提供一种方法来配置 *CSR* 读取时返回的值。例如，mconfigptr 可能表示由平台或 *m* 模式软件在引导过程开始时编写的内存映射寄存器的值。

3.1.18 机器环境配置寄存器 (menvcfg 和 menvcfgh)

menvcfg CSR 是一个 MXLEN 位读/写寄存器，格式化为 MXLEN=64，如图 3.25所示，它控制特权小于 M 模式的执行环境的某些特征。

63	62	61 — 8	7	6	5 — 4	3 — 1	0
STCE	PBMTE	**WPRI**	CBZE	CBCFE	CBIE	**WPRI**	FIOM
1	1	54	1	1	2	3	1

图 3.25: Machine environment configuration register (menvcfg) for MXLEN=64.

如果在 menvcfg 中将位 FIOM (I/O 的 Fence 意味着内存) 设置为 1，那么在特权小于 M 的模式下执行的 Fence 指令将被修改，因此对设备 I/O 的顺序访问要求也意味着对主存访问的顺序。表 3.8详细描述了 FIOM=1 时，对于特权小于 M 的模式，FENCE 指令位 PI、PO、SI 和 SO 的修改解释。

类似地，对于 FIOM=1 时特权小于 M 的模式，如果访问作为设备 I/O 排序的区域的原子指令有它的 *aq* 和/或 *rl* 位设置，那么该指令的排序就像它访问设备 I/O 和内存一样。

如果不支持 s 模式，或者如果 satp。MODE 为只读零 (总是 Bare)，实现可以使 FIOM 为只读零。

---

menvcfg 中需要 *FIOM* 位，因此 *m* 模式可以模拟 *Chapter* 八的 *hypervisor* 扩展，它在 *hypervisor CSR* henvcfg 中有一个等价的 *FIOM* 位。

PBMTE 位控制 Svpbmt 扩展是否可用于 s 模式和 g 级地址转换 (例如，对于由 satp 或 hgatp 指向的页表)。当 PBMTE=1 时，Svpbmt 可用于 s 模式和 g 级地址转换。当 PBMTE=0 时，实现的行为就像没有实现 Svpbmt 一样。如果没有实现 Svpbmt，则 PBMTE 为只读零。此外，对于使用管理程序扩展 henvcfg 的实现。menvcfg 时，PBMTE 只读为零。PBMTE 是零。

Instruction bit	Meaning when set
PI	Predecessor device input and memory reads (PR implied)
PO	Predecessor device output and memory writes (PW implied)
SI	Successor device input and memory reads (SR implied)
SO	Successor device output and memory writes (SW implied)

表 3.8: Modified interpretation of FENCE predecessor and successor sets for modes less privileged than M when FIOM=1.

STCE 字段的定义将由即将到来的 Sstc 扩展提供。它在 menvcfg 内的分配可能在批准该扩展之前发生变化。

CBZE 字段的定义将由即将到来的 Zicboz 扩展提供。它在 menvcfg 内的分配可能在批准该扩展之前发生变化。

CBCFE 和 CBIE 字段的定义将由即将到来的 Zicbom 扩展提供。它们在 menvcfg 内的分配可能在批准该扩展之前发生变化。

当 MXLEN=32 时，menvcfg 包含与 MXLEN=64 时 menvcfg 的 bits 31:0 相同的字段。此外，当 MXLEN=32 时，menvcfgh 是一个 32 位的读/写寄存器，它包含与 MXLEN=64 时 menvcfg 的 63:32 位相同的字段。当 MXLEN=64 时寄存器 menvcfgh 不存在。

如果不支持 u 模式，则寄存器 menvcfg 和 menvcfgh 不存在。

3.1.19 机器安全性配置寄存器 (mseccfg)

mseccfg 是一个可选的 MXLEN-bit 读/写寄存器，格式如图 3.26所示，它控制安全特性。

当 MXLEN=32 时，mseccfgh 是一个 32 位的读/写寄存器，当 MXLEN=64 时，它包含与 mseccfg 位 63:32 相同的字段。

XLEN-1	10	9	8	7	3	2	1	0
WPRI		SSEED	USEED	**WPRI**		RLB	MMWP	MML
XLEN-10		1	1	5		1	1	1

图 3.26: Machine security configuration register (mseccfg).

SSEED 和 USEED 字段的定义将由即将到来的熵源扩展 Zkr 提供。它们在 mseccfg 内的分配可能在批准该扩展之前发生变化。

RLB、MMWP 和 MML 字段的定义将由即将推出的 pmp 增强扩展 Smepmp 提供。它们在 mseccfg 内的分配可能在批准该扩展之前发生变化。

3.2 机器级内存映射寄存器

3.2.1 机器计时器寄存器 (mtime and mtimecmp)

平台提供一个实时计数器，作为内存映射的机器模式读写寄存器 mtime 公开。mtime 必须以恒定的频率递增，并且平台必须提供一种机制来确定 mtime 滴答的周期。如果计数溢出，mtime 寄存器将换行。

mtime 寄存器在所有 RV32 和 RV64 系统上都具有 64 位精度。平台提供 64 位内存映射的机器模式定时器比较寄存器 (mtimecmp)。当 mtime 包含大于或等于 mtimecmp 的值时，机器计时器中断将变为挂起状态，并将这些值视为无符号整数。在 mtimecmp 大于 mtime(通常是写入 mtimecmp 的结果) 之前，中断一直被发送。只有当中断被启用并且在 mie 寄存器中设置了 MTIE 位时，才会执行中断。

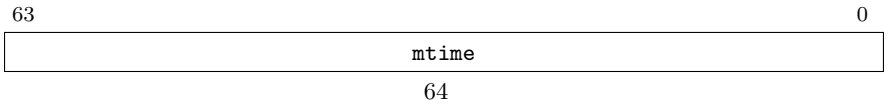


图 3.27: Machine time register (memory-mapped control register).

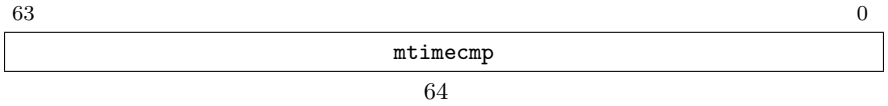


图 3.28: Machine time compare register (memory-mapped control register).

计时器设施被定义为使用挂钟时间而不是周期计数器来支持现代处理器，这些处理器具有高度可变的时钟频率，通过动态电压和频率缩放来节省能源。

提供精确的实时时钟 (rtc) 是相对昂贵的 (需要一个晶体或 MEMS 振荡器)，即使在系统的其余部分断电时也必须运行，因此在一个系统中通常只有一个位于与处理器不同的频率/电压域。因此，RTC 必须由系统中的所有 hart 共享，对 RTC 的访问可能会导致电压电平移位和时钟域交叉的损失。因此，作为内存映射寄存器公开 mtime 比作为 CSR 公开 mtime 更自然。

较低的特权级别没有自己的 timecmp 寄存器。相反，机器模式软件可以通过将下一个计时器中断多路复用到 mtimecmp 寄存器来实现任意数量的虚拟计时器。

简单的固定频率系统可以使用单个时钟进行周期计数和挂钟计时。

写入 mtime 和 mtimecmp 保证最终会反映到 MTIP 中，但不一定会立即反映出来。

如果中断处理程序递增 mtimecmp，然后立即返回，则可能会发生假计时器中断，因为 *MTIP* 可能尚未在这段时间内下降。所有的软件都应该编写为假设此事件是可能的，但大多数软件都应该假设此事件极不可能发生。与轮询 *MTIP* 直到它下降相比，偶尔引发虚假计时器中断几乎总是更具性能。

在 RV32 中，内存映射写入 mtimecmp 只修改寄存器的一个 32 位部分。以下代码序列设置 64 位 mtimecmp 值，而不会由于比较器的中间值而错误地生成计时器中断：

```
# New comparand is in a1:a0.
li t0, -1
la t1, mtimecmp
sw t0, 0(t1)     # No smaller than old value.
sw a1, 4(t1)     # No smaller than new value.
sw a0, 0(t1)     # New value.
```

图 3.29: Sample code for setting the 64-bit time comparand in RV32, assuming a little-endian memory system and that the registers live in a strongly ordered I/O region. Storing -1 to the low-order bits of mtimecmp prevents mtimecmp from temporarily becoming smaller than the lesser of the old and new values.

对于 RV64，还支持对 mtime 和-mtimecmp 寄存器的自然对齐 64 位内存访问，并且是原子的。

3.3 机器模式特权指令

3.3.1 环境调用和断点

31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
ECALL	0	PRIV	0	SYSTEM
EBREAK	0	PRIV	0	SYSTEM

ECALL 指令用于向支持执行环境发出请求。当在 U 模式、S 模式或 M 模式下执行时，它分别生成一个 environment-call-from-U-mode 异常、environmental-call-from-S-mode 例外或 environment-call-fom-mode 异常，并且不执行其他操作。

ECALL 为每个原始特权模式生成不同的异常，以便可以有选择地委托环境调用异常。类 *Unix* 操作系统的一个典型用例是将 *environment-call-from-U-mode* 异常委托给 *S-mode*，而不是其他异常。

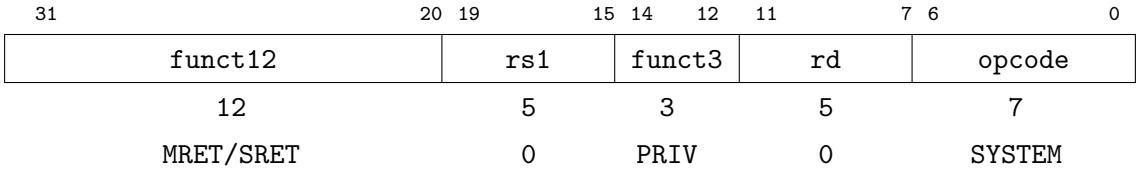
调试器使用 EBREAK 指令将控制权转移回调试环境。它生成断点异常并且不执行其他操作。

如本手册卷 *I* 中压缩指令的“*C*”标准扩展中所述，*C.EBREAK* 指令执行与 *EBREAK* 指令相同的操作。

ECALL 和 EBREAK 导致接收特权模式的 epc 寄存器被设置为 ECALL 或 EBREAK 指令本身的地址，*not* 是下一条指令的地址。由于 ECALL 和 EBREAK 导致同步异常，它们不被视为退休，并且不应增加 minstret CSR。

3.3.2 陷阱返回指令

从陷阱返回的指令在 PRIV 次要操作码下编码。



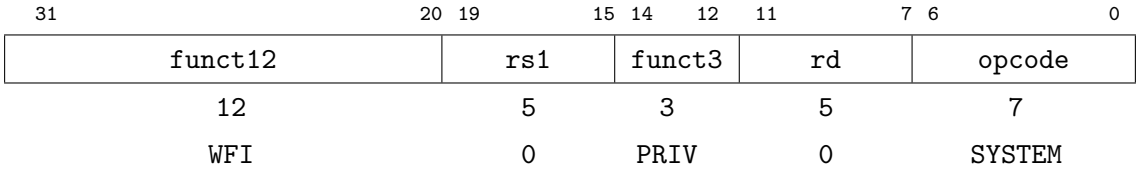
要在处理陷阱后返回，每个权限级别有单独的陷阱返回指令，MRET 和 SRET。始终提供 MRET。如果支持管理员模式，则必须提供 SRET，否则应引发非法指令异常。当 mstatus 中的 TSR=1 时，SRET 也应该引发非法指令异常，如 Section 3.1.6.5 中所述。*x*RET 指令可以在特权模式 *x* 或更高的模式下执行，在特权模式下执行较低的 *x*RET 指令将弹出相关的较低特权中断启用和特权模式堆栈。除了如第 3.1.6.1节中所述操作特权堆栈之外，*x*RET 还将 epc 设置为存储在 *x*epc 寄存器中的值。

如果支持 A 扩展，则允许 *x*RET 指令清除任何未完成的 LR 地址保留，但不是必需的。如果需要，陷阱处理程序应在执行 *x*RET 之前显式清除保留（例如，通过使用虚拟 SC）。

如果 x*RET* 指令总是清除 *LR* 保留，则不可能使用调试器单步执行 *LR/SC* 序列。

3.3.3 等待中断

等待中断指令(WFI)向实现提供了一个提示,即当前 hart 可以暂停,直到中断可能需要维护。WFI 指令的执行也可用于通知硬件平台,适当的中断应优先路由到此 hart。WFI 可用于所有特权模式,也可用于 U 模式。当 mstatus 中的 TW = 1 时,此指令可能引发非法指令异常,如第 ??节所述。



31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
WFI	0	PRIV	0	SYSTEM

If an enabled interrupt is present or later becomes present while the hart is stalled, the interrupt trap will be taken on the following instruction, i.e., execution resumes in the trap handler and mepc = pc + 4.

下面的指令接受中断陷阱,以便陷阱处理程序的简单返回将在 *WFI* 指令之后执行代码。

WFI 指令的目的是为实现提供提示,因此合法实现只是将 WFI 作为 NOP 实现。

如果实现在执行指令时没有暂停 *hart*,那么将对包含 *WFI* 的空闲循环中的某些指令执行中断,并且在处理程序的简单返回时,空闲循环将恢复执行。

当中断被禁用时,也可以执行 WFI 指令。WFI 的操作必须不受全局中断位(MIE 和 SIE)和委托寄存器 mideleg 的影响(即,如果本地启用的中断挂起,即使它被委托到较低特权模式,hart 也必须恢复),但应遵守单独的中断启用(例如,MTIE)(即,如果中断挂起但未单独启用,则实现应避免恢复 hart)。WFI 还需要恢复在任何特权级别上挂起的本地启用中断的执行,而不管在每个特权级别上启用了全局中断。

如果导致 hart 恢复执行的事件未导致中断,则执行将在 pc+4 时恢复,并且软件必须确定要执行的操作,包括在没有可操作事件的情况下循环返回以重复 WFI。

通过在中断被禁用时允许唤醒,可以调用不需要保存当前上下文的中断处理程序的备用入口点,因为在执行 *WFI* 之前可以保存或丢弃当前上下文。

由于实现可以自由地将 *WFI* 实现为 *NOP*,软件必须明确检查 *WFI* 之后的代码中是否存在任何相关的挂起但已禁用的中断,如果没有检测到合适的中断,则应循环回 *WFI*。可以询问 mip 或 sip 寄存器,以分别确定机器或监控程序模式下是否存在任何中断。

WFI 的操作不受委派寄存器设置的影响。

定义 *WFI* 是为了使一个实现可以在遇到 *WFI* 时立即或在某个时间间隔后，启动机器模式转换到较低功率状态，从而陷入较高特权模式。

相同的“等待事件”模板可能用于等待内存位置更改或消息到达的未来扩展。

3.3.4 自定义系统指令

图 3.30中所示的 SYSTEM 主操作码的子空间被指定用于自定义用途。建议这些指令与其他 SYSTEM 指令一样，使用位 29:28 指定所需的最低特权模式。

31 26	25 15	14 12	11 7	6 0	
funct6	*custom*	funct3	*custom*	opcode	Recommended Purpose
6	11	3	5	7	
100011	*custom*	0	*custom*	SYSTEM	Unprivileged or User-Level
110011	*custom*	0	*custom*	SYSTEM	Unprivileged or User-Level
100111	*custom*	0	*custom*	SYSTEM	Supervisor-Level
110111	*custom*	0	*custom*	SYSTEM	Supervisor-Level
101011	*custom*	0	*custom*	SYSTEM	Hypervisor-Level
111011	*custom*	0	*custom*	SYSTEM	Hypervisor-Level
101111	*custom*	0	*custom*	SYSTEM	Machine-Level
111111	*custom*	0	*custom*	SYSTEM	Machine-Level

图 3.30: SYSTEM instruction encodings designated for custom use.

3.4 重置

重置后，哈特的特权模式被设置为 M。mstatus 字段 MIE 和 MPRV 被重置为 0。如果支持小端内存访问，mstatus/mstatush 字段 MBE 将重置为 0。misa 寄存器被重置，以启用最大支持扩展集和最宽的 MXLEN，如第 3.1.1节所述。对于使用“A”标准扩展名的实现，没有有效的加载保留。pc 被设置为实现定义的重置向量。mcause 寄存器被设置为指示复位原因的值。可写 PMP 寄存器的 A 和 L 字段设置为 0，除非平台要求某些 PMP 寄存器 A 和 L 的字段具有不同的重置值。如果实现了虚拟机监控程序扩展。MODE 和 vsatp。MODE 字段重置为 0。没有 WARL 字段包含非法值。所有其他哈特状态都未指定。

重置后的 mcause 值具有特定于实现的解释，但对于不区分不同重置条件的实现，应返回值 0。区分不同重置条件的实现应仅使用 0 表示最完整的重置。

某些设计可能有多种重置原因（例如，开机重置、外部硬重置、检测到断电、看门狗定时器过期、睡眠模式唤醒），机器模式软件和调试器可能希望区分这些原因。

mcause 重置值可能会在同步异常之后别名 mcause 值。此重叠中不应有歧义，因为重置时，pc 通常设置为与其他陷阱不同的值。

3.5 不可屏蔽中断

不可屏蔽中断（NMI）仅用于硬件错误情况，无论 hart 的中断启用位的状态如何，都会立即跳转到以 M 模式运行的实现定义的 NMI 向量。用中断的指令的虚拟地址写入 mepc 寄存器，并将 mcause 设置为指示 NMI 源的值。因此，NMI 可以覆盖活动机器模式中断处理程序中的状态。

在 NMI 上写入 mcause 的值是实现定义的。mcause 的高中断位应设置为指示这是一个中断。保留 0 的异常代码表示“未知原因”，并且不通过 mcause 寄存器区分 NMI 源的实现应在异常代码中返回 0。

与重置不同，NMI 不会重置处理器状态，从而实现诊断、报告和可能的硬件错误遏制。

3.6 物理内存属性

完整系统的物理内存映射包括各种地址范围，一些对应于内存区域，一些对应内存映射控制寄存器，一些对应地址空间中的空穴。某些内存区域可能不支持读、写或执行；有些可能不支持子字或子块访问；有些可能不支持原子操作；并且一些可能不支持高速缓存一致性或者可能具有不同的存储器模型。类似地，内存映射控制寄存器在其支持的访问宽度、对原子操作的支持以及读写访问是否具有相关的副作用方面也有所不同。在 RISC-V 系统中，机器物理地址空间的每个区域的这些属性和能力称为物理内存属性（PMA）。本节介绍 RISC-V PMA 术语以及 RISC-V 系统如何实施和检查 PMA。

PMA 是底层硬件的固有属性，在系统运行期间很少更改。与 3.7节中描述的物理内存保护值不同，PMA 不会因执行上下文而变化。一些存储区域的 PMA 在芯片设计时是固定的，例如，对于片上 ROM。其他存储区域在板上设计时固定，例如，取决于其他芯片连接到芯片外总线。芯片外总线还可能支持在每次电源周期（可冷插拔）或在系统运行时动态更改的设备（可热插拔）。某些设备可能在运行时可配置，以支持不同的使用，这意味着不同的 PMA——例如，片上草稿行 RAM 可能由一个终端应用程序中的一个内核单独缓存，或作为另一个终端程序中的共享非缓存内存进行访问。

大多数系统将要求在知道物理地址之后，在执行管道中的硬件中动态检查至少一些 PMA，因为某些操作在所有物理内存地址上都不受支持，有些操作需要知道可配置 PMA 属性的当前设置。虽然许多其他架构在虚拟内存页表中指定了一些 PMA，并使用 TLB 通知管道这些属性，但这种方法将特定于平台的信息注入虚拟化层，并可能导致系统错误，除非在每个物理内存区域的每个页表条目中正确初始化了属性。此外，可用的页面大小可能不适合在物理内存空间中指定属性，从而导致地址空间碎片化和昂贵 TLB 条目的使用效率低下。

对于 RISC-V，我们将 PMA 的规范和检查分离到一个单独的硬件结构中，即 ***PMA checker***。在许多情况下，每个物理地址区域的属性在系统设计时都是已知的，并且可以硬连接到 PMA 检查器中。如果属性是运行时可配置的，则可以提供特定于平台的内存映射控制寄存器，以便以适合平台上每个区域的粒度指定这些属性（例如，对于可在可缓存和不可缓存用途之间灵活划分的片上 SRAM）。检查 PMA 是否有任何对物理内存的访问，包括经过虚拟到物理内存转换的访问。为了帮助系统调试，我们强烈建议，在可能的情况下，RISC-V 处理器精确捕获无法通过 PMA 检查的物理内存访问。精确捕获的 PMA 违规表现为指令、加载或存储访问错误异常，与虚拟内存页错误异常不同。精确的 PMA 陷阱可能并不总是可能的，例如，当探测使用访问失败作为发现机制一部分的遗留总线架构时。在这种情况下，从设备的错误响应将被报告为不精确的总线错误中断。

PMA 还必须是软件可读的，以正确访问某些设备或正确配置访问内存的其他硬件组件，如 DMA 引擎。由于 PMA 与给定物理平台的组织紧密相连，许多细节本质上是特定于平台的，软件可以通过这些方法学习平台的 PMA 值。某些设备（尤其是传统总线）不支持 PMA 的发现，因此如果尝试不受支持的访问，将给出错误响应或超时。通常，特定于平台的机器模式代码将提取 PMA，并最终使用一些标准表示将此信息呈现给较高级的特权较低的软件。

如果平台支持 PMA 的动态重新配置，将提供一个接口，通过将请求传递给能够正确重新配置平台的机器模式驱动程序来设置属性。例如，在某些内存区域上切换可缓存性属性可能涉及特定于平台的操作，例如仅对机器模式可用的缓存刷新。

3.6.1 主存、I/O 或空闲区

给定内存地址范围的最重要特征是它是否持有常规主内存、I/O 设备或空闲。常规主内存需要具有以下指定的许多属性，而 I/O 设备可以具有更广泛的属性范围。不适合常规主存储器的存储器区域，例如，设备草稿 RAM，被归类为 I/O 区域。空闲区域也被分类为 I/O 区域，但其属性指定不支持访问。

3.6.2 支持访问类型 PMAs

访问类型指定支持哪些访问宽度，从 8 位字节到长多字突发，以及每个访问宽度是否支持未对齐的访问。

尽管在 *RISC-Vhart* 上运行的软件无法直接生成内存突发，但软件可能必须对 *DMA* 引擎进行编程以访问 *I/O* 设备，因此可能需要知道支持哪些访问大小。

主内存区域始终支持所连接设备所需的所有访问宽度的读写，并可以指定是否支持指令获取。

一些平台可能会要求所有主内存都支持指令获取。其他平台可能禁止从某些主内存区域获取指令。

在某些情况下，访问主内存的处理器或设备的设计可能支持其他宽度，但必须能够与主内存支持的类型一起工作。

I/O 区域可以指定支持哪些数据宽度的读、写或执行访问组合。

对于具有基于页的虚拟内存的系统，I/O 和内存区域可以指定支持硬件页表读取和硬件页表写入的组合。

类 *Unix* 操作系统通常要求所有可缓存主内存都支持页表遍历。

3.6.3 原子性 PMAs

原子性 PMA 描述此地址区域中支持哪些原子指令。对原子指令的支持分为两类：*LR/SC* 和 *AMO*。

一些平台可能会要求所有可缓存主内存支持所连接处理器所需的所有原子操作。

3.6.3.1 AMO PMA

在 AMO 中，有四个级别的支持：*AMONone*，*AMOSwap*，*AMOLogical*，and *AMOArithmetic*。AMONone 表示不支持任何 AMO 操作。AMOSwap 表示此地址范围内仅支持 *AMOSwap* 指令。AMOLogical 表示支持交换指令加上所有逻辑 AMO(amoand, amoor, amoxor)。AMOArithmetic 表示支持所有 RISC-V AMO。对于每个支持级别，如果底层内存区域支持给定宽度的读写，则支持该宽度的自然对齐 AMO。主内存和 I/O 区域可能只支持处理器支持的原子操作的一个子集或不支持。

AMO Class	Supported Operations
AMONone	*None*
AMOSwap	`amoswap`
AMOLogical	above + `amoand`, `amoor`, `amoxor`
AMOArithmetic	above + `amoadd`, `amomin`, `amomax`, `amominu`, `amomaxu`

表 3.9: Classes of AMOs supported by I/O regions.

我们建议尽可能为 *I/O* 区域提供至少 *AMOLogical* 支持。

3.6.3.2 Reservability PMA

对于 *LR/SC*,有三个级别的支持表示可保留性和偶然性属性的组合:*RsrvNone*, *RsrvNonEventual*, and *RsrvEventual*。RsrvNone 表示不支持 LR/SC 操作(该位置不可保留)。RsrvNonEventual 表示支持操作(位置是可保留的),但没有特权 ISA 规范中描述的最终成功保证。RsrvEventual 表示支持这些操作,并提供最终的成功保证。

我们建议尽可能为主内存区域提供 *RsrvEventual* 支持。大多数 *I/O* 区域将不支持 *LR/SC* 访问,因为它们最方便地构建在缓存一致性方案之上,但有些可能支持 *RsrvNonEventual* 或 *RsrvEventual*。

当 *LR/SC* 用于标记为 *RsrvNonEventual* 的内存位置时,软件应提供在检测到进度不足时使用的备用回退机制。

3.6.3.3 Alignment

对于某些地址和访问宽度,支持对齐 LR/SC 或对齐 AMO 的内存区域也可能支持未对齐 LR/S 或未对齐 AMO。如果对于给定地址和访问宽度,未对齐的 LR/SC 或 AMO 生成地址未对齐异常,则所有使用该地址和访问带宽的加载、存储、LR/SC 和 AMO 必须生成地址未对准异常。

标准“*A*”扩展不支持未对齐的 *AMO* 或 *LR/SC* 对。标准“*Zam*”扩展提供了对未对齐 *AMO* 的支持。对未对齐 *LR/SC* 序列的支持目前还没有标准化,因此 *LR* 和 *SC* 对未对齐地址的支持必须引发异常。

强制未对齐的加载和存储在未对齐的 *AMO* 引发地址未对齐异常的地方引发地址未对准异常,这允许在 *M* 模式陷阱处理程序中模拟未对齐的 *AM*。处理程序通过获取全局互斥体并模拟

关键部分中的访问来保证原子性。如果未对齐加载和存储的处理程序使用相同的互斥体，则对使用相同字长的给定地址的所有访问都将是相互原子的。

对于某些未对齐的访问，实现可能会引发访问错误异常，而不是地址未对齐异常，这表明陷阱处理程序不应模拟该指令。如果对于给定的地址和访问宽度，所有未对齐的 LR/SC 和 AMO 都会生成访问故障异常，则使用相同地址和访问带宽的常规未对齐加载和存储不需要自动执行。

3.6.4 内存序 PMAs

为了通过 FENCE 指令和原子指令排序位进行排序，地址空间的区域被分类为 *main memory* 或 *I/O*。

一个 hart 对主存储器区域的访问不仅可由其他 hart 观察到，而且可由具有在主存储器系统中发起请求的能力的其他设备（例如 DMA 引擎）观察到。相干主存储器区域总是具有 RVWMO 或 RVTSO 存储器模型。不相干的主存储器区域具有实现定义的存储器模型。

一个 hart 对 I/O 区域的访问不仅可以被其他 hart 和总线主控设备观察到，也可以被目标从 I/O 设备观察到。I/O 区域可以通过 *relaxed* or *strong* 顺序进行访问。对具有宽松顺序的 I/O 区域的访问通常由其他 hart 和总线主控设备以类似于对 RVWMO 存储器区域的访问顺序的方式观察，如本规范第一卷第 a.4.2 节所述。相比之下，对具有强顺序的 I/O 区域的访问通常由其他 hart 和总线主控设备按程序顺序观察。

每个强排序 I/O 区域都指定一个编号的排序通道，这是一种机制，可以在不同的 I/O 区域之间提供排序保证。通道 0 仅用于指示点对点强排序，其中只有 hart 对单个关联 I/O 区域的访问是强排序的。

通道 1 用于跨所有 I/O 区域提供全局强排序。哈特对与通道 1 关联的任何 I/O 区域的任何访问只能被所有其他哈特和 I/O 设备按程序顺序观察到，包括相对于哈特对具有不同通道号的宽松 I/O 区域或强有序 I/O 区域的访问。换言之，对通道 1 中的区域的任何访问都等同于在指令之前和之后执行 fence io,io 指令。

其他较大的频道号为具有相同频道号的任何区域上的哈特访问提供节目顺序。

系统可能支持对每个内存区域的排序属性进行动态配置。

强排序可用于提高与传统设备驱动程序代码的兼容性，或在已知实现不重新排序访问时，与插入显式排序指令相比，提高性能。

本地强排序（通道 *0*）是强排序的默认形式，因为如果 *hart* 和 *I/O* 设备之间只有一条有序的通信路径，通常很容易提供。

通常，如果不同的强排序 *I/O* 区域共享相同的互连路径且路径不重新排序请求，则它们可以共享相同的排序通道，而无需额外的排序硬件。

3.6.5 一致性与可缓存性 PMAs

一致性是为单个物理地址定义的属性，表示一个代理对该地址的写入最终将对系统中的其他代理可见。一致性不应与系统的内存一致性模型混淆，该模型定义了在给定整个内存系统的读写历史的情况下，内存读取可以返回的值。在 RISC-V 平台中，由于软件复杂性、性能和能量影响，不鼓励使用硬件不一致区域。

内存区域的可缓存性不应影响该区域的软件视图，但其他 PMA 中反映的差异除外，如主内存与 I/O 分类、内存排序、支持的访问和原子操作以及一致性。因此，我们将可缓存性视为仅由机器模式软件管理的平台级设置。

如果平台支持内存区域的可配置缓存能力设置，则特定于平台的机器模式例程将更改设置并在必要时刷新缓存，因此系统仅在缓存能力设置之间的转换期间不一致。较低权限级别不应看到此临时状态。

我们将 *RISC-V* 缓存分为三种类型：master-private, shared, *and* slave-private。主专用缓存连接到单个主代理，即向内存系统发出读/写请求的代理。共享缓存位于主缓存和从缓存之间，可以分层组织。从机专用缓存不会影响一致性，因为它们是单个从机的本地缓存，不会影响主机上的其他 *PMA*，因此这里不再进一步讨论。除非另有明确说明，否则在下一节中，我们使用 private cache 表示主专用缓存。

一致性可以直接提供不被任何代理缓存的共享内存区域。这样一个区域的 *PMA* 将简单地指示它不应该缓存在私有或共享缓存中。

对于只读区域，一致性也很简单，可以由多个代理安全地缓存，而不需要缓存一致性方案。此区域的 *PMA* 将指示它可以缓存，但不支持写入。

某些读写区域可能只能由单个代理访问，在这种情况下，它们可以由该代理单独缓存，而不需要一致性方案。这些区域的 *PMA* 将指示它们可以被缓存。数据也可以缓存在共享缓存中，因为其他代理不应该访问该区域。

如果代理可以缓存其他代理（无论是缓存还是非缓存）可以访问的读写区域，则需要缓存一致性方案以避免使用过时值。在缺乏硬件缓存一致性的区域（硬件不一致区域）中，缓存一致性可以完全在软件中实现，但软件一致性方案很难正确实现，并且由于需要保守的软件定向缓存刷新，通常会对性能产生严重影响。硬件缓存一致性方案需要更复杂的硬件，并且由于缓存一致性探测可能会影响性能，但在其他方面对软件是不可见的。

对于每个硬件缓存相干区域，*PMA* 将指示该区域是相干的，以及如果系统具有多个相干控制器，则使用哪个硬件相干控制器。对于某些系统，一致性控制器可能是一个外部级共享缓存，它本身可能会分层访问更多的外部级缓存一致性控制器。

平台内的大多数内存区域将与软件保持一致，因为它们将被固定为未缓存、只读、硬件缓存一致或仅由一个代理访问。

如果 PMA 指示不可缓存，那么对该区域的访问必须由内存本身来满足，而不是由任何缓存来满足。

对于具有可缓存性控制机制的实现，可能会出现程序无法缓存地访问当前驻留的内存位置的情况。在这种情况下，必须忽略缓存副本。此约束对于防止特权模式的推测性缓存重新填充影响特权模式的不可缓存访问行为是必要的。

3.6.6 冥等性 PMAs

幂等 PMA 描述对地址区域的读和写是否为幂等。假设主存储器区域是幂等的。对于 I/O 区域，读取和写入的幂等性可以单独指定（例如，读取是幂等的，但写入不是）。如果访问是非幂等的，即对任何读或写访问都有潜在的副作用，那么必须避免推测性或冗余访问。

为了定义幂等性 PMA，由冗余访问创建的观察到的内存顺序变化不被视为副作用。

虽然硬件的设计应始终避免对标记为非幂等的内存区域进行推测性或冗余访问，但也有必要确保软件或编译器优化不会产生对非幂等内存区域的虚假访问。

非幂等区域可能不支持未对齐的访问。对这些区域的未对齐访问应引发访问错误异常，而不是解决未对齐异常，这表明软件不应使用多个较小的访问来模拟未对齐的访问，这可能会导致意外的副作用。

对于非幂等区域，隐式读取和写入不能过早或推测性地执行，以下例外情况除外。当执行非推测性隐式读取时，允许实现额外读取包含非推测性隐含读取地址的自然对齐的 2 次幂区域内的任何字节。此外，当执行非推测性指令获取时，允许实现额外读取相同大小的 *next* 自然对齐的 power-of-2 区域内的任何字节（该区域的地址取模 2^{XLEN}）。这些额外读取的结果可用于满足后续的早期或推测性隐式读取。这些自然对齐的 power-of-2 区域的大小由实现定义，但对于具有基于页面的虚拟内存的系统，其大小不得超过支持的最小页面大小。

3.7 物理内存保护

为了支持安全处理并包含错误，最好限制运行在 hart 上的软件可访问的物理地址。可选的物理内存保护（PMP）单元提供每哈特机器模式控制寄存器，以允许为每个物理内存区域指定物理内存访问权限（读、写、执行）。PMP 值与第 **??**节中描述的 PMA 检查并行检查。

PMP 访问控制设置的粒度是特定于平台的，但标准 PMP 编码支持小到四个字节的区域。某些区域的权限可以是硬连接的——例如，某些区域可能只在机器模式下可见，但在较低的权限层中不可见。

平台对物理内存保护的需求差异很大，一些平台可能提供其他 *PMP* 结构，以补充或替代本节中描述的方案。

PMP 检查适用于有效特权模式为 S 或 U 的所有访问，包括在 S 和 U 模式下的指令获取，当寄存器中的 MPRV 位为空时在 S 和 U 模式下的数据访问，以及当设置了 mstatus 中的 MPRV 位且 mstatus 中的 MPP 字段包含 S 或 U 时在任何模式下进行的数据访问。PMP 检查也适用于虚拟地址转换的页表访问，有效特权模式为 S。可选地，PMP 检查还可适用于 M 模式访问，在这种情况下，PMP 寄存器本身被锁定，因此即使是 M 模式软件也无法更改它们，直到 hart 被重置。实际上，PMP 可以授予 S 和 U 模式的权限（默认情况下没有），并且可以撤销 M 模式的权限，默认情况下 M 模式具有完全权限。

PMP 违规总是在处理器处被精确捕获。

3.7.1 物理内存保护 CSRs

PMP 条目由 8 位配置寄存器和一个 MXLEN 位地址寄存器描述。一些 PMP 设置还使用与前面的 PMP 条目相关联的地址寄存器。最多支持 64 个 PMP 条目。实现可以实现零、16 或 64 个 PMP CSR；必须首先实现最低编号的 PMP CSR。所有 PMP CSR 字段均为 **WARL** ，并且可能为只读零。PMP CSR 仅可用于 M 模式。

PMP 配置寄存器被密集地封装到 CSR 中，以最小化上下文切换时间。对于 RV32，十六个 CSR，pmpcfg0--pmpcfg15，保持 64 个 PMP 条目的配置 pmp0cfg--pmp63cfg，如图 3.31所示。对于 RV64，八个偶数编号的 CSR，pmpcfg0，pmpcfg2、…、pmpcfg14，保存 64 个 PMP 条目的配置，如图 3.32所示。对于 RV64，奇数编号的配置寄存器 pmpcfg1，pmpcfg3、…、pmpcfg15 是非法的。

RV64 系统使用 pmpcfg2 而不是 pmpcfg1 来保存 *PMP* 条目 *8-15* 的配置。这种设计降低了支持多个 *MXLEN* 值的成本，因为对于 *RV32* 和 *RV64*，*PMP* 条目 *8-11* 的配置都出现在 pmpcfg2*[31:0]* 中。

PMP 地址寄存器是名为 pmpaddr0--pmpaddr63 的 CSR。每个 PMP 地址寄存器对 RV32 的 34 位物理地址的位 33-2 进行编码，如图 3.33所示。对于 RV64，每个 PMP 地址寄存器对 56 位物理地址的 55-2 位进行编码，如图 3.34所示。并非所有物理地址位都可以实现，因此 pmpaddr 寄存器都是 **WARL** 。

第 *4.3*节中描述的基于 *Sv32* 页的虚拟内存方案支持 *RV32* 的 *34* 位物理地址，因此 *PMP* 方案必须支持比 *RV32* 的 *XLEN* 宽的地址。第 *4.4*节和第 **??**节中描述的基于 *Sv39* 和 *Sv48* 页的虚拟内存方案支持 *56* 位物理地址空间，因此 *RV64 PMP* 地址寄存器施加了相同的限制。

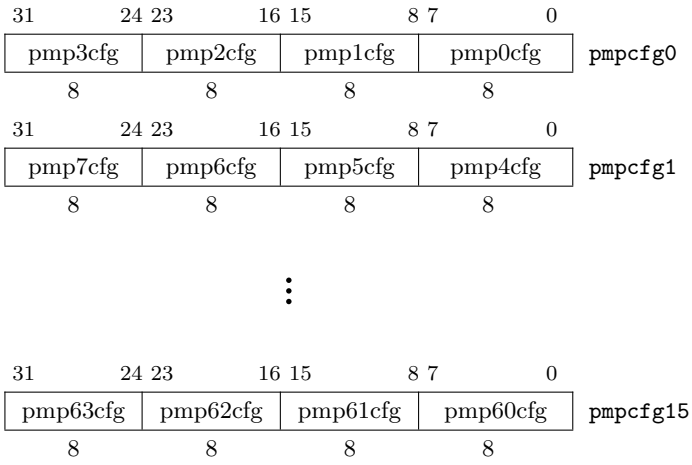


图 3.31: RV32 PMP configuration CSR layout.

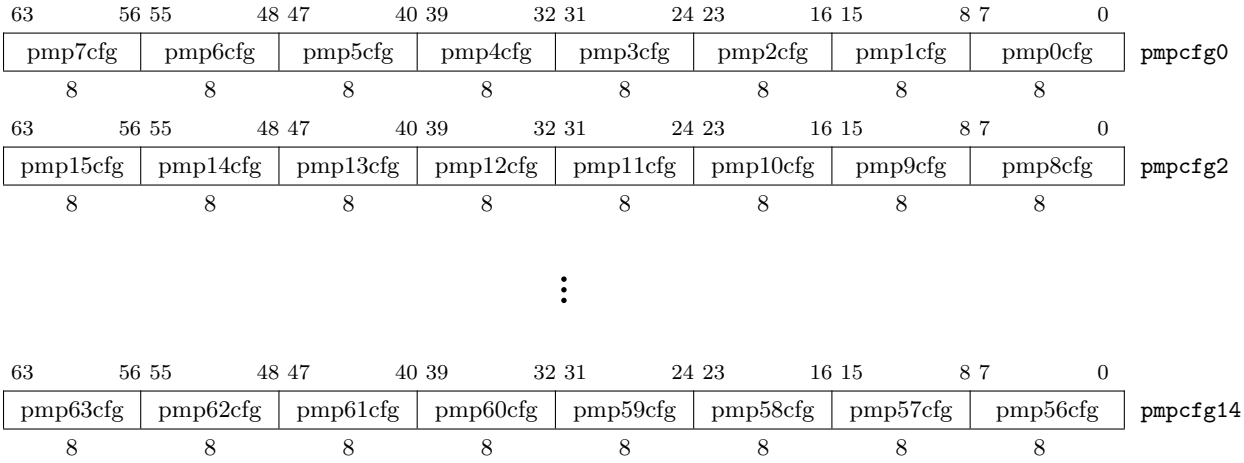


图 3.32: RV64 PMP configuration CSR layout.

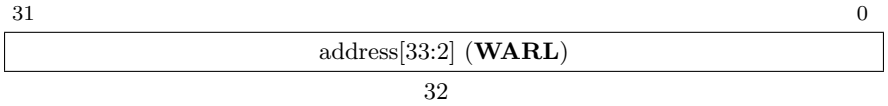


图 3.33: PMP address register format, RV32.

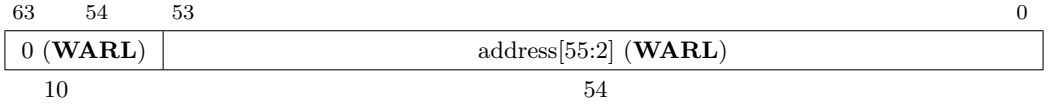


图 3.34: PMP address register format, RV64.

图 3.35显示了 PMP 配置寄存器的布局。设置后，R、W 和 X 位分别表示 PMP 条目允许读、写和指令执行。当其中一个位被清除时，相应的访问类型被拒绝。R、W 和 X 字段形成一个集合 WARL 字段，其中保留了 R=0 和 W=1 的组合。剩下的两个字段 A 和 L 将在下面的部分中介绍。

7	6 5	4 3	2	1	0
L (**WARL**)	0 (**WARL**)	A (**WARL**)	X (**WARL**)	W (**WARL**)	R (**WARL**)
1	2	2	1	1	1

图 3.35: PMP configuration register format.

尝试从没有执行权限的 PMP 区域获取指令会引发指令访问错误异常。尝试执行加载或加载保留指令，该指令在没有读取权限的情况下访问 PMP 区域内的物理地址，会引发加载访问错误异常。尝试执行存储、存储条件或 AMO 指令，该指令在没有写入权限的情况下访问 PMP 区域内的物理地址，会引发存储访问错误异常。

如果更改了 MXLEN，pmp*x*cfg 字段的内容将保留，但会出现在由 MXLEN 的新设置指定的 pmpcfg*y*CSR 中。例如，当 MXLEN 从 6 4 更改为 32 时，pmp4cfg 从 pmpcfg0[39:32] 移动到 pmpcfg1[7:0]。pmpaddr CSRs follow the usual CSR width modulation rules described in Section 2.6. 节中描述的常见 CSR 宽度调制规则。

Address Matching

PMP 条目的配置寄存器中的 A 字段对相关 PMP 地址寄存器的地址匹配模式进行编码。该字段的编码如表 3.10所示。当 A = 0 时，此 PMP 条目被禁用，并且不匹配任何地址。支持其他两种地址匹配模式：自然对齐的 2 区断电（NAPOT），包括自然对齐的 4 字节区的特殊情况（NA4）；以及任意范围（TOR）的顶部边界。这些模式支持四字节粒度。

A	Name	Description
0	OFF	Null region (disabled)
1	TOR	Top of range
2	NA4	Naturally aligned four-byte region
3	NAPOT	Naturally aligned power-of-two region, ≥8 bytes

表 3.10: Encoding of A field in PMP configuration registers.

NAPOT ranges make use of the low-order bits of the associated address register to encode the size of the range, as shown in Table 3.11.

如果选择了 TOR，则相关的地址寄存器构成地址范围的顶部，而前面的 PMP 地址寄存器构成了地址范围的底部。如果 PMP 条目 i 的 A 字段设置为 TOR，则该条目与任何地址 y 匹配，使得 $\texttt{pmpaddr}_{i-1} \leq y < \texttt{pmpaddr}_i$（与 $\texttt{pmpcfg}_{i-1}$ 的值无关）。如果 PMP 条目 0 的 A 字段设置为 TOR，则下限使用零，因此它与任何地址 $y < \texttt{pmpaddr}_0$。

pmpaddr	pmpcfg.A	Match type and size
yyyy...yyyy	NA4	4-byte NAPOT range
yyyy...yyy0	NAPOT	8-byte NAPOT range
yyyy...yy01	NAPOT	16-byte NAPOT range
yyyy...y011	NAPOT	32-byte NAPOT range
...	...	...
yy01...1111	NAPOT	2^{XLEN}-byte NAPOT range
y011...1111	NAPOT	$2^{\text{XLEN}+1}$-byte NAPOT range
0111...1111	NAPOT	$2^{\text{XLEN}+2}$-byte NAPOT range
1111...1111	NAPOT	$2^{\text{XLEN}+3}$-byte NAPOT range

表 3.11: NAPOT range encoding in PMP address and configuration registers.

如果 $\text{pmpaddr}_{i-1} \geq \text{pmpaddr}_i$ 和 pmpcfg_i.A=*TOR*，则 *PMP* 条目 i 不匹配任何地址。

虽然 PMP 机制支持小到四个字节的区域，但平台可以指定更粗的 PMP 区域。通常，PMP 粒度为 2^{G+2} 字节，并且在所有 PMP 区域中必须相同。当 $G \geq 1$ 时，NA4 模式不可选择。当 $G \geq 2$ 和 $-\text{pmpcfg}_i$ 时。设置了 [1]，即模式为 NAPOT，然后将位 pmpaddr_i[G-2:0] 读取为所有 1。当 $G \geq 1$ 和 $-\text{pmpcfg}_i$ 时。A[1] 为空，即模式为 OFF 或 TOR，则位 pmpaddr_i[G-1:0] 读取为全零。位 pmpaddr_i[G-1:0] 不影响 TOR 地址匹配逻辑。尽管正在更改 $-\text{pmpcfg}_i$。[1] 影响从 pmpaddr_i 读取的值，它不影响存储在该寄存器中的基础值——特别是，当 $ttpmpcfg_i$ 时，pmpaddr_i[G-1] 保留其原始值。A 从 NAPOT 变为 TOR/OFF，然后变回 NAPOT。

软件可以通过将 *0* 写入-pmp0cfg，然后将所有 *1* 写入 pmpaddr0，再读回 pmpaddr 0 来确定 *PMP* 粒度。如果 G 是最低有效位集的索引，则 *PMP* 粒度为 2^{G+2} 字节。

如果当前的 XLEN 大于 MXLEN，则为了地址匹配，PMP 地址寄存器将从 MXLEN 位零扩展到 XLEN 位。

Locking and Privilege Mode

L 位表示 PMP 条目被锁定，即对配置寄存器和相关地址寄存器的写入被忽略。锁定的 PMP 条目将保持锁定，直到图表重置。如果 PMP 条目 i 被锁定，将忽略对-PMP$i$cfg 和 pmpaddr$i$ 的写入。此外，如果 PMP 条目 i 被锁定并且-PMP$i$cfg。A 设置为 TOR，写入 pmpaddr$i$-1 将被忽略。

Setting the L bit locks the PMP entry even when the A field is set to OFF.

除了锁定 PMP 条目外，L 位还指示是否对 M 模式访问强制 R/W/X 权限。当设置了 L 位时，将对所有特权模式强制执行这些权限。当 L 位清除时，任何与 PMP 条目匹配的 M 模式访问都将成功；R/W/X 权限仅适用于 S 和 U 模式。

Priority and Matching Logic

PMP 条目按静态优先级排序。与访问的任何字节相匹配的编号最低的 PMP 条目确定该访问是成功还是失败。匹配的 PMP 条目必须匹配访问的所有字节，否则无论 L、R、W 和 X 位如何，访问都会失败。例如，如果 PMP 条目被配置为匹配四字节范围 0xC--0xF，则假设 PMP 条目是匹配这些地址的最高优先级条目，则对范围 0x8--的 8 字节访问将失败。

如果 PMP 条目与访问的所有字节相匹配，则 L、R、W 和 X 位确定访问是成功还是失败。如果 L 位为空且访问的特权模式为 M，则访问成功。否则，如果设置了 L 位或访问的特权模式为 S 或 U，则仅当设置了与访问类型对应的 R、W 或 X 位时，访问才成功。

如果没有 PMP 条目匹配 M 模式访问，则访问成功。如果没有 PMP 条目匹配 S 模式或 U 模式访问，但至少实现了一个 PMP 条目，则访问失败。

If at least one PMP entry is implemented, but all PMP entries' A fields are set to OFF, then all S-mode and U-mode memory accesses will fail.

失败的访问生成指令、加载或存储访问错误异常。请注意，单个指令可能生成多个访问，这些访问可能不是相互原子的。如果指令生成的至少一个访问失败，则会生成访问错误异常，尽管该指令生成的其他访问可能会成功，但会产生明显的副作用。值得注意的是，引用虚拟内存的指令被分解为多次访问。

在某些实现中，未对齐的加载、存储和指令获取也可能分解为多个访问，其中一些访问可能在发生访问错误异常之前成功。特别是，通过 PMP 检查的未对齐存储的一部分可能会变得可见，即使另一部分未通过 PMP 检测。对于宽于 XLEN 位的浮点存储（例如，RV32D 中的 FSD 指令），即使存储地址是自然对齐的，也可能出现相同的行为。

3.7.2 物理内存保护与分页

物理内存保护机制设计为与第 四章中描述的基于页面的虚拟内存系统组成。启用分页时，访问虚拟内存的指令可能会导致多次物理内存访问，包括对分页表的隐式引用。PMP 检查适用于所有这些访问。隐式页表访问的有效特权模式是 S。

使用虚拟内存的实现被允许投机性地执行地址转换，并且比显式内存访问所需的时间更早，并且被允许在地址转换缓存结构中缓存它们——包括可能缓存在裸转换模式和 m 模式中使用的从有效地址到物理地址的身份映射。结果物理地址的 PMP 设置可以在地址转换和显式内存访问之间的任何点被检查 (也可能被缓存)。因此，当 PMP 设置被修改时，m 模式软件必须将 PMP 设置与虚拟内存系统和任何 PMP 或地址转换缓存同步。这是通过执行 SFENCE 来完成的。在写入 PMP csr 之后，VMA 指令 *rs1*=x0 和 *rs2*=x0。

如果没有实现基于页面的虚拟内存，内存访问会同步检查 PMP 设置，因此不需要 SFENCE.VMA。

第四章　Supervisor-Level ISA, Version 1.12 监管者级指令集，版本 1.12

This chapter describes the RISC-V supervisor-level architecture, which contains a common core that is used with various supervisor-level address translation and protection schemes.

本章描述了 RISC-V 监管者级体系结构，它包含一个公共核心，可与各种监管者级别地址转换和保护方案一起使用。

Supervisor mode is deliberately restricted in terms of interactions with underlying physical hardware, such as physical memory and device interrupts, to support clean virtualization. In this spirit, certain supervisor-level facilities, including requests for timer and interprocessor interrupts, are provided by implementation-specific mechanisms. In some systems, a supervisor execution environment (SEE) provides these facilities in a manner specified by a supervisor binary interface (SBI). Other systems supply these facilities directly, through some other implementation-defined mechanism.

为实现干净的虚拟化，监管者模式在与底层物理硬件的交互方面被有意地限制了，如物理内存和设备中断。本着这种精神，某些监管者级的特殊操作，包括对计时器和处理器间中断的请求，都是由特定于实现的机制提供的。在某些系统中，监管器执行环境（*SEE*）以监管者二进制接口（*SBI*）指定的方式提供这些特殊操作。其他系统通过其他一些实现定义的机制直接提供这些特殊操作。

4.1　Supervisor CSRs 监管者 CSR

A number of CSRs are provided for the supervisor.

一些 CSR 被提供给监管者模式使用。

The supervisor should only view CSR state that should be visible to a supervisor-level operating system. In particular, there is no information about the existence (or non-existence) of higher privilege levels (machine level or other) visible in the CSRs accessible by the supervisor.

监管者模式应只查看应该对监管者级操作系统可见的 *CSR* 状态。特别是，在监管者模式可访问的 *CSR* 中，没有关于更高的特权级别（机器级或其他）的存在（或不存在）信息。

Many supervisor CSRs are a subset of the equivalent machine-mode CSR, and the machine-mode chapter should be read first to help understand the supervisor-level CSR descriptions.

许多监管者级 *CSR* 都是等效的机器模式 *CSR* 的一个子集，因此应该首先阅读机器模式章节，以帮助理解监管者级 *CSR* 的描述。

4.1.1 Supervisor Status Register (`sstatus`) 监管者级状态寄存器（`sstatus`）

The `sstatus` register is an SXLEN-bit read/write register formatted as shown in Figure 4.1 when SXLEN=32 and Figure 4.2 when SXLEN=64. The `sstatus` register keeps track of the processor's current operating state.

`sstatus` 寄存器是一个 SXLEN 位读/写寄存器，SXLEN=32 时格式如图4.1 所示，SXLEN=64 时格式如图4.2所示。`sstatus` 寄存器会跟踪处理器的当前运行状态。

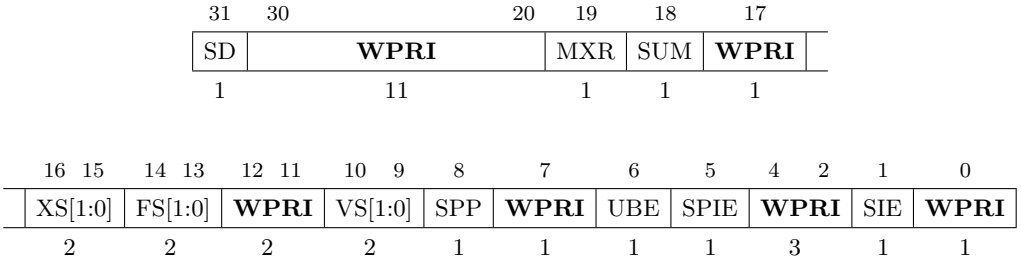


图 4.1: Supervisor-mode status register (`sstatus`) when SXLEN=32. SXLEN=32 时的监管者模式状态寄存器 (`sstatus`)

The SPP bit indicates the privilege level at which a hart was executing before entering supervisor mode. When a trap is taken, SPP is set to 0 if the trap originated from user mode, or 1 otherwise. When an SRET instruction (see Section 3.3.2) is executed to return from the trap handler, the privilege level is set to user mode if the SPP bit is 0, or supervisor mode if the SPP bit is 1; SPP is then set to 0.

SPP 位表示硬件线程在进入监管者模式之前执行的特权级别。当陷阱触发时，如果陷阱来自用户模式，则 SPP 设置为 0，否则设置为 1。当执行 SRET 指令（见第3.3.2节）从陷阱处理程序返回时，

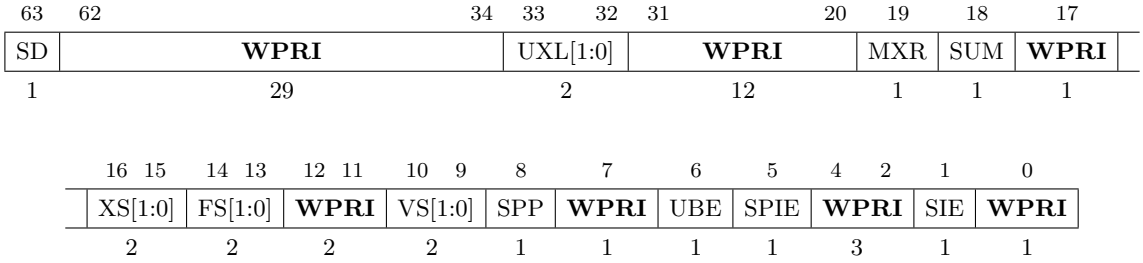


图 4.2: Supervisor-mode status register (`sstatus`) when SXLEN=64. SXLEN=64 时的监管者模式状态寄存器 (`sstatus`)

如果 SPP 位为 0，则将权限级别设置为用户模式；如果 SPP 位为 1，则设置为监管者模式；然后将 SPP 设置为 0。

The SIE bit enables or disables all interrupts in supervisor mode. When SIE is clear, interrupts are not taken while in supervisor mode. When the hart is running in user-mode, the value in SIE is ignored, and supervisor-level interrupts are enabled. The supervisor can disable individual interrupt sources using the `sie` CSR.

SIE 位在监管者模式下用于启用或禁用所有中断。当 SIE 被清除时，在监管者模式下禁止中断。当硬件线程在用户模式运行时，则忽略 SIE 中的值，并开启监管者级中断。监管者者可以使用 `sie` CSR 禁用单个中断源。

The SPIE bit indicates whether supervisor interrupts were enabled prior to trapping into supervisor mode. When a trap is taken into supervisor mode, SPIE is set to SIE, and SIE is set to 0. When an SRET instruction is executed, SIE is set to SPIE, then SPIE is set to 1.

SPIE 位指示在陷入监管者模式之前是否启用了监管者中断。当陷阱触发进入监管者模式时，SPIE 设置为 SIE，SIE 设置为 0。当执行 SRET 指令时，SIE 设置为 SPIE，然后 SPIE 设置为 1。

The `sstatus` register is a subset of the `mstatus` register.

`sstatus` 寄存器是 `mstatus` 寄存器的一个子集。

In a straightforward implementation, reading or writing any field in `sstatus` *is equivalent to reading or writing the homonymous field in* `mstatus`.

在一种简单的实现中，在 `sstatus` 中读或写任何字段都相当于在 `mstatus` 中读或写同名字段。

4.1.1.1 Base ISA Control in `sstatus` Register `sstatus` 寄存器中的基本控制

The UXL field controls the value of XLEN for U-mode, termed *UXLEN*, which may differ from the value of XLEN for S-mode, termed *SXLEN*. The encoding of UXL is the same as that of the MXL field of `misa`, shown in Table 3.1.

UXL 字段控制 U 模式下的 XLEN 值，称为 *UXLEN*，这可能与 S 模式的 XLEN 值不同，称为 *SXLEN*。UXL 的编码与 `misa` 的 MXL 域相同，如表3.1所示。

When SXLEN=32, the UXL field does not exist, and UXLEN=32. When SXLEN=64, it is a **WARL** field that encodes the current value of UXLEN. In particular, an implementation may make UXL be a read-only field whose value always ensures that UXLEN=SXLEN.

当 SXLEN=32 时，UXL 字段不存在，而 UXLEN=32。当 SXLEN=64 时，它是一个编码 UXLEN 当前值的 **WARL** 字段。特别是，一种实现可能使 UXL 成为一个只读字段，其值总是确保 UXLEN=SXLEN。

If UXLEN $\neq$ SXLEN, instructions executed in the narrower mode must ignore source register operand bits above the configured XLEN, and must sign-extend results to fill the widest supported XLEN in the destination register.

如果 UXLEN $\neq$ SXLEN，在较窄模式下执行的指令必须忽略高于配置 XLEN 的源寄存器操作数位，并且必须对结果进行符号扩展，以填充目标寄存器中支持的最宽的 XLEN。

If UXLEN $<$ SXLEN, user-mode instruction-fetch addresses and load and store effective addresses are taken modulo 2^{UXLEN}. For example, when UXLEN=32 and SXLEN=64, user-mode memory accesses reference the lowest 4 GiB of the address space.

如果 UXLEN $<$ SXLEN，用户模式的指令获取地址和加载和存储有效地址需要对 2^{UXLEN} 取模。例如，当 UXLEN=32 而 SXLEN=64 时，用户模式内存访问只引用地址空间的最低的4 GiB。

4.1.1.2 Memory Privilege in `sstatus` Register `sstatus` 寄存器中的内存权限

The MXR (Make eXecutable Readable) bit modifies the privilege with which loads access virtual memory. When MXR=0, only loads from pages marked readable (R=1 in Figure 4.33) will succeed. When MXR=1, loads from pages marked either readable or executable (R=1 or X=1) will succeed. MXR has no effect when page-based virtual memory is not in effect.

MXR（Make eXecutable Readable）位修改了加载访问虚拟内存的权限。当 MXR=0 时，只从标记为可读的页面（图4.33中的 R=1）加载。当 MXR=1 时，从标记为可读或可执行的页面（R=1 或 X=1）加载。当基于页面的虚拟内存无效时，MXR 没有影响。

The SUM (permit Supervisor User Memory access) bit modifies the privilege with which S-mode loads and stores access virtual memory. When SUM=0, S-mode memory accesses to pages that are accessible by U-mode (U=1 in Figure 4.33) will fault. When SUM=1, these accesses are permitted. SUM has no effect when page-based virtual memory is not in effect, nor when executing in U-mode. Note that S-mode can never execute instructions from user pages, regardless of the state of SUM.

SUM（permit Supervisor User Memory access）位修改 S 模式加载和存储指令访问虚拟内存的权限。当 SUM=0 时，S 模式访问 U 模式可访问的页面（图4.33中的 U=1）将出现错误。当使用 SUM=1 时，则允许访问这些内容。当基于页面的虚拟内存不生效时，或者运行在 U 模式下时，SUM 都没有影响。请注意，S 模式永远不能执行来自用户页面的指令，无论 SUM 的状态如何。

SUM is read-only 0 if `satp`.MODE is read-only 0.

如果 `satp`.MODE 为只读 0，则 SUM 也是只读 0。

The SUM mechanism prevents supervisor software from inadvertently accessing user memory. Operating systems can execute the majority of code with SUM clear; the few code segments that should access user memory can temporarily set SUM.

SUM 机制可以防止监管者级软件无意中访问用户内存。操作系统可以在 SUM 清 0 的情况下执行大部分代码；需要访问用户内存的少数代码段可以临时设置 SUM。

The SUM mechanism does not avail S-mode software of permission to execute instructions in user code pages. Legitimate uses cases for execution from user memory in supervisor context are rare in general and nonexistent in POSIX environments. However, bugs in supervisors that lead to arbitrary code execution are much easier to exploit if the supervisor exploit code can be stored in a user buffer at a virtual address chosen by an attacker.

SUM 机制不利用 S 模式软件的权限执行用户代码页中的指令。在监管者模式上下文中从用户内存中执行的合法用例通常很少见，而且在 POSIX 环境中也不存在。但是，如果监管者模式利用的代码可以存储在攻击者选定的用户缓冲区虚拟地址中，那么主管中可以执行任意代码的错误就更容易被利用。

Some non-POSIX single address space operating systems do allow certain privileged software to partially execute in supervisor mode, while most programs run in user mode, all in a shared address space. This use case can be realized by mapping the physical code pages at multiple virtual addresses with different permissions, possibly with the assistance of the instruction page-fault handler to direct supervisor software to use the alternate mapping.

一些非 POSIX 单地址空间操作系统确实允许某些特权软件在共享地址空间中，监管者模式下执行部分程序，而大多数程序在用户模式下运行。这个用例可以通过将物理代码页映射到

多个具有不同权限的虚拟地址上来实现，这可能需要在指令页错误处理程序的帮助下实现，以指导监管者软件使用备用映射。

4.1.1.3 Endianness Control in `sstatus` Register `sstatus` 寄存器中的字节序控制

The UBE bit is a **WARL** field that controls the endianness of explicit memory accesses made from U-mode, which may differ from the endianness of memory accesses in S-mode. An implementation may make UBE be a read-only field that always specifies the same endianness as for S-mode.

UBE 位是一个 **WARL** 字段，它控制在 U 模式下进行的显式内存访问的字节序，这可能不同于 S 模式下的内存访问的字节序。一种实现可以使 UBE 成为一个只读字段，它总是指定与 S 模式相同的字节序。

UBE controls whether explicit load and store memory accesses made from U-mode are little-endian (UBE=0) or big-endian (UBE=1).

UBE 控制在 U 模式中进行的显式加载和存储内存访问是小端字节序（UBE=0）还是大端字节序（UBE=1）。

UBE has no effect on instruction fetches, which are *implicit* memory accesses that are always little-endian.

UBE 对指令获取没有影响，指令获取是一种隐式内存访问，总是小段字节序。

For *implicit* accesses to supervisor-level memory management data structures, such as page tables, S-mode endianness always applies and UBE is ignored.

对监管者级内存管理数据结构的隐式访问，如页表，始终使用 S 模式的字节序，而忽略 UBE。

Standard RISC-V ABIs are expected to be purely little-endian-only or big-endian-only, with no accommodation for mixing endianness. Nevertheless, endianness control has been defined so as to permit an OS of one endianness to execute user-mode programs of the opposite endianness.

标准的 *RISC-V ABI* 被应该采用纯粹的大端字节序或者小端字节序，而不是两者混合。定义字节序控制是为了允许使用一种字节序的操作系统执行使用了相反字节序的用户模式程序。

4.1.2 Supervisor Trap Vector Base Address Register (stvec) 监管者级陷阱矢量基地址寄存器（stvec）

The `stvec` register is an SXLEN-bit read/write register that holds trap vector configuration, consisting of a vector base address (BASE) and a vector mode (MODE).

`stvec` 寄存器是一个 SXLEN 位读/写寄存器，它保存陷阱向量配置，由一个向量基地址（BASE）和一个向量模式（MODE）组成。

图 4.3: Supervisor trap vector base address register (`stvec`). 监管者级陷阱矢量基地址寄存器（`stvec`）

The BASE field in `stvec` is a **WARL** field that can hold any valid virtual or physical address, subject to the following alignment constraints: the address must be 4-byte aligned, and MODE settings other than Direct might impose additional alignment constraints on the value in the BASE field.

`stvec` 中的 BASE 字段是一个 **WARL** 字段，可以填充任何有效的虚拟或物理地址，受以下对齐约束：地址必须是 4 字节对齐，除 Direct 以外的 MODE 设置外，可能对 BASE 字段中的值施加额外的对齐约束。

Value	Name	Description
0	Direct	All exceptions set `pc` to BASE.
1	Vectored	Asynchronous interrupts set `pc` to BASE+4×cause.
≥2	—	*Reserved*

表 4.1: Encoding of `stvec` MODE field.

值	名称	描述
0	Direct	所有的异常都将 `pc` 设置为 BASE。
1	Vectored	异步中断将 `pc` 设置为 BASE+4×cause.
≥2	—	保留

表 4.2: `stvec` MODE 域的编码

The encoding of the MODE field is shown in Table 4.2. When MODE=Direct, all traps into supervisor mode cause the `pc` to be set to the address in the BASE field. When MODE=Vectored,

all synchronous exceptions into supervisor mode cause the `pc` to be set to the address in the BASE field, whereas interrupts cause the `pc` to be set to the address in the BASE field plus four times the interrupt cause number. For example, a supervisor-mode timer interrupt (see Table 4.4) causes the `pc` to be set to BASE+`0x14`. Setting MODE=Vectored may impose a stricter alignment constraint on BASE.

MODE 字段的编码如表4.2所示。当 MODE=Direct 时，所有进入监管者模式的陷阱会导致 `pc` 被设置为 BASE 字段中的地址。当 MODE=Vectored 时，所有进入监控模式的同步异常会导致 `pc` 设置为 BASE 字段中的地址，而中断会导致 `pc` 设置为 BASE 字段中的地址加上中断原因数的四倍。例如，一个监管者模式的计时器中断（见表4.4）会导致 `pc` 被设置为 BASE+`0x14`。设置 MODE=Vectored 可能会对 BASE 字段施加更严格的对齐约束。

4.1.3 Supervisor Interrupt Registers (`sip` and `sie`) 监管者级中断寄存器（`sip` 和 `sie`）

The `sip` register is an SXLEN-bit read/write register containing information on pending interrupts, while `sie` is the corresponding SXLEN-bit read/write register containing interrupt enable bits. Interrupt cause number i (as reported in CSR `scause`, Section 4.1.8) corresponds with bit i in both `sip` and `sie`. Bits 15:0 are allocated to standard interrupt causes only, while bits 16 and above are designated for platform or custom use.

`sip` 寄存器是一个包含待决中断信息的 SXLEN 位读/写寄存器，而 `sie` 是相应的包含中断启用位的 SXLEN 位读/写寄存器。中断原因编号 i（如 CSR `scause`，第4.1.8节所述）与 `sip` 和 `sie` 中的第 i 位对应。位 15：0 仅被分配给标准的中断原因，而位 16 位及以上被指定为平台或自定义使用。

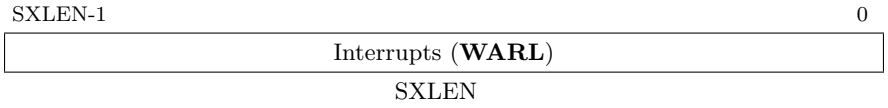


图 4.4: Supervisor interrupt-pending register (`sip`).

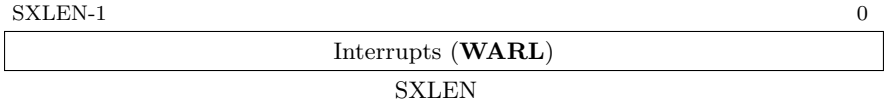


图 4.5: 监管者级中断寄存器（`sip`）

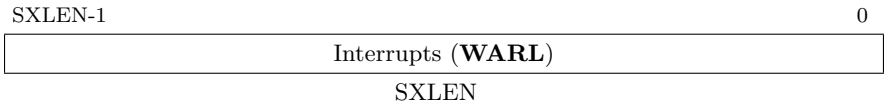


图 4.6: Supervisor interrupt-enable register (`sie`).

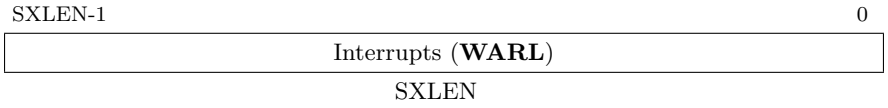


图 4.7: 监管者级中断寄存器（`sie`）

An interrupt i will trap to S-mode if both of the following are true: (a) either the current privilege mode is S and the SIE bit in the `sstatus` register is set, or the current privilege mode has less privilege than S-mode; and (b) bit i is set in both `sip` and `sie`.

如果同时满足以下两个条件，中断 i 将陷入到 S 模式：(a) 要么当前特权模式为 S，`sstatus` 寄存器中的 SIE 位被设置为 1，或者当前特权模式比 S 模式拥有更少的特权；(b) 第 i 位在 `sip` 和 `sie` 中都被设置。

These conditions for an interrupt trap to occur must be evaluated in a bounded amount of time from when an interrupt becomes, or ceases to be, pending in `sip`, and must also be evaluated immediately following the execution of an SRET instruction or an explicit write to a CSR on which these interrupt trap conditions expressly depend (including `sip`, `sie` and `sstatus`).

上述发生中断陷阱的条件必须在一定的时间内进行评估，这个时间从中断在 `sip` 中变为挂起或停止挂起时算起，还必须在执行 SRET 指令或显式写入 CSR 后立即对这些中断陷阱条件明确依赖的寄存器（包括 `sip`、`sie` 和 `sstatus`）进行评估。

Interrupts to S-mode take priority over any interrupts to lower privilege modes.

陷入到 S 模式的中断优先于任何陷入到较低特权模式的中断。

Each individual bit in register `sip` may be writable or may be read-only. When bit i in `sip` is writable, a pending interrupt i can be cleared by writing 0 to this bit. If interrupt i can become pending but bit i in `sip` is read-only, the implementation must provide some other mechanism for clearing the pending interrupt (which may involve a call to the execution environment).

寄存器 `sip` 中的每个单独的位可以是可写的，也可以是只读的。当 `sip` 中的第 i 位是可写的，一个挂起的中断 i 可以通过给这一位写 0 来清除。如果中断 i 可以挂起，但 `sip` 中的第 i 位是只读的，那么具体实现必须提供一些其他机制来清除待决的中断（这可能涉及对执行环境的调用）。

A bit in `sie` must be writable if the corresponding interrupt can ever become pending. Bits of `sie` that are not writable are read-only zero.

如果相应的中断可以挂起，`sie` 中的一个位必须是可写的。`sie` 中不可写的位是只读零。

The standard portions (bits 15:0) of registers `sip` and `sie` are formatted as shown in Figures 4.10 and 4.11 respectively.

寄存器 `sip` 和 `sie` 的标准部分（位 15：0）的格式分别如图4.10和4.11所示。

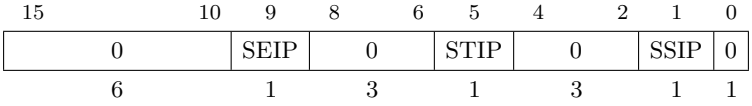


图 4.8: Standard portion (bits 15:0) of `sip`.

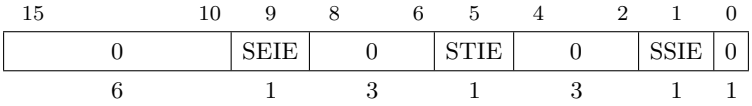


图 4.9: Standard portion (bits 15:0) of `sie`.

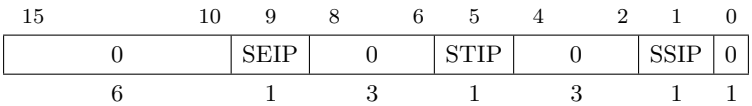


图 4.10: 寄存器 `sip` 的标准部分（位 15:0）

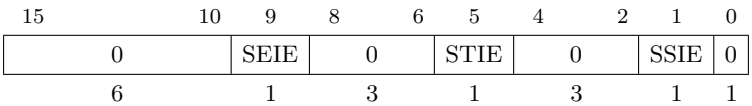


图 4.11: 寄存器 `sie` 的标准部分（位 15:0）

Bits `sip`.SEIP and `sie`.SEIE are the interrupt-pending and interrupt-enable bits for supervisor-level external interrupts. If implemented, SEIP is read-only in `sip`, and is set and cleared by the execution environment, typically through a platform-specific interrupt controller.

位 `sip`.SEIP 和 `sie`.SEIE 是监管者级外部中断的中断挂起位和中断启用位。如果实现，SEIP 在 `sip` 中是只读的，并由执行环境设置和清除，这个过程通常通过一个特定于平台的中断控制器。

Bits `sip`.STIP and `sie`.STIE are the interrupt-pending and interrupt-enable bits for supervisor-level timer interrupts. If implemented, STIP is read-only in `sip`, and is set and cleared by the execution environment.

位 `sip`.STIP 和 `sie`.STIE 是监管者级计时器中断的中断挂起位和中断启用位。如果实现，STIP 在 `sip` 中是只读的，并由执行环境设置和清除。

Bits `sip`.SSIP and `sie`.SSIE are the interrupt-pending and interrupt-enable bits for supervisor-level software interrupts. If implemented, SSIP is writable in `sip` and may also be set to 1 by a platform-specific interrupt controller.

位 `sip`.SSIP 和 `sie`.SSIE 是监管者级软件中断的中断挂起位和中断启用位。如果实现，SSIP 在 `sip` 中是可写的，也可以被特定于平台的中断控制器设置为 1。

Interprocessor interrupts are sent to other harts by implementation-specific means, which will ultimately cause the SSIP bit to be set in the recipient hart's `sip` *register.*

处理器间中断通过特定于具体实现的方式被发送到其他端口，这将最终导致在接收中断的 *hart* 的 `sip` 寄存器中的 *SSIP* 位被设置。

Each standard interrupt type (SEI, STI, or SSI) may not be implemented, in which case the corresponding interrupt-pending and interrupt-enable bits are read-only zeros. All bits in `sip` and `sie` are **WARL** fields. The implemented interrupts may be found by writing one to every bit location in `sie`, then reading back to see which bit positions hold a one.

每个标准中断类型（SEI、STI 或 SSI）可能未被实现，在这种情况下，相应的中断挂起位和中断启用位都是只读零。`sip` 和 `sie` 中的所有位都是 **WARL** 字段。实现的中断可以通过在 `sie` 中的每个位位置写入 1，然后回读查看哪个位保持 1 来找到。

The `sip` *and* `sie` *registers are subsets of the* `mip` *and* `mie` *registers. Reading any implemented field, or writing any writable field, of* `sip`*/*`sie` *effects a read or write of the homonymous field of* `mip`*/*`mie`*.*

`sip` 和 `sie` 寄存器是 `mip` 和 `mie` 寄存器的子集。读取 `sip`/`sie` 的任何已实现字段，或写入任何可写字段，都会影响 `mip`/`mie` 的同名字段的读写。

Bits 3, 7, and 11 of `sip` *and* `sie` *correspond to the machine-mode software, timer, and external interrupts, respectively. Since most platforms will choose not to make these interrupts delegatable from M-mode to S-mode, they are shown as 0 in Figures 4.10 and 4.11.*

`sip` 和 `sie` 的第 *3*、*7* 和 *11* 位分别对应于机器模式软件、计时器和外部中断。由于大多数平台选择不将这些中断从 *M* 模式委托给 *S* 模式，因此它们在图*4.10*和*4.11*中展示为 *0*。

Multiple simultaneous interrupts destined for supervisor mode are handled in the following decreasing priority order: SEI, SSI, STI.

多个分配给监管者模式的并发中断将按照以下优先级顺序处理：SEI、SSI、STI。

4.1.4 Supervisor Timers and Performance Counters 监管者级计时器和性能计数器

Supervisor software uses the same hardware performance monitoring facility as user-mode software, including the `time`, `cycle`, and `instret` CSRs. The implementation should provide a mechanism to modify the counter values.

监管者级软件使用与用户模式软件相同的硬件性能监控工具，包括 `time`、`cycle` 和 `instret` CSRs。具体实现应该提供一种修改计数器值的机制。

The implementation must provide a facility for scheduling timer interrupts in terms of the real-time counter, `time`.

具体实现必须提供一个根据实时计数器、`time` 来安排计时器中断的工具。

4.1.5 Counter-Enable Register (`scounteren`) 计数器启用寄存器（`scounteren`）

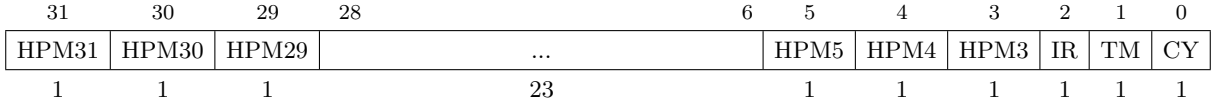


图 4.12: Counter-enable register (`scounteren`).

图 4.13: 计数器启用寄存器（`scounteren`）

The counter-enable register `scounteren` is a 32-bit register that controls the availability of the hardware performance monitoring counters to U-mode.

计数器启用寄存器 `scounteren` 是一个 32 位寄存器，它控制 U 模式下硬件性能监控计数器的可用性。

When the CY, TM, IR, or HPMn bit in the `scounteren` register is clear, attempts to read the `cycle`, `time`, `instret`, or `hpmcounter`n register while executing in U-mode will cause an illegal instruction exception. When one of these bits is set, access to the corresponding register is permitted.

当 `scounteren` 寄存器中的 CY、TM、IR 或 HPMn 位清 0 时，在 U 模式下执行尝试读取 `cycle`、`time`、`instret` 或 `hpmcounter`n 寄存器将导致非法指令异常。当这些位的其中一个被设置时，就允许访问相应的寄存器。

`scounteren` must be implemented. However, any of the bits may be read-only zero, indicating reads to the corresponding counter will cause an exception when executing in U-mode. Hence, they are effectively **WARL** fields.

必须实现 `scounteren`。然而，任何任意一位都可能是只读零，这表明在 U 模式下读取相应的计数器将导致异常。因此，它们实际上是 **WARL** 字段。

The setting of a bit in `mcounteren` *does not affect whether the corresponding bit in* `scounteren` *is writable. However, U-mode may only access a counter if the corresponding bits in* `scounteren` *and* `mcounteren` *are both set.*

`mcounteren` 中位的设置不影响 `scounteren` 中对应的位是否可写。然而，只有当 `scounteren` 和 `mcounteren` 中的相应的位都被设置时，*U* 型模式才能访问计数器。

4.1.6 Supervisor Scratch Register (`sscratch`) 监管者级 Scratch 寄存器 (`sscratch`)

The `sscratch` register is an SXLEN-bit read/write register, dedicated for use by the supervisor. Typically, `sscratch` is used to hold a pointer to the hart-local supervisor context while the hart is executing user code. At the beginning of a trap handler, `sscratch` is swapped with a user register to provide an initial working register.

`sscratch` 寄存器是一个 SXLEN 位读/写寄存器，专门供监管者模式使用。通常，`sscratch` 用于 hart 执行用户代码时保存指向本地硬件线程监管者模式上下文的指针。在陷阱处理程序的开始，`sscratch` 与用户寄存器交换，以提供一个初始的工作寄存器。

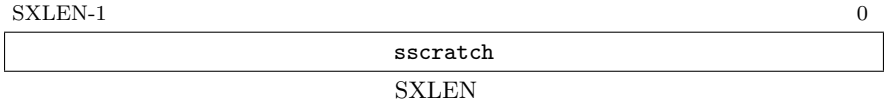


图 4.14: Supervisor Scratch Register.

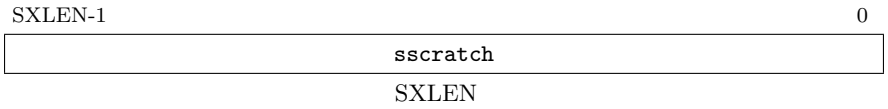


图 4.15: 监管者级 Scratch 寄存器

4.1.7 Supervisor Exception Program Counter (sepc) 监管者级异常程序计数器 (sepc)

sepc is an SXLEN-bit read/write register formatted as shown in Figure 4.17. The low bit of sepc (sepc[0]) is always zero. On implementations that support only IALIGN=32, the two low bits (sepc[1:0]) are always zero.

sepc 是一个 SXLEN 位读/写寄存器，其格式如图4.17所示。sepc 的最低位（sepc[0]）始终为零。在仅支持 IALIGN=32 的实现中，最低两位（sepc[1:0]）始终为零。

If an implementation allows IALIGN to be either 16 or 32 (by changing CSR misa, for example), then, whenever IALIGN=32, bit sepc[1] is masked on reads so that it appears to be 0. This masking occurs also for the implicit read by the SRET instruction. Though masked, sepc[1] remains writable when IALIGN=32.

如果一种实现允许 IALIGN 是 16 或 32（比如，通过改变 CSR misa），那么，每当 IALIGN=32 时，sepc[1] 在读取时将被屏蔽，使其看起来是 0。这种屏蔽也发生在 SRET 指令的隐式读取中。虽然读取被屏蔽了，但 sepc[1] 在 IALIGN=32 时仍然可写。

sepc is a **WARL** register that must be able to hold all valid virtual addresses. It need not be capable of holding all possible invalid addresses. Prior to writing sepc, implementations may convert an invalid address into some other invalid address that sepc is capable of holding.

sepc 是一个 **WARL** 寄存器，它必须能够保存所有有效的虚拟地址。它不需要能够保存所有可能的无效地址。在写入 sepc 之前，具体实现可以将一个无效地址转换为 sepc 能够保存的其他无效地址。

When a trap is taken into S-mode, sepc is written with the virtual address of the instruction that was interrupted or that encountered the exception. Otherwise, sepc is never written by the implementation, though it may be explicitly written by software.

当陷阱触发进入 S 模式时，被中断或遇到异常的指令的虚拟地址被写入 sepc。否则，sepc 永远不会由具体实现写入，尽管它可能由软件显式地写入。

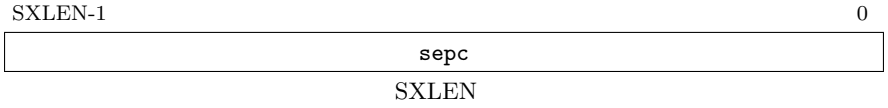


图 4.16: Supervisor exception program counter register.

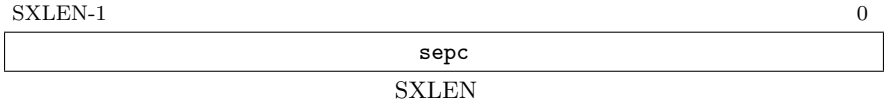


图 4.17: 监管者级异常程序计数器

4.1.8 Supervisor Cause Register (scause) 监管者级原因寄存器（scause）

The scause register is an SXLEN-bit read-write register formatted as shown in Figure 4.19. When a trap is taken into S-mode, scause is written with a code indicating the event that caused the trap. Otherwise, scause is never written by the implementation, though it may be explicitly written by software.

scause 寄存器是一个 SXLEN 位读写寄存器，其格式如图4.19所示。当陷阱触发进入 S 模式时，scause 会被写入指示导致陷阱的事件的代码。否则，scause 永远不会由具体实现写入，尽管它可能由软件显式地写入。

The Interrupt bit in the scause register is set if the trap was caused by an interrupt. The Exception Code field contains a code identifying the last exception or interrupt. Table 4.4 lists the possible exception codes for the current supervisor ISAs. The Exception Code is a **WLRL** field. It is required to hold the values 0–31 (i.e., bits 4–0 must be implemented), but otherwise it is only guaranteed to hold supported exception codes.

如果陷阱是由中断引起的，则会设置 scause 寄存器中的中断位。异常代码字段包含一个标识最后一次异常或中断的代码。表4.4列出了当前监管者级 ISAs 可能出现的异常代码。异常代码是一个 **WLRL** 字段。需要保留值 0–31（即，必须实现第 4–0 位），否则仅保证保留受支持的异常代码。

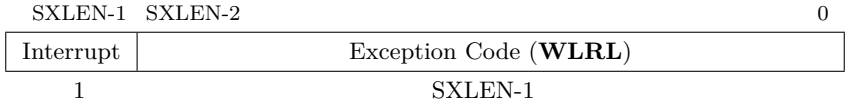


图 4.18: Supervisor Cause register scause.

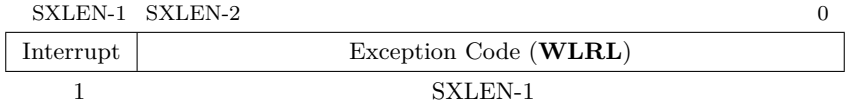


图 4.19: 监管者级原因寄存器 scause

Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2–4	*Reserved*
1	5	Supervisor timer interrupt
1	6–8	*Reserved*
1	9	Supervisor external interrupt
1	10–15	*Reserved*
1	≥16	*Designated for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10–11	*Reserved*
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	16–23	*Reserved*
0	24–31	*Designated for custom use*
0	32–47	*Reserved*
0	48–63	*Designated for custom use*
0	≥64	*Reserved*

表 4.3: Supervisor cause register (`scause`) values after trap. Synchronous exception priorities are given by Table 3.7.

4.1.9 Supervisor Trap Value (`stval`) Register 监管者级陷阱值（`stval`）寄存器

The `stval` register is an SXLEN-bit read-write register formatted as shown in Figure 4.21. When a trap is taken into S-mode, `stval` is written with exception-specific information to assist software

中断	异常编号	描述
1	0	保留
1	1	监管者软件中断
1	2–4	保留
1	5	监管者计时器中断
1	6–8	保留
1	9	监管者外部中断
1	10–15	保留
1	≥16	设计为平台使用
0	0	指令地址为对齐
0	1	指令访问故障
0	2	非法指令
0	3	断点
0	4	加载地址未对齐
0	5	加载访问故障
0	6	Store/AMO 地址未对齐
0	7	Store/AMO 访问故障
0	8	来自 U 模式的环境调用
0	9	来自 S 模式的环境调用
0	10–11	保留
0	12	指令页错误
0	13	加载页错误
0	14	保留
0	15	Store/AMO 页错误
0	16–23	保留
0	24–31	指定用于自定义用途
0	32–47	保留
0	48–63	指定用于自定义用途
0	≥64	保留

表 4.4: 陷入后监管者级原因寄存器 `scause` 的值，表3.7中给出了同步异常优先级

in handling the trap. Otherwise, `stval` is never written by the implementation, though it may be explicitly written by software. The hardware platform will specify which exceptions must set `stval` informatively and which may unconditionally set it to zero.

stval 寄存器是一个 SXLEN 位读写寄存器，其格式如图4.21所示。当陷阱触发进入 S 模式时，stval 会写入特定于异常的信息，以帮助软件处理陷阱。否则，stval 永远不会由具体实现写入，尽管它可能由软件显式地写入。硬件平台将指定哪些异常必须有信息含义地设置 stval，哪些可以无条件地将其设置为零。

If stval is written with a nonzero value when a breakpoint, address-misaligned, access-fault, or page-fault exception occurs on an instruction fetch, load, or store, then stval will contain the faulting virtual address.

如果在指令获取、加载或存储时发生断点、地址错位、访问错误或缺页异常时 stval 被写入非零值，则 stval 中将包含错误的虚拟地址。

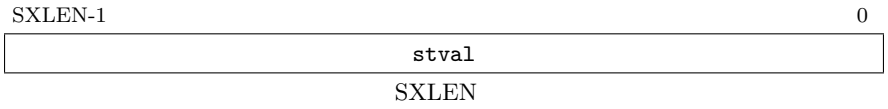


图 4.20: Supervisor Trap Value register.

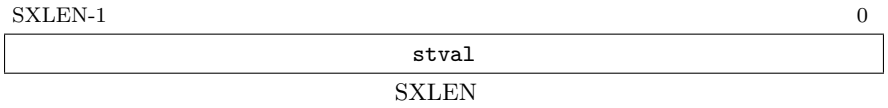


图 4.21: 监管者级陷阱值寄存器

If stval is written with a nonzero value when a misaligned load or store causes an access-fault or page-fault exception, then stval will contain the virtual address of the portion of the access that caused the fault.

如果在不对齐的加载或存储导致访问故障或缺页异常时 stval 被写入非零值，则 stval 将包含导致故障的访问部分的虚拟地址。

If stval is written with a nonzero value when an instruction access-fault or page-fault exception occurs on a system with variable-length instructions, then stval will contain the virtual address of the portion of the instruction that caused the fault, while sepc will point to the beginning of the instruction.

如果在支持可变长指令的系统上出现指令访问故障或缺页异常时 stval 被写入非零值，则 stval 将包含导致错误的指令部分的虚拟地址，而 sepc 将指向指令的开头。

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 stval is written with a nonzero value when an illegal-instruction exception occurs, then stval will contain the shortest of:

`stval` 寄存器还可以选择性地用于返回非法指令异常上的故障指令位（`sepc` 指向内存中的故障指令）。如果在发生非法指令异常时，`stval` 是用非零值写入的，则 `stval` 将包含以下各项最短者：

- the actual faulting instruction
- the first ILEN bits of the faulting instruction
- the first SXLEN bits of the faulting instruction
- 实际故障指令
- 故障指令的第一个 ILEN 位
- 故障指令的第一个 SXLEN 位

The value loaded into `stval` on an illegal-instruction exception is right-justified and all unused upper bits are cleared to zero.

在非法指令异常上加载到 `stval` 的值是右对齐的，并且所有未使用的高位都被清零。

For other traps, `stval` is set to zero, but a future standard may redefine `stval`'s setting for other traps.

对于其他陷阱，`stval` 被设置为零，但是未来的标准可能会为其他陷阱重新定义 `stval` 的设置。

`stval` is a **WARL** register that must be able to hold all valid virtual addresses and the value 0. It need not be capable of holding all possible invalid addresses. Prior to writing `stval`, implementations may convert an invalid address into some other invalid address that `stval` is capable of holding. 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 SXLEN and ILEN.

`stval` 是一个 **WARL** 寄存器，它必须能够保存所有有效的虚拟地址和值 0。它不需要能够保存所有可能的无效地址。在写入 `stval` 之前，具体实现可能会将一个无效地址转换为 `stval` 能够保存的其他无效地址。如果实现了返回故障指令位的特性，那么 `stval` 还必须能够保存所有小于 2^N 的值，其中 N 是 SXLEN 和 ILEN 的较小值。

4.1.10 Supervisor Environment Configuration Register (`senvcfg`) 监管者级环境配置寄存器（`senvcfg`）

The `senvcfg` CSR is an SXLEN-bit read/write register, formatted as shown in Figure 4.23, that controls certain characteristics of the U-mode execution environment.

`senvcfg` CSR 是一个 SXLEN 位读/写寄存器，格式如图4.23所示，它控制 U 模式执行环境的一些特性。

SXLEN-1 … 8	7	6	5 4	3 1	0
WPRI	CBZE	CBCFE	CBIE	**WPRI**	FIOM
SXLEN-8	1	1	2	3	1

图 4.22: Supervisor environment configuration register (`senvcfg`).

SXLEN-1 … 8	7	6	5 4	3 1	0
WPRI	CBZE	CBCFE	CBIE	**WPRI**	FIOM
SXLEN-8	1	1	2	3	1

图 4.23: 监管者级环境配置寄存器（`senvcfg`）

If bit FIOM (Fence of I/O implies Memory) is set to one in `senvcfg`, FENCE instructions executed in U-mode are modified so the requirement to order accesses to device I/O implies also the requirement to order main memory accesses. Table 4.6 details the modified interpretation of FENCE instruction bits PI, PO, SI, and SO in U-mode when FIOM=1.

如果在 `senvcfg` 中，位 FIOM（Fence of I/O implies Memory ）被设置为 1，那么在 U 模式下执行的 FENCE 指令将会被修改，因此对设备 I/O 进行顺序访问的要求也意味着对主存进行顺序访问的要求。表4.6详细说明了在 FIOM=1 时，在 U 模式下的 FENCE 指令位 PI、PO、SI 和 SO 的修改解释。

Similarly, for U-mode when FIOM=1, if an atomic instruction that accesses a region ordered as device I/O has its *aq* and/or *rl* bit set, then that instruction is ordered as though it accesses both device I/O and memory.

类似地，对于 U 模式，当 FIOM=1 时，如果原子指令访问按设备 I/O 顺序排列的区域，且指令设置了 *aq* 或 *rl* 位，那么该指令的顺序就好像它同时访问设备 I/O 和内存一样。

If `satp`.MODE is read-only zero (always Bare), the implementation may make FIOM read-only zero.

如果 `satp`.MODE 为只读零（总为 Bare），实现中可能使 FIOM 为只读零。

Instruction bit	Meaning when set
PI	Predecessor device input and memory reads (PR implied)
PO	Predecessor device output and memory writes (PW implied)
SI	Successor device input and memory reads (SR implied)
SO	Successor device output and memory writes (SW implied)

表 4.5: Modified interpretation of FENCE predecessor and successor sets in U-mode when FIOM=1.

指令位	设置该位时的含义
PI	前置设备输入和存储器读取（隐含 PR）
PO	前置设备输出和存储器写入（隐含 PW）
SI	后续设备输入和存储器读取（隐含 SR）
SO	后续设备输出和存储器写入（隐含 SW）

表 4.6: 当 FIOM=1 时，对 U 模式下 FENCE 前置集和后继集修改的解释

Bit FIOM exists for a specific circumstance when an I/O device is being emulated for U-mode and both of the following are true: (a) the emulated device has a memory buffer that should be I/O space but is actually mapped to main memory via address translation, and (b) multiple physical harts are involved in accessing this emulated device from U-mode.

当正在为 *U* 模式模拟 *I/O* 设备，并且以下两个条件都成立时，位 *FIOM* 用于特殊情况：(*a*) 模拟设备有一个内存缓冲区，它本应是 *I/O* 空间，但实际上是通过地址转换映射到主存；(*b*) 从 *U* 模式访问此模拟设备时涉及多个物理硬件线程。

A hypervisor running in S-mode without the benefit of the hypervisor extension of Chapter 八 may need to emulate a device for U-mode if paravirtualization cannot be employed. If the same hypervisor provides a virtual machine (VM) with multiple virtual harts, mapped one-to-one to real harts, then multiple harts may concurrently access the emulated device, perhaps because: (a) the guest OS within the VM assigns device interrupt handling to one hart while the device is also accessed by a different hart outside of an interrupt handler, or (b) control of the device (or partial control) is being migrated from one hart to another, such as for interrupt load balancing within the VM. For such cases, guest software within the VM is expected to properly coordinate access to the (emulated) device across multiple harts using mutex locks and/or interprocessor interrupts as usual, which in part entails executing I/O fences. But those I/O fences may not be sufficient if some of the device "I/O" is actually main memory, unknown to the guest. Setting FIOM=1 modifies those fences (and all other I/O fences executed in U-mode) to include main memory, too.

运行在 *S* 模式下并且没有第八章的虚拟机管理器扩展的虚拟机管理器，如果不能使用平行虚拟化，可能需要为 *U* 模式模拟设备。如果同一个虚拟机管理器提供了具有多个虚拟硬件线程的虚拟机（*VM*），并一对一地映射到真实的硬件线程，那么多个硬件线程可能并发地访问模拟设备，这可能是因为：(*a*) 虚拟机中的宾客操作系统将设备中断处理分配给一个硬件线程，而设备也被中断处理程序之外的另一个硬件线程访问，或者（*b*）设备的控制（或部分控制）正在从一个硬件线程迁移到另一个硬件线程，例如在虚拟机内的中断负载平衡。对于这种情况，虚拟机内的客户软件应该像往常一样使用互斥锁以及处理器间中断正确地协调跨多个硬件线程的对（模拟）设备的访问，这在一定程度上使执行 *I/O* 屏障成为必要。但是，如果某些设备"*I/O*"实际上是在主存进行的，那么这些 *I/O* 屏障可能是不够的，而客户并不知道这些。设置 *FIOM=1* 还会将这些屏障（以及 *U* 模式下执行的所有其他 *I/O* 屏障）修改为包含主存。

Software can always avoid the need to set FIOM by never using main memory to emulate a device memory buffer that should be I/O space. However, this choice usually requires trapping all

U-mode accesses to the emulated buffer, which might have a noticeable impact on performance. The alternative offered by FIOM is sufficiently inexpensive to implement that we consider it worth supporting even if only rarely enabled.

软件总是可以通过从不使用主存来模拟一个本应该是 I/O 空间的设备内存缓冲区来避免设置 FIOM 的需要。然而，这种选择通常需要捕获 U 模式下所有对模拟缓冲区的访问，这可能会对性能产生显著的影响。FIOM 提供的替代方案足够便宜，即使很少启用，我们也认为它值得支持。

The definition of the CBZE field will be furnished by the forthcoming Zicboz extension. Its allocation within `senvcfg` may change prior to the ratification of that extension.

CBZE 字段的定义将由即将到来的 Zicboz 扩展提供。在批准该拓展之前，其在 `senvcfg` 内的分配可能会改变。

The definitions of the CBCFE and CBIE fields will be furnished by the forthcoming Zicbom extension. Their allocations within `senvcfg` may change prior to the ratification of that extension.

CBCFE 和 CBIE 字段的定义将由即将到来的 Zicbom 扩展来提供。在批准该拓展之前，它们在 `senvcfg` 内的分配可能会改变。

4.1.11 Supervisor Address Translation and Protection (`satp`) Register 监管者地址转换和保护（`satp`）寄存器

The `satp` register is an SXLEN-bit read/write register, formatted as shown in Figure 4.25 for SXLEN=32 and Figure 4.27 for SXLEN=64, 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. Further details on the access to this register are described in Section 3.1.6.5.

`satp` 寄存器是一个 SXLEN 位读/写寄存器，格式如 SXLEN=32 的图4.25和 SXLEN=64 的图4.27所示，它控制监管者模式的地址转换和保护。该寄存器保存根页表的物理页号（PPN），即其监管者物理地址除以4 KiB；地址空间标识符（ASID），方便在每个地址空间的基础上进行地址转换；和 MODE 字段，它选择当前的地址转换方案。关于访问该寄存器的进一步细节见第3.1.6.5节。

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

在 `satp` 中存储 PPN，而不是物理地址，使 RV32 支持大于4 GiB的物理地址空间。

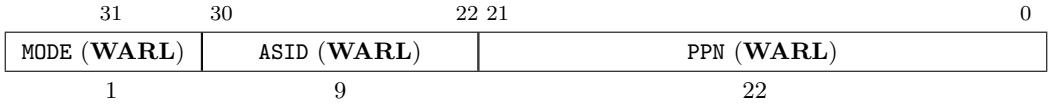


图 4.24: Supervisor address translation and protection register `satp` when SXLEN=32.

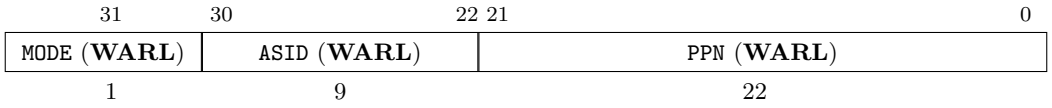


图 4.25: SXLEN=32 时的监管者地址转换和保护寄存器 `satp`

The `satp`*.PPN field might not be capable of holding all physical page numbers. Some platform standards might place constraints on the values* `satp`*.PPN may assume, e.g., by requiring that all physical page numbers corresponding to main memory be representable.*

`satp`*.PPN* 字段可能无法表示所有的物理页码。一些平台标准可能会对 `satp`*.PPN* 的值施加限制，例如，通过要求与主存对应的所有物理页号都是可表示的。

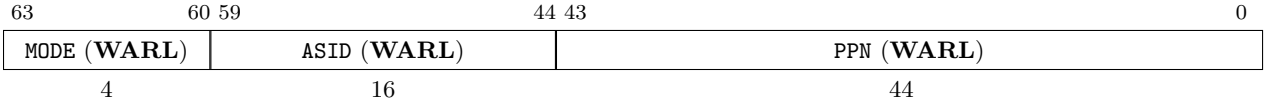


图 4.26: Supervisor address translation and protection register `satp` when SXLEN=64, for MODE values Bare, Sv39, Sv48, and Sv57.

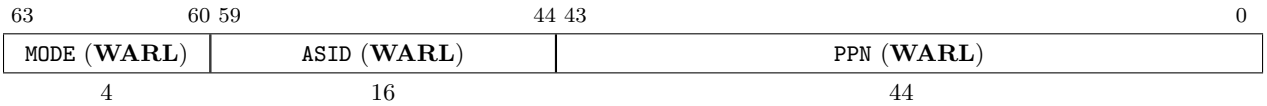


图 4.27: 对于 MODE 值为 Bare、Sv39、Sv48 以及 Sv57，SXLEN=64 时的监管者地址转换和保护寄存器 `satp`

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.

我们将 *ASID* 和页表基地址存储在同一个 *CSR* 中，以允许在上下文切换时原子地更改这对数据对。以非原子的方式修改它们可能会使新的转换地址污染旧的虚拟地址空间，反之亦然。这种方法还略微降低了上下文切换的成本。

Table 4.8 shows the encodings of the MODE field when SXLEN=32 and SXLEN=64. 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 3.7. To select MODE=Bare, software must write zero to the remaining fields of `satp` (bits 30–0 when SXLEN=32, or bits 59–0 when SXLEN=64). Attempting to select MODE=Bare with a nonzero pattern in the remaining fields has an 未指定的 effect on the value that the remaining fields assume and an 未指定的 effect on address translation and protection behavior.

表4.8显示了当 SXLEN=32 和 SXLEN=64 时，MODE 字段的编码。当 MODE=Bare 时，监管者虚拟地址等于监管者物理地址，并且除了第3.7节中描述的物理内存保护方案之外，没有额外的内存保护。要选择 MODE=Bare，软件必须将零写入 `satp` 的其余字段（SXLEN=32 时为第 30–0 位，或 SXLEN=64 时为第 59–0 位）。在其余字段非零模式下尝试选择 MODE=Bare 会对其余字段假定的值产生一个未指定的（未指定的 ）影响，并对地址转换和保护行为产生未指定的（未指定的 ）影响。

When SXLEN=32, the `satp` encodings corresponding to MODE=Bare and ASID[8:7]=3 are designated for custom use, whereas the encodings corresponding to MODE=Bare and ASID[8:7]≠3 are reserved for future standard use. When SXLEN=64, all `satp` encodings corresponding to MODE=Bare are reserved for future standard use.

当 SXLEN=32 时，MODE=Bare 和 ASID[8:7]=3 对应的 `satp` 编码被指定为自定义使用，而 MODE=Bare 和 ASID[8:7]≠3 对应的编码被保留以供将来的标准使用。当 SXLEN=64 时，所有与 MODE=Bare 对应的 `satp` 编码都被保留，以供未来的标准使用。

Version 1.11 of this standard stated that the remaining fields in `satp` *had no effect when MODE=Bare. Making these fields reserved facilitates future definition of additional translation and protection modes, particularly in RV32, for which all patterns of the existing MODE field have already been allocated.*

本标准的 *1.11* 版本指出，当 *MODE=Bare* 时，`satp` 中的其余字段没有效果。保留这些字段有助于未来定义额外的转换和保护模式，特别是在 *RV32* 中，因为存在的 *MODE* 字段的所有模式已经被分配。

When SXLEN=32, the only other valid setting for MODE is Sv32, a paged virtual-memory scheme described in Section 4.3.

当 SXLEN=32 时，MODE 的唯一其他有效设置是 Sv32，这是第4.3节中描述的一种分页虚拟内存方案。

When SXLEN=64, three paged virtual-memory schemes are defined: Sv39, Sv48, and Sv57, described in Sections 4.4, 4.5, and 4.6, respectively. One additional scheme, 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`.

当 SXLEN=64 时，定义了三种分页虚拟内存方案：Sv39、Sv48 和 Sv57，分别在第4.4、4.5和4.6节中描述。另外一个方案 Sv64 将在本规范的后续版本中定义。其余的 MODE 设置被保留以供将来使用，并可能定义 `satp` 中其他字段的不同解释。

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.

具体实现中不需要支持所有的 MODE 设置，如果 `satp` 中写入了不支持的 MODE，则将不会有任何效果；`satp` 中的任何字段都不会被修改。

SXLEN=32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32	Page-based 32-bit virtual addressing (see Section 4.3).

SXLEN=64		
Value	Name	Description
0	Bare	No translation or protection.
1–7	—	*Reserved for standard use*
8	Sv39	Page-based 39-bit virtual addressing (see Section 4.4).
9	Sv48	Page-based 48-bit virtual addressing (see Section 4.5).
10	Sv57	Page-based 57-bit virtual addressing (see Section 4.6).
11	*Sv64*	*Reserved for page-based 64-bit virtual addressing.*
12–13	—	*Reserved for standard use*
14–15	—	*Designated for custom use*

表 4.7: Encoding of `satp` MODE field.

The number of ASID bits is 未指定的 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, Sv48, and Sv57.

ASID 位数是未指定（未指定的 ）的话，可能为零。实现的 ASID 位的数量，称为*ASIDLEN*，可以通过向 ASID 字段中的每个位写入 1，然后读取 `satp` 中的值，查看 ASID 字段中的哪些位保持 1 来确定。ASID 的最低有效位首先被实现：也就是说，如果 ASIDLEN > 0，ASID[ASIDLEN-1:0] 是可写的。ASIDLEN 的最大值称为 ASIDMAX，对于 Sv32 为 9，Sv39、Sv48 和 Sv57 为 16。

SXLEN=32		
值	名称	描述
0	Bare	没有转换或保护。
1	Sv32	基于页面的 32 位虚拟寻址（见第4.3节）。

SXLEN=64		
值	名称	描述
0	Bare	没有转换或保护。
1–7	—	保留用于标准用途
8	Sv39	基于页面的 39 位虚拟寻址（见第4.4节）。
9	Sv48	基于页面的 48 位虚拟寻址（见第4.5节）。
10	Sv57	基于页面的 57 位虚拟寻址（见第4.6节）。
11	*Sv64*	保留用于基于页面的 *64* 位虚拟寻址。
12–13	—	保留用于标准用途
14–15	—	指定用于自定义用途

表 4.8: `satp` MODE 域的编码

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.

对于许多应用程序而言，页面大小的选择对性能有实质性的影响。一个大的页面大小增加了 *TLB* 的覆盖范围，并放松了对用虚拟地址中部分位域作为索引、物理地址中部分位域作为标记的物理高速缓存的关联约束。与此同时，大的页面加剧了内部碎片化，浪费了物理内存和可能的缓存容量。

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.

经过深思熟虑，我们已经确定了 *RV32* 和 *RV64* 的常规页面大小为 *4 KiB*。我们希望这个决定可以简化低级运行时软件和设备驱动程序的移植。现代操作系统 *[2]* 中对透明的超级页的支持改善了 *TLB* 覆盖范围的问题。此外，相比于映射地址空间的多级高速缓存层次结构，多级 *TLB* 层次结构非常便宜。

The `satp` register is considered *active* when the effective privilege mode is S-mode or U-mode. Executions of the address-translation algorithm may only begin using a given value of `satp` when `satp` is active.

当有效特权模式为 S 模式或 U 模式时，`satp` 寄存器被认为是活动的（*active*）。只有当 `satp` 处于活动状态时，地址转换算法才能开始使用给定的 `satp` 值执行。

Translations that began while `satp` *was active are not required to complete or terminate when* `satp` *is no longer active, unless an SFENCE.VMA instruction matching the address and ASID is executed. The SFENCE.VMA instruction must be used to ensure that updates to the address-translation data structures are observed by subsequent implicit reads to those structures by a hart.*

除非执行与地址和 *ASID* 匹配的 *SFENCE.VMA* 指令，否则当 `satp` 不再处于活动状态时，在 `satp` 处于活动状态期间开始的转换不需要完成或终止。*SFENCE.VMA* 指令必须被用来确保对地址转换数据结构的更新可以被随后隐式读取这些结构的硬件线程观察到。

Note that writing `satp` does not imply any ordering constraints between page-table updates and subsequent address translations, nor does it imply any invalidation of address-translation caches. If the new address space's page tables have been modified, or if an ASID is reused, it may be necessary to execute an SFENCE.VMA instruction (see Section 4.2.1) after, or in some cases before, writing `satp`.

请注意，写入 `satp` 并不意味着页表更新和后续的地址转换之间有任何顺序约束，也不意味着地址转换缓存的无效。如果新地址空间的页表已被修改，或者如果 ASID 被重用，则可能需要在写入 `satp` 之后，或在某些情况下在写入之前执行 SFENCE.VMA 指令（见第4.2.1节）。

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

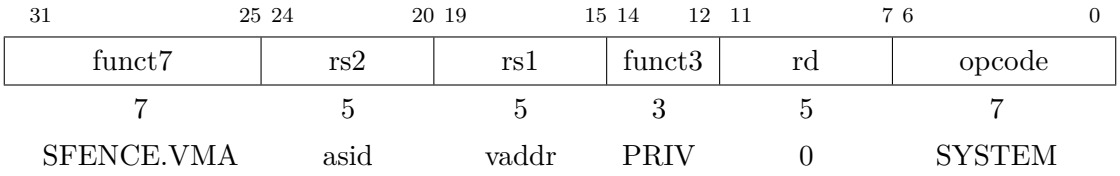
如果有足够大的 *ASID* 空间，不强制实现在 `satp` 写入时刷新地址转换缓存可降低上下文切换的成本。

4.2 Supervisor Instructions 监管者指令

In addition to the SRET instruction defined in Section 3.3.2, one new supervisor-level instruction is provided.

除了在第3.3.2节中定义的 SRET 指令外，还提供了一个新的监管者级指令。

4.2.1 Supervisor Memory-Management Fence Instruction 监管者内存管理屏障指令



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 explicit loads and stores. Executing an SFENCE.VMA instruction guarantees that any previous stores already visible to the current RISC-V hart are ordered before certain implicit references by subsequent instructions in that hart to the memory-management data structures. The specific set of operations ordered by SFENCE.VMA is determined by *rs1* and *rs2*, as described below. SFENCE.VMA is also used to invalidate entries in the address-translation cache associated with a hart (see Section 4.3.2). Further details on the behavior of this instruction are described in Section 3.1.6.5 and Section 3.7.2.

监管者内存管理屏障指令 SFENCE.VMA 用于将内存中内存管理数据结构的更新与当前运行同步。指令执行会引起对这些数据结构的隐式读写；然而，这些隐式的引用通常不会根据显式加载和存储进行排序。SFENCE.VMA 指令的执行可保证任何之前的、对当前的硬件线程可视的存储动作，在来自那个硬件线程中的后续指令的对内存管理数据结构的某些隐式引用之前进行排序。SFENCE.VMA 规定的特定操作由 *rs1* 和 *rs2* 确定，如下所述。SFENCE.VMA 还用于使与 hart 相关联的地址转换缓存中的条目无效（见第4.3.2节）。关于本指令行为的进一步细节见第3.1.6.5节和第3.7.2节。

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.

SFENCE.VMA 用于刷新所有与地址转换相关的本地硬件缓存。它被指定为一个屏障而不是 *TLB* 刷新，是为了提供关于哪些指令受刷新操作影响的更清晰的语义，并支持更广泛的动态缓存结构和内存管理方案。更高权限级别也使用 *SFENCE.VMA* 来同步页表写入和地址转换硬件。

SFENCE.VMA orders only the local hart's implicit references to the memory-management data structures.

SFENCE.VMA 只对本地 hart 对内存管理数据结构的隐式引用进行排序。

Consequently, other harts must be notified separately when the memory-management data structures have been modified. 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.

因此，当内存管理数据结构被修改时，必须分别通知其他硬件线程。一种方法是：首先 1) 使用本地数据围栏，以确保本地写入在全局内可见，然后 2) 将一个处理器间中断发送到其他线程，随后 3) 在远程线程的中断处理程序中使用本地 SFENCE.VMA，最后 4) 重新向原始线程发出信号，表明操作已完成。当然，这是 RISC-V 模拟 TLB 被击落。

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:

对于转换数据结构仅为单个地址映射（即一页或超级页）进行了修改的常见情况，*rs1* 可以在该映射中指定一个虚拟地址，以仅为该映射实现转换屏障。此外，对于转换数据结构仅为单个地址空间标识符进行了修改的常见情况，*rs2* 可以指定地址空间。SFENCE.VMA 的行为取决于 *rs1* 和 *rs2*，如下所示：

- 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. The fence also invalidates all address-translation cache entries, 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. The fence also invalidates all address-translation cache entries matching the address space identified by integer register *rs2*, except for entries containing global mappings.
- If *rs1*≠`x0` and *rs2*=`x0`, the fence orders only reads and writes made to leaf page table entries corresponding to the virtual address in *rs1*, for all address spaces. The fence also invalidates all address-translation cache entries that contain leaf page table entries 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 leaf page table entries corresponding to the virtual address in *rs1*, for the address space identified by integer register *rs2*. Accesses to global mappings are not ordered. The fence also invalidates all address-translation cache entries that contain leaf page table entries corresponding to the virtual address in *rs1* and that match the address space identified by integer register *rs2*, except for entries containing global mappings.

- 如果 *rs1*=x0 且 *rs2*=x0，则屏障指令将对所有地址空间的页表的全部级别的所有读取和写入进行排序。屏障指令还将使所有地址空间中，包含与 rs1 中的虚拟地址对应的叶页表条目的所有地址转换缓存条目无效。

- 如果 *rs1*=x0 且 *rs2*≠x0，屏障指令将对所有页表的读取和写入进行排序，但仅对整数寄存器 *rs2* 标识的地址空间进行排序。对全局映射的访问（见第4.3.1节）不进行排序。除去包含全局映射的条目外，该屏障指令还将使包含与整数寄存器 *rs2* 标识的地址空间匹配的所有地址转换缓存条目无效。

- 如果 *rs1*≠x0 且 *rs2*=x0，则对于所有地址空间，屏障指令仅对与 *rs1* 中的虚拟地址相对应的叶页表条目进行读取和写入。对于所有地址空间，该屏障指令还使包含与 *rs1* 中的虚拟地址相对应的叶页表条目的所有地址转换缓存条目无效。

- 如果 *rs1*≠x0 且 *rs2*≠x0，则，屏障指令将仅对整数寄存器 *rs2* 标识的地址空间中，作用于与 *rs1* 中的虚拟地址相对应的叶页表条目的读取和写入进行排序。对全局映射的访问不进行排序。除去包含全局映射的条目外，该屏障指令还将使包含与 *rs1* 中的虚拟地址对应的叶页表条目以及与整数寄存器 *rs2* 标识的地址空间匹配的所有地址转换缓存条目无效。

If the value held in *rs1* is not a valid virtual address, then the SFENCE.VMA instruction has no effect. No exception is raised in this case.

如果在 *rs1* 中保存的值不是一个有效的虚拟地址，则 SFENCE.VMA 指令没有效果。在这种情况下，不会出现任何异常。

When *rs2*≠x0, bits SXLEN-1:ASIDMAX of the value held in *rs2* are reserved for future standard use. Until their use is defined by a standard extension, they 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*.

当 *rs2*≠x0 时，*rs2* 保持的位 SXLEN-1:ASIDMAX 被保留，以供将来的标准使用。在它们的使用被标准扩展定义之前，它们应该被软件归零，并且被当前的具体实现忽略。此外，如果 ASIDLEN < ASIDMAX，则具体实现应忽略 *rs2* 中保持的值的位 ASIDMAX-1:ASIDLEN。

It is always legal to over-fence, e.g., by fencing only based on a subset of the bits in rs1 *and/or* rs2*, and/or by simply treating all SFENCE.VMA instructions as having* rs1=x0 *and/or* rs2=x0*. For example, simpler implementations can ignore the virtual address in* rs1 *and the ASID value in* rs2 *and always perform a global fence. The choice not to raise an exception when an invalid virtual address is held in* rs1 *facilitates this type of simplification.*

越过屏障总是合法的，例如，仅基于 rs1 以及 rs2 中的一个子集进行屏障跨越，以及简单地将所有 *SFENCE.VMA* 指令视为 rs1=x0 以及 rs2=x0。例如，一种更简单的实现可以是忽略 rs1 中的虚拟地址和 rs2 中的 *ASID* 值，并且始终执行一个全局的屏障指令。当在 rs1 中保存了一个无效的虚拟地址时，选择不引发异常促进了这种类型的简化。

An implicit read of the memory-management data structures may return any translation for an address that was valid at any time since the most recent SFENCE.VMA that subsumes that address. The ordering implied by SFENCE.VMA does not place implicit reads and writes to the memory-management data structures into the global memory order in a way that interacts cleanly with the standard RVWMO ordering rules. In particular, even though an SFENCE.VMA orders prior explicit accesses before subsequent implicit accesses, and those implicit accesses are ordered before their associated explicit accesses, SFENCE.VMA does not necessarily place prior explicit accesses before subsequent explicit accesses in the global memory order. These implicit loads also need not otherwise obey normal program order semantics with respect to prior loads or stores to the same address.

内存管理数据结构的隐式读取可能会返回对一个地址的全部转换，此地址自将它的包含的最近的 SFENCE.VMA 指令以来，任何时候都有效。SFENCE.VMA 暗示的顺序将与标准 RVWMO 顺序规则清晰地交互，不会以将对内存管理数据结构的隐式读取和写入置于全局内存顺序中。特别是，即使 SFENCE.VMA 在后续隐式访问之前先对显式访问进行排序，并且这些隐式访问在相关显式访问之前进行排序，SFENCE.VMA 也不一定在全局内存顺序中将先前显式访问置于后续显式访问之后。这些隐式加载也不需要遵守与先前加载或存储到同一地址相关的正常程序顺序语义。

A consequence of this specification is that an implementation may use any translation for an address that was valid at any time since the most recent SFENCE.VMA that subsumes that address. In particular, if a leaf PTE is modified but a subsuming SFENCE.VMA is not executed, either the old translation or the new translation will be used, but the choice is unpredictable. The behavior is otherwise well-defined.

该规范的一个结果是，具体实现可以使用一个地址的任意转换，该地址自最近包含它的最新 *SFENCE.VMA* 执行以来，任何时候都是有效的。特别是，如果修改了叶 *PTE*，但未执行包含 *SFENCE.VMA*，则将使用旧转换或新转换，但选择是不可预测的。除此以外，该行为是被明确定义的。

In a conventional TLB design, it is possible for multiple entries to match a single address if, for example, a page is upgraded to a superpage without first clearing the original non-leaf

PTE's valid bit and executing an SFENCE.VMA with rs1=x0*. In this case, a similar remark applies: it is unpredictable whether the old non-leaf PTE or the new leaf PTE is used, but the behavior is otherwise well defined.*

在传统 *TLB* 设计中，如果页面升级为超级页，而不首先清除原始非叶 *PTE* 的有效位并执行 rs1=x0 的 *SFENCE.VMA*，则可能会有多个条目匹配单个地址。在这种情况下，类似的话也适用：使用旧的非叶 *PTE* 还是新的叶 *PTE* 是不可预测的，但行为在其他方面定义得很好。

Another consequence of this specification is that it is generally unsafe to update a PTE using a set of stores of a width less than the width of the PTE, as it is legal for the implementation to read the PTE at any time, including when only some of the partial stores have taken effect.

该规范的另一个后果是，使用一组宽度小于 *PTE* 宽度的存储更新 *PTE* 通常是不安全的，因为实现在任何时候读取 *PTE* 都是合法的，包括只有部分存储生效时。

This specification permits the caching of PTEs whose V (Valid) bit is clear. Operating systems must be written to cope with this possibility, but implementers are reminded that eagerly caching invalid PTEs will reduce performance by causing additional page faults.

此规范允许缓存 *V*（有效）位被清除的 *PTE*。必须编写操作系统来处理这种可能性，但要提醒实现者，急切地缓存无效的 *PTE* 会导致额外的页面错误，从而降低性能。

Implementations must only perform implicit reads of the translation data structures pointed to by the current contents of the `satp` register or a subsequent valid (V=1) translation data structure entry, and must only raise exceptions for implicit accesses that are generated as a result of instruction execution, not those that are performed speculatively.

实现只能对 `satp` 寄存器的当前内容或后续有效（V=1）翻译数据结构条目所指向的翻译数据结构执行隐式读取，并且只能对指令执行而非推测性执行的隐式访问引发异常。

Changes to the `sstatus` fields SUM and MXR take effect immediately, without the need to execute an SFENCE.VMA instruction. Changing `satp`.MODE from Bare to other modes and vice versa also takes effect immediately, without the need to execute an SFENCE.VMA instruction. Likewise, changes to `satp`.ASID take effect immediately.

对 `sstatus` 字段 SUM 和 MXR 的更改将立即生效，而不需要执行 SFENCE.VMA 指令。将 `satp`.MODE 从 Bare 模式更改为其他模式，也会立即生效，而无需执行 SFENCE.VMA 指令，反之亦然。同样，对 `satp`.ASID 的改变也立即生效。

The following common situations typically require executing an SFENCE.VMA instruction:
以下常见的情况通常需要执行一个 *SFENCE.VMA* 指令：

- *When software recycles an ASID (i.e., reassociates it with a different page table), it should* first *change* `satp` *to point to the new page table using the recycled ASID,* then *execute SFENCE.VMA with* rs1=x0 *and* rs2 *set to the recycled ASID. Alternatively, software can execute the same SFENCE.VMA instruction while a different ASID is loaded into* `satp`*,*

provided the next time `satp` *is loaded with the recycled ASID, it is simultaneously loaded with the new page table.*

- *If the implementation does not provide ASIDs, or software chooses to always use ASID 0, then after every* `satp` *write, software should execute SFENCE.VMA with* rs1=`x0`*. In the common case that no global translations have been modified,* rs2 *should be set to a register other than* `x0` *but which contains the value zero, so that global translations are not flushed.*
- *If software modifies a non-leaf PTE, it should execute SFENCE.VMA with* rs1=`x0`*. If any PTE along the traversal path had its G bit set,* rs2 *must be* `x0`*; otherwise,* rs2 *should be set to the ASID for which the translation is being modified.*
- *If software modifies a leaf PTE, it should execute SFENCE.VMA with* rs1 *set to a virtual address within the page. If any PTE along the traversal path had its G bit set,* rs2 *must be* `x0`*; otherwise,* rs2 *should be set to the ASID for which the translation is being modified.*
- *For the special cases of increasing the permissions on a leaf PTE and changing an invalid PTE to a valid leaf, software may choose to execute the SFENCE.VMA lazily. After modifying the PTE but before executing SFENCE.VMA, either the new or old permissions will be used. In the latter case, a page-fault exception might occur, at which point software should execute SFENCE.VMA in accordance with the previous bullet point.*
- 当软件回收 *ASID*（即，将其与其他页表重新关联）时，应首先更改 `satp` 以使用回收的 *ASID* 指向新页表，然后执行 *SFENCE.VMA*，其 rs1=`x0`，rs2 设置为回收的 *ASID*。或者，软件可以在将不同的 *ASID* 加载到 `satp` 时，执行相同的 *SFENCE.VMA* 指令，前提是下次将 `satp` 加载到回收的 *ASID* 时，它将与新的页表同时加载。
- 如果实现不提供 *ASIDs*，或者软件选择始终使用 *ASID 0*，则在每次 `satp` 写入后，软件应执行 rs1=`x0` 的 *SFENCE.VMA*。在未修改全局转换的常见情况下，rs2 应设置为除 `x0` 以外的寄存器，该寄存器包含值零，以便不刷新全局转换。
- 如果软件修改了非叶 *PTE*，则应执行 rs1=`x0` 的 *SFENCE.VMA*。如果沿遍历路径的任何 *PTE* 设置了 *G* 位，则 rs2 必须为 `x0`；否则，rs2 应设置为要修改其转换的 *ASID*。
- 如果软件修改了叶 *PTE*，则应执行 *SFENCE.VMA*，并将 rs1 设置为页面内的虚拟地址。如果遍历路径上有任何 *PTE* 设置了 *G* 位，则 rs2 必须为 `x0`；否则，rs2 应设置为要修改其转换的 *ASID*。
- 对于增加叶 *PTE* 的权限并将无效 *PTE* 更改为有效叶的特殊情况，软件可能会选择延迟执行 *SFENCE.VMA* 。修改 *PTE* 后、执行 *SFENCE.VMA* 之前，将使用新权限或旧权限。在后一种情况下，可能会出现缺页异常，此时软件应根据前面的要点执行 *SFENCE.VMA*。

If a hart employs an address-translation cache, that cache must appear to be private to that hart. In particular, the meaning of an ASID is local to a hart; software may choose to use the same ASID to refer to different address spaces on different harts.

如果硬件线程使用地址转换缓存，则该缓存必须为该硬件线程的专用缓存。特别是，ASID 的含义是硬件线程特有的；软件可以选择使用相同的 ASID 来引用不同硬件线程上的不同地址空间。

A future extension could redefine ASIDs to be global across the SEE, enabling such options as shared translation caches and hardware support for broadcast TLB shootdown. However, as OSes have evolved to significantly reduce the scope of TLB shootdowns using novel ASID-management techniques, we expect the local-ASID scheme to remain attractive for its simplicity and possibly better scalability.

未来的扩展可能会将 ASID 重新定义为整个 SEE 的全局 ASID，从而实现共享转换缓存和广播 TLB 击落的硬件支持等选项。然而，随着操作系统的发展，使用新型 ASID 管理技术大大减少了 TLB 击落的可能，我们预计本 ASID 方案将因其简单性和更好的可扩展性而保持吸引力。

For implementations that make `satp`.MODE read-only zero (always Bare), attempts to execute an SFENCE.VMA instruction might raise an illegal instruction exception.

对于使 `satp`.MODE 为只读零（始终为 Bare）的具体实现，尝试执行 SFENCE.VMA 指令可能会引发非法指令异常。

4.3 Sv32: Page-Based 32-bit Virtual-Memory Systems Sv32：基于页面的 32 位虚拟内存系统

When Sv32 is written to the MODE field in the `satp` register (see Section 4.1.11), the supervisor operates in a 32-bit paged virtual-memory system. In this mode, supervisor and user virtual addresses are translated into supervisor physical addresses by traversing a radix-tree page table. Sv32 is supported when SXLEN=32 and is designed to include mechanisms sufficient for supporting modern Unix-based operating systems.

当 Sv32 写入 `satp` 寄存器中的 MODE 字段时（参见第4.1.11节），监管程序在 32 位分页虚拟内存系统中运行。在这种模式下，通过遍历基数树页表，将监管者和用户的虚拟地址转换为监管者的物理地址。当 SXLEN=32 时支持 Sv32，并且它被设计为包含足以支持现代基于 unix 的操作系统的机制。

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.

最初的 *RISC-V* 分页虚拟内存体系结构被设计为支持现有操作系统的直接实现。我们已经设计了页表布局以支持硬件页表查找。软件 *TLB* 再填充是高性能系统上的一个性能瓶颈，在解耦的专门协处理器中尤其麻烦。具体实现可以选择使用机器模式陷阱处理程序作为 *M* 模式的扩展来实现软件 *TLB* 重新填充。

Some ISAs architecturally expose virtually indexed, physically tagged *caches, in that accesses to the same physical address via different virtual addresses might not be coherent unless the virtual addresses lie within the same cache set. Implicitly, this specification does not permit such behavior to be architecturally exposed.*

一些 *ISA* 在体系结构上公开了虚拟索引、物理标记的缓存，因为通过不同的虚拟地址访问同一物理地址可能不一致，除非虚拟地址位于同一缓存集中。这也就是在暗示，此规范不允许在架构上公开此类行为。

4.3.1 Addressing and Memory Protection 寻址和内存保护

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 4.31. 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 3.7), before being directly converted to machine-level physical addresses. If necessary, supervisor-level physical addresses are zero-extended to the number of physical address bits found in the implementation.

Sv32 具体实现支持一个 32 位的虚拟地址空间，分为4 KiB页面。Sv32 虚拟地址被划分为虚拟页号（VPN）和页内偏移量，如图4.31所示。当在 `satp` 寄存器的 MODE 字段中选择 Sv32 虚拟内存模式时，监管者虚拟地址通过一个两级页表转换为监管者物理地址。20 位 VPN 被转换为 22 位物理页码（PPN），而 12 位页内偏移量则不被转换。然后，产生的监管者级物理地址在被直接转换为机器级物理地址之前，使用任何物理内存保护结构（第3.7节）进行检查。如果有必要，监管者级物理地址将零扩展到具体实现中找到的物理地址位数。

For example, consider an RV32 system supporting 34 bits of physical address. When the value of `satp`*.MODE is Sv32, a 34-bit physical address is produced directly, and therefore no zero-extension is needed. When the value of* `satp`*.MODE is Bare, the 32-bit virtual address is translated (unmodified) into a 32-bit physical address, and then that physical address is zero-extended into a 34-bit machine-level physical address.*

例如，考虑一个支持 *34* 位物理地址的 *RV32* 系统。当 `satp`.*MODE* 的值为 *Sv32* 时，可以直接生成 *34* 位物理地址，因此不需要零扩展。当 `satp`.*MODE* 的值为 *Bare* 时，*32* 位虚拟地址被转换（未修改）为 *32* 位物理地址，然后该物理地址被零扩展为 *34* 位机器级物理地址。

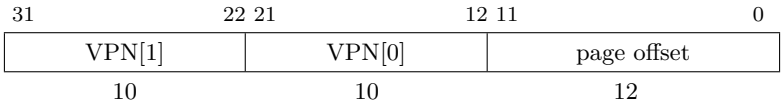


图 4.28: Sv32 virtual address.

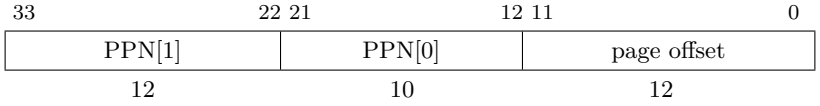


图 4.29: Sv32 physical address.

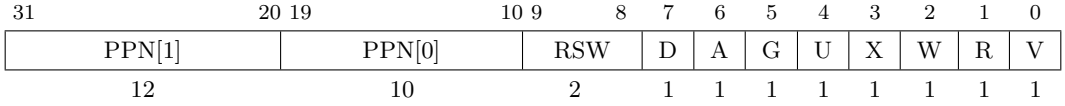


图 4.30: Sv32 page table entry.

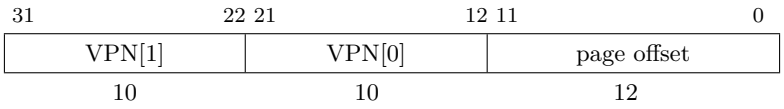


图 4.31: Sv32 虚拟地址

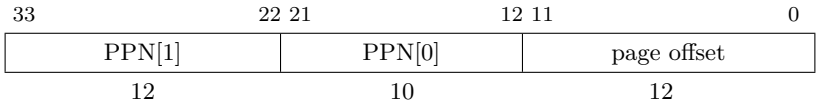


图 4.32: Sv32 物理地址

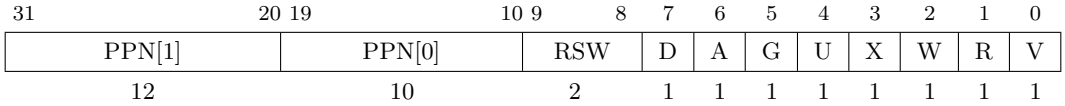


图 4.33: Sv32 页表项

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.

Sv32 页表由 2^{10} 个页表条目（PTEs）组成，每条有 4 个字节。页表与页面的大小完全一致，并且必须始终与页面边界对齐。根页表的物理页号存储在 `satp` 寄存器中。

The PTE format for Sv32 is shown in Figures 4.33. The V bit indicates whether the PTE is valid; if it is 0, all other bits in 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 4.10 summarizes the encoding of the permission bits.

Sv32 的 PTE 格式如图4.33所示。V 位表示 PTE 是否有效；如果是 0，PTE 中的所有其他位都不被关心，可以被软件自由使用。权限位 R、W 和 X 分别指示页面是否可读、可写和可执行。当这三个都为零时，PTE 是指向页表下一级的指针；否则，它是一个叶 PTE。可写的页面也必须被标记为可读的；相反的组合留作将来使用。表4.10总结了权限位的编码情况。

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.

表 4.9: Encoding of PTE R/W/X fields.

X	W	R	含义
0	0	0	下一级页表的指针
0	0	1	只读页
0	1	0	保留供将来使用
0	1	1	读取-写入页
1	0	0	仅执行页
1	0	1	读取-执行页
1	1	0	保留供将来使用
1	1	1	读取-写入-执行页

表 4.10: PTE R/W/X 域的编码

Attempting to fetch an instruction from a page that does not have execute permissions raises a fetch page-fault exception. Attempting to execute a load or load-reserved instruction whose effective

address lies within a page without read permissions raises a load page-fault exception. Attempting to execute a store, store-conditional, or AMO instruction whose effective address lies within a page without write permissions raises a store page-fault exception.

试图从没有执行权限的页面中获取指令，会引发取指缺页异常。尝试执行的加载或加载保留指令的有效地址位于没有读取权限的页面内，会引发加载缺页异常。试图执行有效地址位于没有写权限的页面内的存储、条件存储或 AMO 指令，会引发存储缺页异常。

AMOs never raise load page-fault exceptions. Since any unreadable page is also unwritable, attempting to perform an AMO on an unreadable page always raises a store page-fault exception.

AMOs 永远不会引发加载页面错误异常。因为任何不可读的页面也是不可写的，因此试图在不可读的页面上执行 AMO 总是会导致存储页面错误异常。

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. Irrespective of SUM, the supervisor may not execute code on pages with U=1.

U 位表示用户模式是否可以访问该页面。U 模式软件只能在 U=1 时访问该页面。如果设置了 `sstatus` 寄存器中的 SUM 位，监管者模式软件也可以访问具有 U=1 的页面。然而，监管者代码通常在 SUM 位清除的情况下运行，在这种情况下，监管者代码将在访问用户模式页面时出错。

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 4.1.1.2) and would require more encoding space in the PTE.

另一种 PTE 格式将支持监管者和用户的不同权限。我们省略了这个特性，因为它在很大程度上是 SUM 机制的冗余（参见第4.1.1.2节），并且在 PTE 中需要更多的编码空间。

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 a software bug that, after switching to an address space with a different non-global mapping for that address range, can unpredictably result in either mapping being used.

G 位指定全局映射。全局映射是存在于所有地址空间中的映射。对于非叶 PTE，全局设置意味着页表的后续级别中的所有映射都是全局的。请注意，未将全局映射标记为全局只会降低性能，而将非

全局映射标记成全局是一个软件错误，在切换到具有该地址范围的不同非全局映射的地址空间后，可能会不可预知地导致使用任一映射。

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.

全局映射无需冗余存储在多个 *ASID* 的地址转换缓存中。此外，当执行 rs2≠x0 的 *SFENCE.VMA* 指令时，不需要从本地地址转换缓存中刷新它们。

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

RSW 字段保留，供监管软件使用；具体实现应忽略此字段。

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.

每个叶 PTE 包含一个已访问（A）和脏（D）位。A 位表示自上次清除 A 位以来，虚拟页面已被读取、写入或取指。D 位表示自上次清除 D 位以来已写入虚拟页。

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

以下两种方案可以用来管理 A 位和 D 位：

- 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.
- 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(s) 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. Updates of the A bit may be performed as a result of speculation, but updates to the D bit must be exact (i.e., not speculative), and observed in program order by the local hart. Furthermore, the PTE update must appear in the global memory order no later than the explicit memory access, or any subsequent explicit memory access to that virtual page 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.

  The PTE update is not required to be atomic with respect to the explicit memory access that caused the update, and the sequence is interruptible. However, the hart must not perform the explicit memory access before the PTE update is globally visible.

- 当虚拟页面被访问并且 A 位为 0 时，或虚拟页面被写入且 D 位为 0 时，一个缺页异常将被引发。

- 当虚拟页面被访问并且 A 位为 0 时，或虚拟页面被写入且 D 位为 0 时，具体实现将在 PTE 中设置相应的一个或多个位。PTE 更新和对 PTE 的其他访问必须是原子性的，并且必须原子性地检查 PTE 是否有效并授予足够的权限。A 位的更新可能是推测执行的结果，但 D 位的更新必须是精确的（即，不是推测的），并由本地硬件线程按程序顺序监视。此外，PTE 更新必须以全局内存顺序出现，不得晚于显式内存访问或本地硬件线程对该虚拟页的任何后续显式内存存取。FENCE 指令提供的加载和存储排序以及原子指令上的获取/释放位也会根据远程硬件线程观察到的情况，对与这些加载和存储关联的 PTE 更新进行排序。

  对于引起更新的显式内存访问，PTE 更新不需要是原子的，执行序列是可中断的。然而，在 PTE 更新全局可见之前，硬件线程不能执行显式内存访问。

All harts in a system must employ the same PTE-update scheme as each other.

系统中的所有硬件线程必须使用相同的 PTE 更新方案。

Prior versions of this specification required PTE A bit updates to be exact, but allowing the A bit to be updated as a result of speculation simplifies the implementation of address translation prefetchers. System software typically uses the A bit as a page replacement policy hint, but does not require exactness for functional correctness. On the other hand, D bit updates are still required to be exact and performed in program order, as the D bit affects the functional correctness of page eviction.

该规范的早期版本要求精确更新 PTE 的 A 位，但允许 A 位因推测而更新，因而简化了地址转换预取器的实现。系统软件通常使用 A 位作为页面替换策略的提示，但不需要精确的功能正确性。另一方面，D 位的更新仍然需要精确并按程序顺序执行，因为 D 位影响页面收回的功能正确性。

Implementations are of course still permitted to perform both A and D bit updates only in an exact manner.

当然，仍然允许具体实现以精确的方式执行 A 位和 D 位更新。

In both cases, requiring atomicity ensures that the PTE update will not be interrupted by other intervening writes to the page table, as such interruptions could lead to A/D bits being set on PTEs that have been reused for other purposes, on memory that has been reclaimed for other purposes, and so on. Simple implementations may instead generate page-fault exceptions.

在这两种情况下，要求原子性可以确保 PTE 更新不会被其他干预写入页表的操作中断，因为这样的中断可能会导致 A/D 位被设置在为其他目的使用的 PTE 上，或设置在为其它目的回收的内存上，以此类推。一种简单的实现可能会生成页错误异常。

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.

A 位和 D 位永远不会被具体实现清除。如果监管者软件不依赖于已访问以及脏位，例如，如果它不将内存页交换到辅助存储器，或者如果这些页面正用于映射 I/O 空间，则应始终在 PTE 中将它们设置为 1，以提高性能。

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.

任何级别的 PTE 都可以是叶 PTE，因此除了 4 个 KiB 页面外，Sv32 还支持 4 MiB 巨型页面。一个巨型页面必须与一个 4 MiB 边界在物理上和虚拟上对齐；如果物理地址未充分对齐，则会引发页面错误异常。

For non-leaf PTEs, the D, A, and U bits are reserved for future standard use. Until their use is defined by a standard extension, they must be cleared by software for forward compatibility.

对于非叶 PTE，D、A 和 U 位保留供未来标准使用。在标准扩展定义其使用之前，必须通过软件将它们清 0 以实现向前兼容性。

For implementations with both page-based virtual memory and the "A" standard extension, the LR/SC reservation set must lie completely within a single base page (i.e., a naturally aligned 4 KiB region).

对于具有基于页面的虚拟内存和"A"标准扩展的具体实现，LR/SC 保留集必须完全位于单个基本页面内（即自然对齐的4 KiB区域）。

4.3.2 Virtual Address Translation Process 虚拟地址转换过程

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

一个虚拟地址 *va* 被转换为一个物理地址 *pa* 的过程如下所示：

1. Let a be `satp`.$ppn \times$ PAGESIZE, and let $i =$ LEVELS $- 1$. (For Sv32, PAGESIZE=2^{12} and LEVELS=2.) The `satp` register must be *active*, i.e., the effective privilege mode must be S-mode or U-mode.

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-fault exception corresponding to the original access type.

3. If $pte.v = 0$, or if $pte.r = 0$ and $pte.w = 1$, or if any bits or encodings that are reserved for future standard use are set within pte, stop and raise a page-fault exception corresponding to the original access type.

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 corresponding to the original access type. Otherwise, let $a = pte.ppn \times \text{PAGESIZE}$ 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 corresponding to the original access type.

6. If $i > 0$ and $pte.ppn[i-1:0] \neq 0$, this is a misaligned superpage; stop and raise a page-fault exception corresponding to the original access type.

7. If $pte.a = 0$, or if the original memory access is a store and $pte.d = 0$, either raise a page-fault exception corresponding to the original access type, or:

   - If a store to pte would violate a PMA or PMP check, raise an access-fault exception corresponding to the original access type.
   - Perform the following steps atomically:
     - Compare pte to the value of the PTE at address $a + va.vpn[i] \times \text{PTESIZE}$.
     - If the values match, set $pte.a$ to 1 and, if the original memory access is a store, also set $pte.d$ to 1.
     - If the comparison fails, return to step 2

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 $pa.ppn[i-1:0] = va.vpn[i-1:0]$.
   - $pa.ppn[\text{LEVELS}-1:i] = pte.ppn[\text{LEVELS}-1:i]$.

1. 设 a 为 `satp`.$ppn \times$ PAGESIZE，设 $i =$ LEVELS $- 1$ （对于 Sv32，PAGESIZE=2^{12}，LEVELS=2）。`satp` 寄存器必须处于*活动*状态，即有效特权模式必须为 S 模式或 U 模式。

2. 设 pte 为地址 $a + va.vpn[i] \times$ PTESIZE 处 PTE 的值（对于 Sv32，PTESIZE=4）。如果访问 pte 违反了 PMA 或 PMP 检查，则引发与原始访问类型对应的访问错误异常。

3. 如果 $pte.v = 0$，或者 $pte.r = 0$ 且 $pte.w = 1$，或者如果在 pte 中设置了被保留以供将来标准使用的任何位或编码，则停止并引发与原始访问类型对应的页面错误异常。

4. 否则，PTE 有效。如果 $pte.r = 1$ 或 $pte.x = 1$，转至步骤 5；否则，此 pte 是指向页表下一级的指针。使 $i = i - 1$. 如果 $i < 0$，则停止并引发与原始访问类型对应的页面错误异常；否则，使 $a = pte.ppn \times$ PAGESIZE 并转至步骤 2。

5. 已找到叶 PTE。在给定当前特权模式和 `mstatus` 寄存器的 SUM 和 MXR 字段值的情况下，确定 $pte.r$、$pte.w$、$pte.x$ 和 $pte.u$ 位是否允许请求的内存访问。如果不允许，则停止并引发与原始访问类型对应的页面错误异常。

6. 如果 $i > 0$ 且 $pte.ppn[i-1:0] \neq 0$，这是未对齐的超级页；停止并引发与原始访问类型对应的页面错误异常。

7. 7. 如果 $pte.a = 0$，或者如果原始内存访问是存储且 $pte.d = 0$，则引发与原始访问类型对应的页面错误异常，或者：

   - 如果 pte 的存储违反了 PMA 或 PMP 检查，则引发与原始访问类型对应的访问错误异常。
   - 自动执行以下步骤：
     - 将 pte 与地址 $a + va.vpn[i] \times$ PTESIZE 处的 PTE 值进行对比。
     - 如果值匹配，则将 $pte.a$ 设置为 1；如果原始内存访问是存储，则也将 $pte.d$ 设置为 1。
     - 如果对比不匹配，返回步骤 2。

8. 8. 翻译成功。转换后的物理地址如下：T

   - $pa.pgoff = va.pgoff$。
   - 如果 $i > 0$，则这是一个超级页转换，$pa.ppn[i-1:0] = va.vpn[i-1:0]$。
   - $pa.ppn[\text{LEVELS} - 1 : i] = pte.ppn[\text{LEVELS} - 1 : i]$。

All implicit accesses to the address-translation data structures in this algorithm are performed using width PTESIZE.

该算法中所有对地址转换数据结构的隐式访问都是使用宽度 PTESIZE 执行的。

This implies, for example, that an Sv48 implementation may not use two separate 4B reads to non-atomically access a single 8B PTE, and that A/D bit updates performed by the implementation are treated as atomically updating the entire PTE, rather than just the A and/or D bit alone (even though the PTE value does not otherwise change).

这意味着，例如，*Sv48* 实现可能不会使用两次单独的 *4B* 读取来非原子化地访问单个 *8B PTE*，并且该具体实现执行的 *A/D* 位更新被视为原子化地更新整个 *PTE*，而不仅仅是 *a* 位和/或 *D* 位（即使 *PTE* 值没有改变）。

The results of implicit address-translation reads in step 2 may be held in a read-only, incoherent *address-translation cache* but not shared with other harts. The address-translation cache may hold an arbitrary number of entries, including an arbitrary number of entries for the same address and ASID. Entries in the address-translation cache may then satisfy subsequent step 2 reads if the ASID associated with the entry matches the ASID loaded in step 0 or if the entry is associated with a *global* mapping. To ensure that implicit reads observe writes to the same memory locations, an SFENCE.VMA instruction must be executed after the writes to flush the relevant cached translations.

步骤 2 中隐式地址转换读取的结果可能保存在一个只读的、非相干**地址转换缓存**中，但不会与其他硬件线程共享。地址转换缓存可以保存任意数量的条目，包括任意数量的同一地址和 ASID 的条目。如果与条目相关联的 ASID 与步骤 0 中加载的 ASID 匹配，或者如果条目与**全局**映射相关联，则地址转换缓存中的条目可以满足后续步骤 2 的读取。为了确保隐式读取能观察到对相同内存位置的写入，必须在写入后执行 SFENCE.VMA 指令，以刷新相关的转换缓存。

The address-translation cache cannot be used in step 7; accessed and dirty bits may only be updated in memory directly.

步骤 7 中无法使用地址转换缓存；访问位和脏位只能直接在内存中更新。

It is permitted for multiple address-translation cache entries to co-exist for the same address. This represents the fact that in a conventional TLB hierarchy, it is possible for multiple entries to match a single address if, for example, a page is upgraded to a superpage without first clearing the original non-leaf PTE's valid bit and executing an SFENCE.VMA with rs1=x0*, or if multiple TLBs exist in parallel at a given level of the hierarchy. In this case, just as if an SFENCE.VMA is not executed between a write to the memory-management tables and subsequent implicit read of the same address: it is unpredictable whether the old non-leaf PTE or the new leaf PTE is used, but the behavior is otherwise well defined.*

允许同一地址的多个地址转换缓存项共存。这表明，在传统 *TLB* 层次结构中，如果在不清除原始非叶 *PTE* 的有效位并执行 rs1=x0 的 *SFENCE.VMA* 的情况下，将一个页面升级为一个超级页，或者如果在层次结构的给定级别上并行存在多个 *TLB*，则可能有多个条目匹配单个地址。在这种情况下，就像没有在内存管理表的写入和同一地址的后续隐式读取之间执行 *SFENCE.VMA* 一样：使用旧的非叶 *PTE* 还是新的叶 *PTE* 是不可预测的，但在其他情况下，该行为是被明确定义的。

Implementations may also execute the address-translation algorithm speculatively at any time, for any virtual address, as long as `satp` is active (as defined in Section 4.1.11). Such speculative executions have the effect of pre-populating the address-translation cache.

只要 `satp` 处于活动状态（如第4.1.11节所定义），具体实现还可以在任何时候对任何虚拟地址推测性地执行地址转换算法。此类推测性执行具有预填充地址转换缓存的效果。

Speculative executions of the address-translation algorithm behave as non-speculative executions of the algorithm do, except that they must not set the dirty bit for a PTE, they must not trigger an exception, and they must not create address-translation cache entries if those entries would have been invalidated by any SFENCE.VMA instruction executed by the hart since the speculative execution of the algorithm began.

地址转换算法的推测性执行与算法的非推测性执行的行为相同，但它们禁止为 PTE 设置脏位，也禁止触发异常，并且，如果自算法的推测执行开始以来，硬件线程执行的任何 SFENCE.VMA 指令可能已经使这些条目无效了，则它们不得创建地址转换缓存条目。

For instance, it is illegal for both non-speculative and speculative executions of the translation algorithm to begin, read the level 2 page table, pause while the hart executes an SFENCE.VMA with rs1=rs2=`x0`*, then resume using the now-stale level 2 PTE, as subsequent implicit reads could populate the address-translation cache with stale PTEs.*

例如，开始执行转换算法的非推测执行和推测执行，读取 *2* 级页表，在硬件线程执行 rs1=rs2=`x0` 的 *SFENCE.VMA* 时暂停，然后继续使用已经过时的 *2* 级 *PTE*，这些行为是非法的，因为后续的隐式读取可能会用过时的 *PTE* 填充地址转换缓存。

In many implementations, an SFENCE.VMA instruction with rs1=`x0` *will therefore either terminate all previously-launched speculative executions of the address-translation algorithm (for the specified ASID, if applicable), or simply wait for them to complete (in which case any address-translation cache entries created will be invalidated by the SFENCE.VMA as appropriate). Likewise, an SFENCE.VMA instruction with* rs1≠`x0` *generally must either ensure that previously-launched speculative executions of the address-translation algorithm (for the specified ASID, if applicable) are prevented from creating new address-translation cache entries mapping leaf PTEs, or wait for them to complete.*

因此，在许多实现中，rs1=`x0` 的 *SFENCE.VMA* 指令将终止先前启动的地址转换算法的所有推测执行（如果适用，针对指定的 *ASID*），或只是等待它们完成（在这种情况下创建的任何地址转换缓存项将由 *SFENCE.VMA* 视情况而定为无效）。同样，rs1≠`x0` 的 *SFENCE.VMA* 指令通常必须确保防止先前启动的地址转换算法的推测执行（如果适用，针对指定的 *ASID*）创建映射叶 *PTE* 的新地址转换缓存项，或者等待它们完成。

A consequence of implementations being permitted to read the translation data structures arbitrarily early and speculatively is that at any time, all page table entries reachable by executing the algorithm may be loaded into the address-translation cache.

允许具体实现任意提前和推测性地读取转换数据结构的结果是，在任何时候，通过执行算法可以访问的所有页表条目都可以加载到地址转换缓存中。

Although it would be uncommon to place page tables in non-idempotent memory, there is no explicit prohibition against doing so. Since the algorithm may only touch page tables reachable from the root page table indicated in `satp`*, the range of addresses that an implementation's page table walker will touch is fully under supervisor control.*

虽然在非幂等内存中放置页表并不常见，但并没有明确禁止这样做。由于该算法可能只触及从 `satp` 中指示的根页表中可以访问的页表，因此具体实现的页表遍历器能触及的地址范围完全由监管者控制。

The algorithm does not admit the possibility of ignoring high-order PPN bits for implementations with narrower physical addresses.

该算法不允许在物理地址较窄的实现中忽略 *PPN* 的高位。

4.4 Sv39: Page-Based 39-bit Virtual-Memory System Sv39：基于页面的 39 位虚拟内存系统

This section describes a simple paged virtual-memory system for SXLEN=64, 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.

本节介绍一个简单的 SXLEN=64 分页虚拟内存系统，它支持 39 位虚拟地址空间。Sv39 的设计遵循 Sv32 的总体方案，本节仅详细说明方案之间的差异。

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. Sv57 increases the virtual address space, page table capacity requirement, and translation latency even further.

我们为 *RV64* 制定了多个虚拟内存系统，以缓解提供大地址空间和最小化地址转换成本之间的紧张关系。对于许多系统，*512 GiB*的虚拟地址空间充足，所以 *Sv39* 就足够了。*Sv48* 将虚拟地址空间增加到 *256 256 TiB*，但同时增加了专用于页表的物理内存容量，遍历页表的延迟，以及存储虚拟地址的硬件结构的大小。*Sv57* 系列进一步增加了虚拟地址空间、页表容量需求以及转换延迟。

4.4.1 Addressing and Memory Protection 寻址和内存保护

Sv39 implementations support a 39-bit virtual address space, divided into 4 KiB pages. An Sv39 address is partitioned as shown in Figure 4.37. Instruction fetch addresses and load and store effective addresses, which are 64 bits, must have bits 63–39 all equal to bit 38, or else a page-fault exception will occur. The 27-bit VPN is translated into a 44-bit PPN via a three-level page table, while the 12-bit page offset is untranslated.

Sv39 具体实现支持 39 位虚拟地址空间，分为4 KiB页。Sv39 地址的划分如图4.37所示。指令获取地址以及加载和存储有效地址（64 位）的位 63–39 必须全部等于位 38，否则将发生页错误异常。27 位 VPN 通过三级页表转换为 44 位 PPN，而 12 位页内偏移量未转换。

When mapping between narrower and wider addresses, RISC-V zero-extends a narrower physical address to a wider size. The mapping between 64-bit virtual addresses and the 39-bit usable address space of Sv39 is not based on zero-extension but instead follows an entrenched convention that allows an OS to use one or a few of the most-significant bits of a full-size (64-bit) virtual address to quickly distinguish user and supervisor address regions.

当在宽和窄的地址之间映射时，RISC-V 将更窄的物理地址零扩展到宽地址的大小。64 位虚拟地址和 Sv39 的 39 位可用地址空间之间的映射不是基于零扩展，而是遵循一种根深蒂固的约定，即允许操作系统使用全尺寸（64 位）虚拟地址中的一个或几个最重要的位来快速区分用户和监管者地址区域。

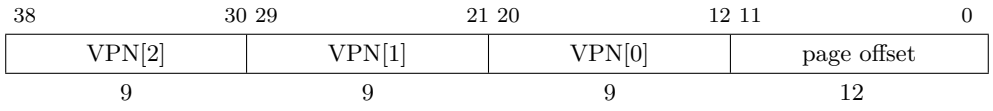


图 4.34: Sv39 virtual address.

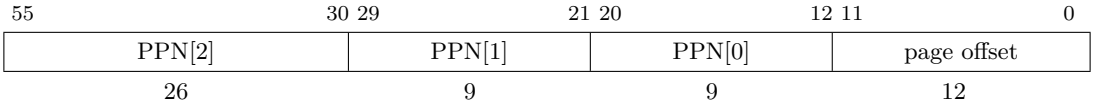


图 4.35: Sv39 physical address.

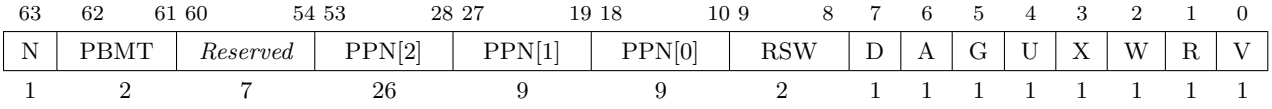


图 4.36: Sv39 page table entry.

Sv39 page tables contain 2^9 page table entries (PTEs), eight bytes each. 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's PPN field.

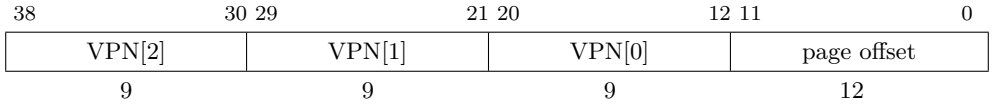


图 4.37: Sv39 虚拟地址

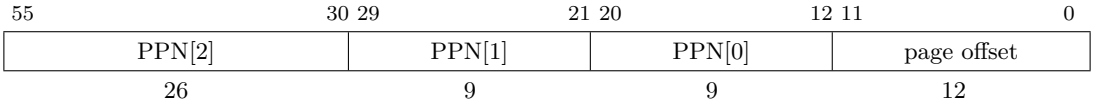


图 4.38: Sv39 物理地址

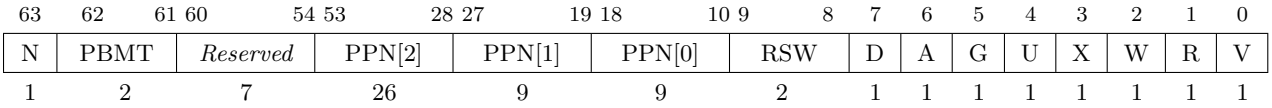


图 4.39: Sv39 页表项

Sv39 页表包含 2^9 个页表项（PTE），每项 8 个字节。一张页表正好是一张页面的大小，页表必须始终与页面边界对齐。根页表的物理页号存储在 `satp` 寄存器的 PPN 字段中。

The PTE format for Sv39 is shown in Figure 4.39. Bits 9–0 have the same meaning as for Sv32. Bit 63 is reserved for use by the Svnapot extension in Chapter 五. If Svnapot is not implemented, bit 63 remains reserved and must be zeroed by software for forward compatibility, or else a page-fault exception is raised. Bits 62–61 are reserved for use by the Svpbmt extension in Chapter 六. If Svpbmt is not implemented, bits 62–61 remain reserved and must be zeroed by software for forward compatibility, or else a page-fault exception is raised. Bits 60–54 are reserved for future standard use and, until their use is defined by some standard extension, must be zeroed by software for forward compatibility. If any of these bits are set, a page-fault exception is raised.

Sv39 的 PTE 格式如图4.39所示。位 9–0 与在 Sv32 中的含义相同。第五章中，位 63 保留给 Svnapot 扩展使用。如果未实现 Svnapot，则位 63 保留，并且必须由软件归零以实现前向兼容性，否则会引发页面错误异常。第六章中，位 62–61 保留给 Svpbmt 扩展使用。如果未实现 Svpbmt，则位 62–61 保留，并且必须由软件归零以实现前向兼容性，否则会引发页面错误异常。位 60–54 保留供将来的标准使用，在由某些标准扩展定义其使用之前，必须由软件归零，以实现前向兼容性。如果设置了这些位中的任何一位，则会引发页面错误异常。

We reserved several PTE bits for a possible extension that improves support for sparse address spaces by allowing page-table levels to be skipped, reducing memory usage and TLB refill latency. These reserved bits may also be used to facilitate research experimentation. The cost is reducing the physical address space, but 64 PiB is presently ample. When it no longer suffices, the reserved bits that remain unallocated could be used to expand the physical address space.

我们为可能的扩展保留了几个 *PTE* 位，这些扩展通过允许跳过页表级别，减少内存使用量和 *TLB* 重新填充延迟，从而改进了对稀疏地址空间的支持。这些保留位也可用于促进研究实验。成本是减少物理地址空间，但 *64 64 PiB*目前已足够。当它不再足够时，保留的未分配位可用于扩展物理地址空间。

Any level of PTE may be a leaf PTE, so in addition to 4 KiB pages, Sv39 supports 2 MiB *megapages* and 1 GiB *gigapages*, each of which must be virtually and physically aligned to a boundary equal to its size. A page-fault exception is raised if the physical address is insufficiently aligned.

任何级别的 PTE 都可以是叶 PTE，因此除了4 KiB页面外，Sv39 还支持2 MiB巨型页面 和1 GiB千兆页面，每个页面都必须在物理上和虚拟上与和它大小相等的边界对齐。如果物理地址未充分对齐，则会引发页面错误异常。

The algorithm for virtual-to-physical address translation is the same as in Section 4.3.2, except LEVELS equals 3 and PTESIZE equals 8.

虚拟地址到物理地址转换的算法与第4.3.2节中的相同，但 LEVELS 等于 3，PTESIZE 等于 8。

4.5 Sv48: Page-Based 48-bit Virtual-Memory System Sv48：基于页面的 48 位虚拟内存系统

This section describes a simple paged virtual-memory system for SXLEN=64, which supports 48-bit virtual address spaces. Sv48 is intended for systems for which a 39-bit virtual address space is insufficient. It closely follows the design of Sv39, simply adding an additional level of page table, and so this chapter only details the differences between the two schemes.

本节介绍一个简单的 SXLEN=64 分页虚拟内存系统，它支持 39 位虚拟地址空间。Sv48 适用于 39 位虚拟地址空间不足的系统。它紧跟 Sv39 的设计，只是添加了一个额外的页表级别，因此本章只详细介绍了两种方案之间的差异。

Implementations that support Sv48 must also support Sv39.

支持 Sv48 的具体实现必须能够支持 Sv39。

Systems that support Sv48 can also support Sv39 at essentially no cost, and so should do so to maintain compatibility with supervisor software that assumes Sv39.

支持 *Sv48* 的系统也可以基本上没有代价地支持 *Sv39*，因此应该这样做以保持与 *Sv39* 的管理软件的兼容性。

4.5.1 Addressing and Memory Protection 寻址和内存保护

Sv48 implementations support a 48-bit virtual address space, divided into 4 KiB pages. An Sv48 address is partitioned as shown in Figure 4.43. Instruction fetch addresses and load and store effective addresses, which are 64 bits, must have bits 63–48 all equal to bit 47, or else a page-fault exception will occur. The 36-bit VPN is translated into a 44-bit PPN via a four-level page table, while the 12-bit page offset is untranslated.

Sv48 具体实现支持 48 位虚拟地址空间，分为4 KiB页。Sv48 地址的划分如图4.43所示。指令获取地址以及加载和存储有效地址（64 位）的位 63–48 必须全部等于位 47，否则将发生页错误异常。36 位 VPN 通过四级页表转换为 44 位 PPN，而 12 位页内偏移量未转换。

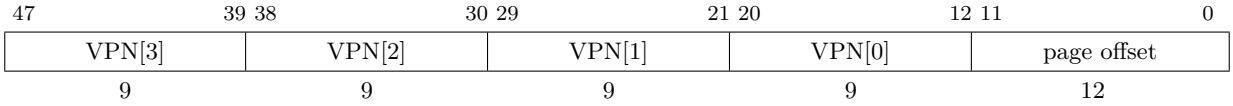


图 4.40: Sv48 virtual address.

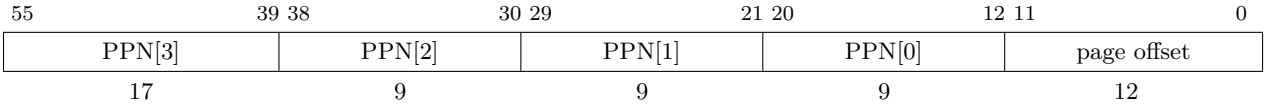


图 4.41: Sv48 physical address.

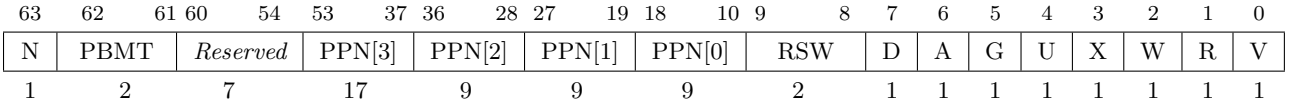


图 4.42: Sv48 page table entry.

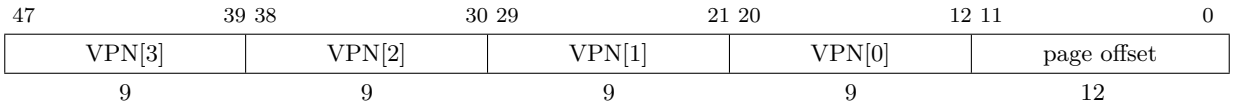


图 4.43: Sv48 虚拟地址

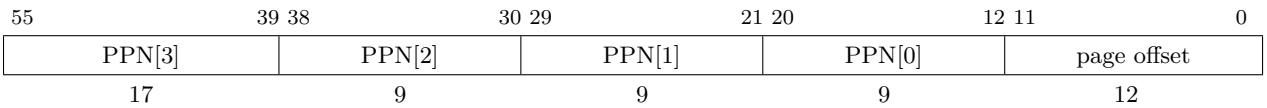


图 4.44: Sv48 物理地址

The PTE format for Sv48 is shown in Figure 4.45. Bits 63–54 and 9–0 have the same meaning as for Sv39. Any level of PTE may be a leaf PTE, so in addition to 4 KiB pages, Sv48 supports 2 MiB

63	62–61	60–54	53–37	36–28	27–19	18–10	9–8	7	6	5	4	3	2	1	0
N	PBMT	*Reserved*	PPN[3]	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
1	2	7	17	9	9	9	2	1	1	1	1	1	1	1	1

图 4.45: Sv48 页表项

megapages, 1 GiB *gigapages*, and 512 GiB *terapages*, each of which must be virtually and physically aligned to a boundary equal to its size. A page-fault exception is raised if the physical address is insufficiently aligned.

Sv48 的 PTE 格式如图4.45所示。位 63–54 和 9–0 与在 Sv39 中的含义相同。任何级别的 PTE 都可以是叶 PTE，因此，除了4 KiB页外，Sv48 还支持2 MiB巨型页、1 GiB千兆页和512 GiB 万亿页，每个页面都必须在虚拟和物理上与和它大小相等的边界对齐。如果物理地址未充分对齐，则会引发页面错误异常。

The algorithm for virtual-to-physical address translation is the same as in Section 4.3.2, except LEVELS equals 4 and PTESIZE equals 8.

虚拟地址到物理地址转换的算法与第4.3.2节中的相同，但 LEVELS 等于 4，PTESIZE 等于 8。

4.6 Sv57: Page-Based 57-bit Virtual-Memory System Sv57：基于页面的 57 位虚拟内存系统

This section describes a simple paged virtual-memory system designed for RV64 systems, which supports 57-bit virtual address spaces. Sv57 is intended for systems for which a 48-bit virtual address space is insufficient. It closely follows the design of Sv48, simply adding an additional level of page table, and so this chapter only details the differences between the two schemes.

本节介绍一个为 RV64 系统设计的简单分页虚拟内存系统，该系统支持 57 位虚拟地址空间。Sv57 适用于 48 位虚拟地址空间不足的系统。它紧跟 Sv48 的设计，只是添加了一个额外的页表级别，因此本章只详细介绍了两种方案之间的差异。

Implementations that support Sv57 must also support Sv48.

支持 Sv57 的具体实现必须能够支持 Sv48。

Systems that support Sv57 can also support Sv48 at essentially no cost, and so should do so to maintain compatibility with supervisor software that assumes Sv48.

支持 *Sv57* 的系统也可以基本上没有代价地支持 *Sv48*，因此应该这样做以保持与 *Sv48* 的管理软件的兼容性。

4.6.1 Addressing and Memory Protection 寻址和内存保护

Sv57 implementations support a 57-bit virtual address space, divided into 4 KiB pages. An Sv57 address is partitioned as shown in Figure 4.49. Instruction fetch addresses and load and store effective addresses, which are 64 bits, must have bits 63–57 all equal to bit 56, or else a page-fault exception will occur. The 45-bit VPN is translated into a 44-bit PPN via a five-level page table, while the 12-bit page offset is untranslated.

Sv57 具体实现支持 57 位虚拟地址空间，分为4 KiB页。Sv57 地址的划分如图4.49所示。指令获取地址以及加载和存储有效地址（64 位）的位 63–57 必须全部等于位 56，否则将发生页错误异常。45 位 VPN 通过五级页表转换为 44 位 PPN，而 12 位页内偏移量未转换。

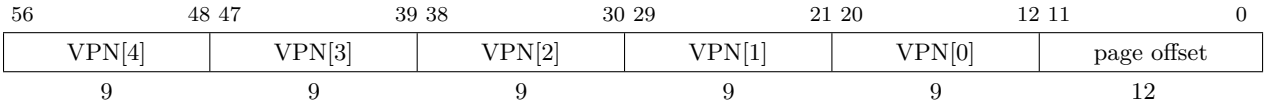


图 4.46: Sv57 virtual address.

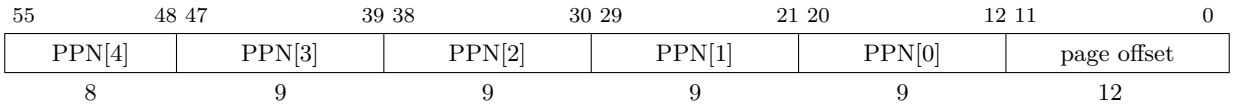


图 4.47: Sv57 physical address.

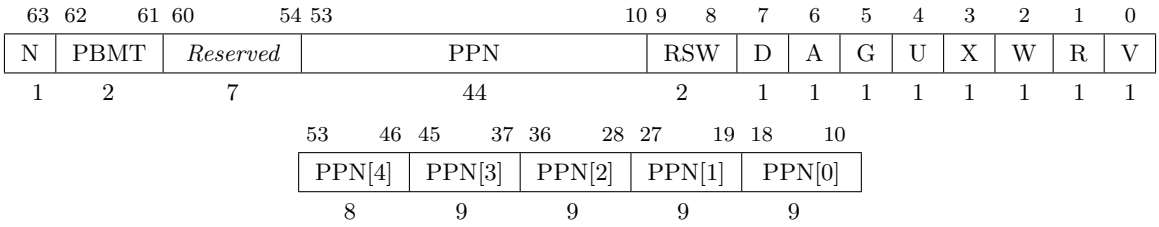


图 4.48: Sv57 page table entry.

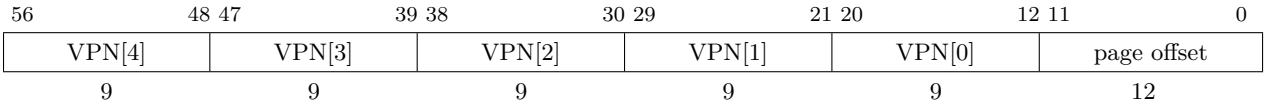


图 4.49: Sv57 虚拟地址

The PTE format for Sv57 is shown in Figure 4.51. Bits 63–54 and 9–0 have the same meaning as for Sv39. Any level of PTE may be a leaf PTE, so in addition to 4 KiB pages, Sv57 supports

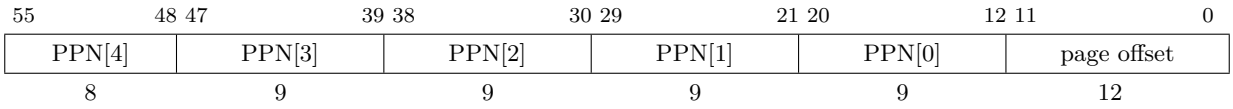


图 4.50: Sv57 物理地址

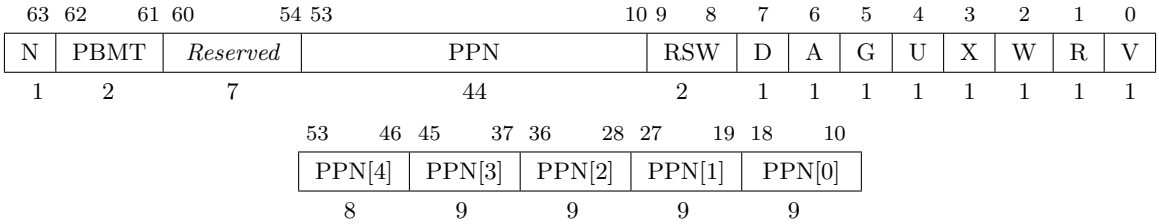


图 4.51: Sv57 页表项

2 MiB *megapages*, 1 GiB *gigapages*, 512 GiB *terapages*, and 256 TiB *petapages*, each of which must be virtually and physically aligned to a boundary equal to its size. A page-fault exception is raised if the physical address is insufficiently aligned.

Sv57 的 PTE 格式如图4.51所示。位 63–54 和 9–0 与在 Sv39 中的含义相同。任何级别的 PTE 都可以是叶 PTE，因此，除了4 KiB页外，Sv57 还支持2 MiB 巨型页、1 GiB 千兆页、512 GiB 万亿页和256 TiB *PETA* 页，每个页面都必须在虚拟和物理上与和它大小相等的边界对齐。如果物理地址未充分对齐，则会引发页错误异常。

The algorithm for virtual-to-physical address translation is the same as in Section 4.3.2, except LEVELS equals 5 and PTESIZE equals 8.

虚拟地址到物理地址转换的算法与第4.3.2节中的相同，但 LEVELS 等于 5，PTESIZE 等于 8。

第五章 “Svnapot” NAPOT 翻译连续性的标准扩展，版本 1.0

在 Sv39、Sv48 和 Sv57 中，当 PTE 具有 N=1 时，PTE 表示作为连续虚拟到物理转换范围的一部分的转换，PTE 位 5-0 的值相同。这样的范围必须具有自然对齐的 2 次方 (NAPOT) 粒度，大于基本页面大小。

Svnapot 扩展依赖于 Sv39。

i	*pte.ppn*[*i*]	Description	*pte.napot_bits*
0	`x xxxx xxx1`	*Reserved*	–
0	`x xxxx xx1x`	*Reserved*	–
0	`x xxxx x1xx`	*Reserved*	–
0	`x xxxx 1000`	64 KiB contiguous region	4
0	`x xxxx 0xxx`	*Reserved*	–
≥ 1	`x xxxx xxxx`	*Reserved*	–

表 5.1: Page table entry encodings when *pte*.N=1

在 Section 4.3.2的地址转换算法中，NAPOT PTEs 的行为与非 NAPOT PTEs 相同，除了:

- 如果 *pte* 中的编码根据 Table 5.1是有效的，那么对 NAPOT PTE 的隐式读取不会返回 *pte* 的原始值，而是返回 *pte* 的副本，其中 $pte.ppn[pte.napot\_bits-1:0]$ 被 $vpn[i][pte.napot\_bits-1:0]$ 取代。如果根据表 5.1保留 *pte* 中的编码，则必须引发页面错误异常。
- 隐式读取 NAPOT 页表项可能会创建地址转换缓存项，将 $a + va.vpn[j] \times$ PTESIZE 映射到 *pte* 的一个副本，其中 $pte.ppn[pte.napot\_bits-1:0]$ 被 $vpn[0][pte.napot\_bits-1:0]$ 取代，对于任何或所有 j，$j[8:napot\_bits] = i[8:napot\_bits]$ 中的所有地址空间标识为步骤 0 加载的 *satp*。

使用 *NAPOT PTE* 的原因是，它可以作为一个或多个表示连续区域的项缓存到 *TLB* 中，就像一个翻译覆盖的单个 *(*大*)* 页一样。在某些场景下，这种压缩可以帮助减轻 *TLB* 压力。该编码被设计成符合预先存在的 *Sv39*、*Sv48* 和 *Sv57 PTE* 格式，从而不会破坏选择不实现该方案的现有实现或设计。此外，还对地址转换算法的定义进行了简化。

地址转换缓存抽象捕获了由于创建覆盖整个 *NAPOT* 区域的单个 *TLB* 条目而导致的行为。它还被设计成与支持 *NAPOT PTEs* 的实现一致，将 *NAPOT* 区域分割为 *TLB* 项，覆盖任何较小的 *2* 次方区域大小。例如，一个 *64 KiB* 的 *NAPOT PTE* 可能会触发 *16* 个标准的 *4 KiB TLB* 条目的创建，所有条目的内容都是从 *NAPOT PTE* 生成的 *(*即使其他 *4 KiB* 区域的 *PTE* 具有不同的内容*)*。

在典型的使用场景中，相同区域的 *NAPOT PTEs* 将具有相同的属性、相同的 *PPNs* 和位 *5-0* 的相同值。*RSW* 仍保留为主管软件控制。操作系统和/或管理程序负责配置页表，使 *NAPOT pte* 和其他重叠相同地址范围的 *NAPOT* 或非 *NAPOT PTEs* 之间不存在不一致。如果需要进行更新，操作系统通常应该首先标记所有的 *PTE* 无效，然后发出覆盖范围内所有 *4 KiB* 区域的 *SFENCE.VMA* 指令 *(*要么通过带有 rs1=x0 的单个 *SFENCE.VMA*，要么通过带有 rs1≠x0 的多个 *SFENCE.VMA* 指令*)*，然后更新 *PTE*，如 **??**小节所述，除非已知的任何不一致都是无害的。

如果一个实现选择使用 *NAPOT PTE(*或其缓存版本*)*，它可能根本不会查询 *Section 4.3.2*中算法直接指定的 *PTE*。因此，即使在典型的用例中，在相同地址范围的所有映射中，*D* 位和 *A* 位也可能不相同。操作系统必须查询一个页面的所有 *NAPOT* 别名，以确定该页面是否已被访问和/或是脏的。如果操作系统手动设置了一个页面的 *A* 位和/或 *D* 位，建议操作系统也相应地设置其他 *NAPOT* 别名的 *A* 位和/或 *D* 位，以避免不必要的陷阱。

与普通 *PTEs* 一样，*TLBs* 允许缓存 *V (Valid)* 位为空的 *NAPOT PTEs*。

根据需要，将来可以将 *NAPOT* 方案扩展到其他中间页大小和/或页表的其他级别。编码的设计是为了适应需要出现的其他 *NAPOT* 大小。例如*:*

i	*pte.ppn*[*i*]	Description	*pte.napot_bits*
0	x xxxx xxx1	8 KiB contiguous region	1
0	x xxxx xx10	16 KiB contiguous region	2
0	x xxxx x100	32 KiB contiguous region	3
0	x xxxx 1000	64 KiB contiguous region	4
0	x xxx1 0000	128 KiB contiguous region	5
...	...	...	...
1	x xxxx xxx1	4 MiB contiguous region	1
1	x xxxx xx10	8 MiB contiguous region	2
...	...	...	...

在这种情况下，实现可能支持，也可能不支持所有选项。该扩展的发现机制将被扩展，以允许系统软件确定支持的大小。

其他的大小可以被故意排除，这样没有被用来表示有效的 *NAPOT* 区域大小的 *PPN* 位 *(*例如，最低有效位 *pte.ppn*[*i*]*)* 将来可能被用于其他用途。

但是，如果细粒度的中间页面大小支持被证明没有用处，我们选择只标准化 *64 KiB* 支持作为第一步。

第六章 用于基于页面的内存类型的标准扩展“Svpbmt”，版本 1.0

In Sv39, Sv48, and Sv57, bits 62–61 of a leaf page table entry indicate the use of page-based memory types that override the PMA(s) for the associated memory pages. The encoding for the PBMT bits is captured in Table 6.2.

在 Sv39、Sv48 和 Sv57 中，叶页表项的第 62–61 位指示基于页的内存类型的使用，该类型覆盖相关内存页的 PMA(s)。PBMT 比特的编码如表 6.2所示。

The Svpbmt extension depends on Sv39.

Svpbmt 扩展依赖于 Sv39。

Mode	Value	Requested Memory Attributes
PMA	0	None
NC	1	Non-cacheable, idempotent, weakly-ordered (RVWMO), main memory
IO	2	Non-cacheable, non-idempotent, strongly-ordered (I/O ordering), I/O
–	3	*Reserved for future standard use*

表 6.1: Encodings for the PBMT field in Sv39, Sv48, and Sv57 PTEs. Attributes not mentioned are inherited from the PMA associated with the physical address.

模式	值	需要的内存属性
PMA	0	无
NC	1	不可缓存、幂等、弱内存顺序（RVWMO）、主存储器
IO	2	不可缓存、非幂等、强内存顺序（I/O 排序）、I/O
–	3	保留以供未来标准使用

表 6.2: 表 6.1：Sv39、Sv48 和 Sv57 PTEs 中 PBMT 字段的编码。未提及的属性是从与物理地址关联的 PMA 继承的。

Future extensions may provide more and/or finer-grained control over which PMAs can be overridden.

未来的扩展可能提供更多和/或更细粒度的控制，这些拓展中 *PMA* 可以被覆盖。

For non-leaf PTEs, bits 62–61 are reserved for future standard use. Until their use is defined by a standard extension, they must be cleared by software for forward compatibility, or else a page-fault exception is raised.

对于非叶 PTE，位 62–61 保留供将来的标准使用。在标准扩展定义它们的用法之前，必须由软件清除它们以实现前向兼容性，否则会引发页面错误异常。

For leaf PTEs, setting bits 62−-61 to the value 3 is reserved for future standard use. Until this value is defined by a standard extension, using this reserved value in a leaf PTE raises a page-fault exception.

对于叶 PTE，将位 62-61 设置为值 3 被保留以供将来的标准使用。在该值由标准扩展定义之前，在叶 PTE 中使用该保留值会引发页面错误异常。

If the underlying physical memory attribute for a page is vacant, the PBMT settings do not override that.

如果页面的底层物理内存属性为空，则 PBMT 设置不会覆盖该属性。

When PBMT settings override a main memory page into I/O or vice versa, memory accesses to such pages obey the memory ordering rules of the final effective attribute, as follows.

当 PBMT 设置将主存页覆盖为 I/O 或者在相反的情况下，对这些页的内存访问遵循最终有效属性的内存排序规则，如下所示。

If the underlying physical memory attribute for a page is I/O, and the page has PBMT=NC, then accesses to that page obey RVWMO. However, accesses to such pages are considered to be *both* I/O and main memory accesses for the purposes of FENCE, *.aq*, and *.rl*.

如果页面的底层物理内存属性是 I/O，并且该页面具有 PBMT=NC，则对该页面的访问遵循 RVWMO。然而，出于 FENCE、*.aq* 和*.rl* 的目的，对这些页面的访问被认为是 I/O 和主内存访问。

If the underlying physical memory attribute for a page is main memory, and the page has PBMT=IO, then accesses to that page obey strong channel 0 I/O ordering rules with respect to other accesses to physical main memory and to other accesses to pages with PBMT=IO. However,

accesses to such pages are considered to be *both* I/O and main memory accesses for the purposes of FENCE, *.aq*, and *.rl*.

如果页面的底层物理内存属性是主存，并且该页面具有 PBMT=IO，那么对于物理主存的其他访问以及对于 PBMT=I/O 的页面的其他访问，遵循强通道 0 I/O 排序规则。然而，出于 FENCE、*.aq* 和*.rl* 的目的，对这些页面的访问被认为是 I/O*both* 主内存访问。

A device driver written to rely on I/O strong ordering rules will not operate correctly if the address range is mapped with PBMT=NC. As such, this configuration is discouraged.

如果地址范围映射到 PBMT=NC 的区域，则编写为依赖 I/O 强排序规则的设备驱动程序将无法正常运行。因此，不鼓励这种配置。

It will often still be useful to map physical I/O regions using PBMT=NC so that write combining and speculative accesses can be performed. Such optimizations will likely improve performance when applied with adequate care.

使用 PBMT=NC 映射物理 I/O 区域通常仍然很有用，可以方便执行写合并和推测访问。如果应用时足够小心，这样的优化可能会提高性能。

当 Svpbmt 与非零 PBMT 编码一起使用时，同一个物理页面的多个虚拟别名可能同时存在，且具有不同的内存属性。通过启用 Svpbmt 的 PTE, U 模式或 S 模式映射也可以观察给定物理内存区域的不同内存属性，而不是通过 M 模式或 MODE=Bare 时执行的对同一页面的并发访问。在这种情况下，可能会违反由属性决定的行为 (包括一致性，它在其他方面是不受影响的)。

使用非缓存的不同属性 (例如 NC 和 IO) 访问相同的位置不会导致一致性的丧失，但可能会导致较弱的内存排序，而不是通常保证的更严格的属性。在这些访问之间执行 `fence iorw, iorw` 指令足以防止内存顺序的丢失。

`fence iorw, iorw`, followed by `cbo.flush` to an address of that location, followed by a `fence iorw, iorw`. 使用不同的缓存属性访问相同的位置可能会导致一致性的丧失。在这种访问之间执行以下序列，可以防止一致性和内存顺序的丢失: `fence iorw, iorw`，然后是 `cbo.flush` 到那个位置的地址，然后是 `fence iorw, iorw`。

因此，如果以后可能使用原始属性引用相同的位置，则必须事先重复此序列。

在某些情况下，较弱的序列可能足以防止相干性的丧失。这些情况将在即将进行的 RVWMO 内存模型与 Zicbom 扩展中的指令交互的形式化之后详细介绍。

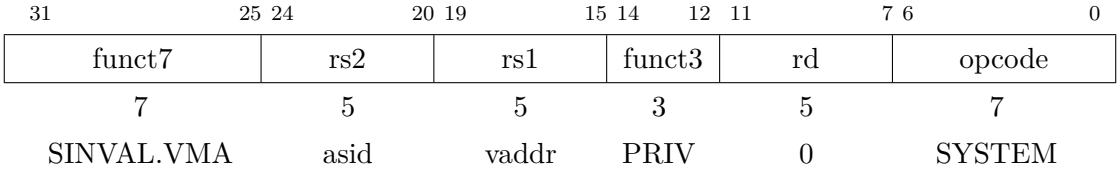
When two-stage address translation is enabled within the H extension, the page-based memory types are also applied in two stages. First, if `hgatp`.MODE is not equal to zero, non-zero G-stage PTE PBMT bits override the attributes in the PMA to produce an intermediate set of attributes. Otherwise, the PMAs serve as the intermediate attributes. Second, if `vsatp`.MODE is not equal to

zero, non-zero VS-stage PTE PBMT bits override the intermediate attributes to produce the final set of attributes used by accesses to the page in question. Otherwise, the intermediate attributes are used as the final set of attributes. 当在 H 扩展中启用两阶段地址转换时，基于页面的内存类型也会分两阶段应用。首先，如果 `hgatp`.MODE 不等于零，非零 G-stage PTE PBMT 位覆盖 PMA 中的属性以生成中间属性集。否则，PMA 作为中间属性。其次，如果 `vsatp`.MODE 不等于零，非零 VS-stage PTE PBMT 位将覆盖中间属性，以生成访问相关页面所使用的最终属性集。否则，将使用中间属性作为最终的属性集。

第七章　用于细粒度地址转换缓存失效的“Svinval”标准扩展，1.0 版

The Svinval extension splits SFENCE.VMA, HFENCE.VVMA, and HFENCE.GVMA instructions into finer-grained invalidation and ordering operations that can be more efficiently batched or pipelined on certain classes of high-performance implementation.

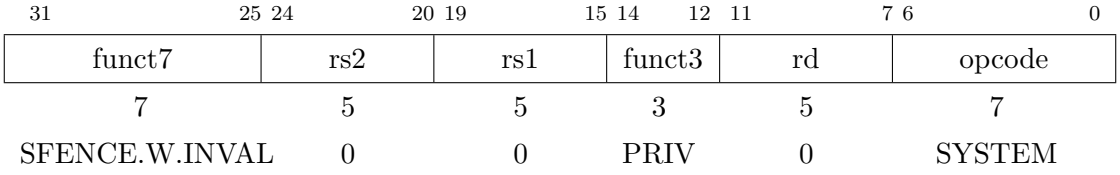
Svinval 扩展将 SFENCE.VMA、HFENCE.VVMA 和 HFENCE.GVMA 指令拆分为细粒度的无效化和排序操作，这些操作可以在某些高性能的具体实现上更高效地进行批处理或流水线化。



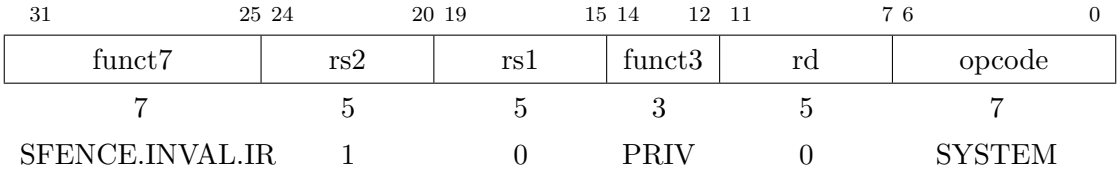
31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
SINVAL.VMA	asid	vaddr	PRIV	0	SYSTEM

The SINVAL.VMA instruction invalidates any address-translation cache entries that an SFENCE.VMA instruction with the same values of *rs1* and *rs2* would invalidate. However, unlike SFENCE.VMA, SINVAL.VMA instructions are only ordered with respect to SFENCE.VMA, SFENCE.W.INVAL, and SFENCE.INVAL.IR instructions as defined below.

SINVAL.VMA 指令和具有相同 *rs1* 和 *rs2* 值的 SFENCE.VMA 指令功能类似，会使相同的地址转换缓存条目无效。然而，与 SFENCE.VMA 不同，SINVAL.VMA 指令仅对下面定义的 SFENCE.VMA、SFENCE.W.INVAL 和 SFENCE.INVAL.IR 指令进行排序。



31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
SFENCE.W.INVAL	0	0	PRIV	0	SYSTEM



The SFENCE.W.INVAL instruction guarantees that any previous stores already visible to the current RISC-V hart are ordered before subsequent SINVAL.VMA instructions executed by the same hart. The SFENCE.INVAL.IR instruction guarantees that any previous SINVAL.VMA instructions executed by the current hart are ordered before subsequent implicit references by that hart to the memory-management data structures.

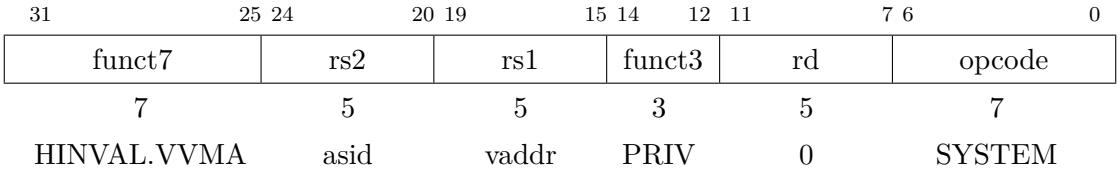
SFENCE.W.INVAL 指令保证在同一硬件线程执行后续 SINVAL.VMA 指令之前，对当前 RISC-V 硬件线程已经可见的任何先前的存储进行排序。SFENCE.INVAL.IR 指令保证在当前硬件线程在后续对内存管理数据结构隐式引用之前，对当前硬件线程先前执行的所有 SINVAL.VMA 指令进行排序。

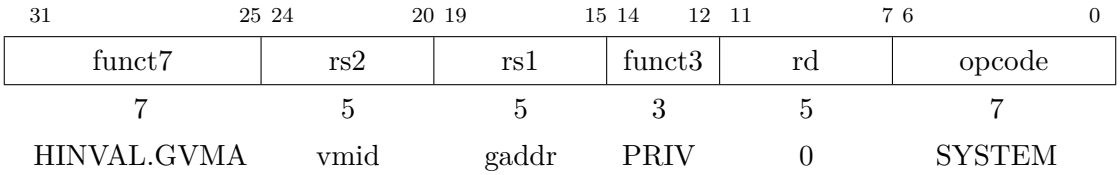
When executed in order (but not necessarily consecutively) by a single hart, the sequence SFENCE.W.INVAL, SINVAL.VMA, and SFENCE.INVAL.IR has the same effect as a hypothetical SFENCE.VMA instruction in which:

- the values of *rs1* and *rs2* for the SFENCE.VMA are the same as those used in the SINVAL.VMA,
- reads and writes prior to the SFENCE.W.INVAL are considered to be those prior to the SFENCE.VMA, and
- reads and writes following the SFENCE.INVAL.IR are considered to be those subsequent to the SFENCE.VMA.

当由单个硬件线程按顺序（但不一定连续）执行时，序列 SFENCE.W.INVAL、SINVAL.VMA 和 SFENCE.INVAL.IR 与假设的 SFENCE.VMA 指令具有相同的效果，假设的 SFENCE.VMA 指令满足：

- SFENCE.VMA 的 rs1 和 rs2 值与 SINVAL.VMA 中使用的值相同，
- SFENCE.W.INVAL 之前的读和写被视为 SFENCE.VMA 之前的读写，以及
- SFENCE.INVAL.IR 之后的读和写被认为是 SFENCE.VMA 在之后。





If the hypervisor extension is implemented, the Svinval extension also provides two additional instructions: HINVAL.VVMA and HINVAL.GVMA. These have the same semantics as SINVAL.VMA, except that they combine with SFENCE.W.INVAL and SFENCE.INVAL.IR to replace HFENCE.VVMA and HFENCE.GVMA, respectively, instead of SFENCE.VMA. In addition, HINVAL.GVMA uses VMIDs instead of ASIDs.

如果实现了管理程序扩展，Svinval 扩展还提供了两条附加指令：HINVAL.VVMA 和 HINVAL.GVMA。它们与 SINVAL.VMA 具有相同的含义，只是它们分别地与 SFENCE.W.INVAL 和 SFENCE.INVAL.IR 结合，并分别替换 HFENCE.VVMA 和 HFENCE.GVMA，而不是 SFENCE.VMA。此外，HINVAL.GVMA 使用 VMIDs 而不是 ASIDs。

SINVAL.VMA, HINVAL.VVMA, and HINVAL.GVMA require the same permissions and raise the same exceptions as SFENCE.VMA, HFENCE.VVMA, and HFENCE.GVMA, respectively. In particular, an attempt to execute any of these instructions in U-mode always raises an illegal instruction exception, and an attempt to execute SINVAL.VMA or HINVAL.GVMA in S-mode or HS-mode when `mstatus`.TVM=1 also raises an illegal instruction exception. An attempt to execute HINVAL.VVMA or HINVAL.GVMA in VS-mode or VU-mode, or to execute SINVAL.VMA in VU-mode, raises a virtual instruction exception. When `hstatus`.VTVM=1, an attempt to execute SINVAL.VMA in VS-mode also raises a virtual instruction exception.

SINVAL.VMA、HINVAL.VVMA 和 HINVAL.GVMA 分别需要与 SFENCE.VMA、HFENCE.VVMA 和 HFENCE.GVMA 相同的权限并引发相同的异常。特别是，尝试在 U 模式下执行这其中任何一个指令总是会引发非法指令异常，而当 `mstatus`.TVM=1 时，尝试在 S 模式或 HS 模式下执行 SINVAL.VMA 或 HINVAL.GVMA 也会引发非法的指令异常。尝试在 VS 模式或 VU 模式下执行 HINVAL.VVMA 或 HINVAL.GVMA 指令，或在 VU 模式中执行 SINVAL.VMA 指令，会引发虚拟指令异常。当 `hstatus`.VTVM=1 时，尝试在 VS 模式下执行 SINVAL.VMA 也会引发虚拟指令异常。

SFENCE.W.INVAL and SFENCE.INVAL.IR instructions do not need to be trapped when `mstatus`*.TVM=1 or when* `hstatus`*.VTVM=1, as they only have ordering effects but no visible side effects. Trapping of the SINVAL.VMA instruction is sufficient to enable emulation of the intended overall TLB maintenance functionality.*

当 `mstatus`*.TVM=1* 或 `hstatus`*.VTVM=1* 时，*SFENCE.W.INVAL* 和 *SFENCE.INVALIR* 指令不需要陷阱陷入，因为它们只具有排序效果，没有可见的副作用。*SINVAL.VMA* 指令的陷入足以实现对预期全部 *TLB* 维护功能的仿真。

In typical usage, software will invalidate a range of virtual addresses in the address-translation caches by executing an SFENCE.W.INVAL instruction, executing a series of SINVAL.VMA, HINVAL.VVMA, or HINVAL.GVMA instructions to the addresses (and optionally ASIDs or VMIDs) in question, and then executing an SFENCE.INVAL.IR instruction.

在典型用法中，软件将通过执行 *SFENCE.W.INVAL* 指令，对相关的地址（和可选的 *ASIDs* 或 *VMIDs*）执行一系列 *SINVAL.VMA*、*HINVAL.VVMA* 或 *HINVAL.GVMA* 指令，然后执行 *SFENCE.INVAL.IR* 指令，使地址转换缓存中的一系列虚拟地址无效。

High-performance implementations will be able to pipeline the address-translation cache invalidation operations, and will defer any pipeline stalls or other memory ordering enforcement until an SFENCE.W.INVAL, SFENCE.INVAL.IR, SFENCE.VMA, HFENCE.GVMA, or HFENCE.VVMA instruction is executed.

高性能的具体实现将能够对地址转换缓存无效操作进行流水线处理，这将延迟任何流水线暂停或其他强制内存排序，直到执行一个 *SFENCE.W.INVAL*、*SFENCE.INVAL.IR*、*SFENCE.VMA*、*HFENCE.GVMA* 或 *HFENCE.VVMA* 指令。

Simpler implementations may implement SINVAL.VMA, HINVAL.VVMA, and HINVAL.GVMA identically to SFENCE.VMA, HFENCE.VVMA, and HFENCE.GVMA, respectively, while implementing SFENCE.W.INVAL and SFENCE.INVAL.IR instructions as no-ops.

更简单的实现可以分别依照 *SFENCE.VMA*、*HFENCE.VVMA* 和 *HFENCE.GVMA* 相同地实现 *SINVAL.VMA*、*HINVAL.VVMA* 和 *HINVAL.GVMA*，而将 *SFENCE.W.INVAL* 和 *SFENCE.INVAL.IR* 指令实现为空操作。

第八章　超级监管器拓展，1.0 版

这章描述了 RISC-V 超级监管级拓展。为了支持运行在 type-1 或 type-2 超级监管器之上的宾客（guest）操作系统的 efficient hosting，超级监管器虚拟化了监管级架构。超级监管级拓展把监管模式修改成了超级监管器拓展的监管模式（*hypervisor-extended supervisor mode*）（HS-模式, 或简称监管级模式）。一个超级监管器或一个兼容宿主机的操作系统运行在这个 HS-模式上。超级监管级拓展也添加了从宾客物理地址（*guest physical addresses*）到监管级物理地址的翻译阶段。其为宾客操作系统虚拟化内存和内存映射输入输出子系统。HS-模式的行为很像 S-模式，但是多了额外的指令和在虚拟 S-模式（VS-mode）上，控制地址翻译新阶段和支持 host 一个宾客 OS 的控制状态寄存器。HS-mode

常规的 S-模式操作系统能够在 HS-模式或在 VS-模式下作为宾客操作系统不加修改的运行。

在 HS-模式中，一个操作系统或一个超级监管器可以通过相同的 SBI（监管器二进制接口）与机器交互，就像一个操作系统直接在 S-模式与机器交互一样。一个 HS-模式的超级监管器应该为它的 VS-模式的宾客操作系统实现 SBI。

超级监管器拓展依赖于基于"I"的带有 32x 寄存器的基础整数 ISA（RV32I 或 RV64I），而不是 RV32E，因为它只有 16x 寄存器。CSR 的 `mtval` 寄存器不可以是只读的零，并且标准的基于页面的地址翻译（RV32 的 Sv32 或 RV64 的 Sv39 的精简版）必须被支持，

通过为 CSR 的 `misa` 寄存器的第 7 位（关联字母 H）设置为 1 来开启超级监管级拓展。

实现超级管理器拓展的 RISC—V 硬件线程不提倡硬布线 `misa`[7]，如若不然，这个拓展可能被关闭。

这个基线特权架构被设计用来简化经典的虚拟技术的使用，一个宾客操作系统在用户级别运行在其上，因为很少的特权指令可以被很容易的检测到和陷入。

这个超级监管器通过减少这种陷入的频率提升了虚拟化的性能。

在没有实现超级管理器拓展的平台上，通过在 *S*-模式运行超级监管器和为了超级监管器 *CSR* 访问而陷入到 *M*-模式和为了维持阴影页表（*shadow page table*）而陷入到 *M*-模式，超级监管器已经被设计去高效地模拟.

大部分 *type-2* 类型的超级监管器 *CSR* 访问是有效的 *S*-模式访问，所以不需要被陷入。超级监管器可以类似地支持嵌套虚拟化。

8.1 特权模式

当前的虚拟模式（由字母 V 表示）指示硬件线程当前是否运行在一个宾客中。

当 V=1 时，运行在 VS-模式的一个宾客操作系统之上的这个硬件线程，或者是处于虚拟 S-模式（VS-模式），或者是处于虚拟 U-模式（VU-模式）。

当 V=0 时，运行在 HS-模式的一个操作系统之上的硬件线程，或者是处于 M-模式，处于 HS-模式，或者是处于 U-模式。

虚拟模式也指示是否两个阶段地址翻译是激活（V=1）的还是非激活的（V=0）。表 8.1列出了带有超级监管器拓展的一个 RISC-V 硬件线程的可能的特权模式。

虚拟模式 (V)	名义上的特权	简称	名字	双阶段（Two-Stage）翻译
0	U	U-模式	用户模式	关闭
0	S	HS-模式	超级监管器拓展的监管级模式	关闭
0	M	M-模式	机器模式	关闭
1	U	VU-模式	虚拟的用户模式	开启
1	S	VS-模式	虚拟的监管级模式	开启

表 8.1: 带有超级监管级拓展的特权模式。Privilege modes with the hypervisor extension.

对于特权模式 U 和 VU 来说，名义上的特权模式是 U 模式。对于特权模式 HS 和 VS 来说，名义上的特权模式是 S 模式。

HS-模式比 VS-模式有更高的权限，VS-模式比 VU-模式有更高的权限。当执行在 U-模式下，VS-模式的中断是被全局关闭的。

这个描述没有考虑到 *U*-模式或 *VU*-模式的中断的可能性。并且如果一个用户级别的中断拓展被采用的话，上述描述也将被更正。

8.2 超级监管器和虚拟监管器的控制状态寄存器

一个运行在 HS-模式的操作系统或超级监管器用监管级 CSR 与异常、中断和地址转换子系统进行交互。为了管理两阶段地址翻译（two-stage address translation）和控制 VS-模式宾客操作系统的行为，额外的 CSR 被提供给了 HS-模式，但没有提供给 VS-模式。额外的 CSR 如下：`hstatus`, `hedeleg`, `hideleg`, `hvip`, `hip`, `hie`, `hgeip`, `hgeie`, `henvcfg`, `henvcfgh`, `hcounteren`, `htimedelta`, `htimedeltah`, `htval`, `htinst`, and `hgatp`.

此外，一些虚拟监管级 CSR（VS CSRs）是普通监管级 CSR 的复制。例如：`vsstatus` 是一般 `sstatus`CSR 在 VS 模式下的复制。

当 V=1 时，VS 控制状态寄存器一般的读或修改一个监管级 CSR 的指令应该访问相应的 VS 控制状态寄存器。当 V=1 时，试图通过各自的 CSR 地址，直接读或写一个 VS 控制状态寄存器会引起一个虚拟指令异常。（从 U-模式的尝试会像往常那样引起一个非法指令异常。）VS 控制状态寄存器只能从 M-模式或 HS-模式访问它们。

当 V=1 时，一般被 VS 级 CSR 代替的 HS 级监管级 CSR 保持它们的值，但是不影响这个机器的行为，除非具体的文件声明不这么做。相反的，当 V=0 时，VS 级控制状态寄存器通常不影响这个机器的行为，除非被 CSR 指令读或写。

一些标准的监管级控制状态寄存器（`senvcfg`，`scounteren`，和 `scontext`，可能还有其他）没有对应的 VS 控制状态寄存器。即使当 V=1 时，这些监管级控制状态寄存器保持他们的功能和可访问性，除非 VS 模式和 VU 模式代替了 HS 模式和 U 模式。超级监管器软件应该在需要时手动更换这些寄存器的值。

只有当监管级控制状态寄存器必须被复制时，与之匹配的 *VS* 控制状态寄存器才会存在。（监管级控制状态寄存器在被陷入自动写入，或在陷入实体（*trap entry*）之后、*SRET* 之前影响了指令的执行，或当软件独自不能在恰当时刻交换一个控制状态寄存器时，才被复制）*Matching VS CSRs exist only for the supervisor CSRs that must be duplicated, which are mainly those that get automatically written by traps or that impact instruction execution immediately after trap entry and/or right before SRET, when software alone is unable to swap a CSR at exactly the right moment.* 当前，大多数监管级控制状态寄存器都属于这一类，但将来可能不会。*Currently, most supervisor CSRs fall into this category, but future ones might not.*

在这章，我们用术语“*HSXLEN*”指代在执行在 HS 模式下有效的 XLEN，和术语“*VSXLEN*”指代执行在 VS 模式下有效的 XLEN。

8.2.1 超级管理器状态寄存器 (hstatus)

hstatus 寄存器是一个 HSXLEN 位读/写寄存器。当 HSXLEN=32 时，其规格表示为图 8.1。当 HSXLEN=64 时，其规格表示为图 8.2 这个 hstatus 寄存器为追踪和控制 VS 模式宾客操作系统的异常行为提供便利，类似于 mstatus 寄存器。

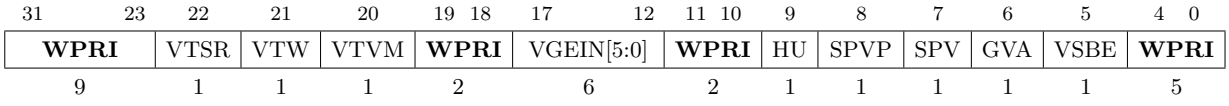


图 8.1: 当 HSXLEN=32 时，超级监管器状态寄存器（hstatus）。Hypervisor status register (hstatus) when HSXLEN=32.

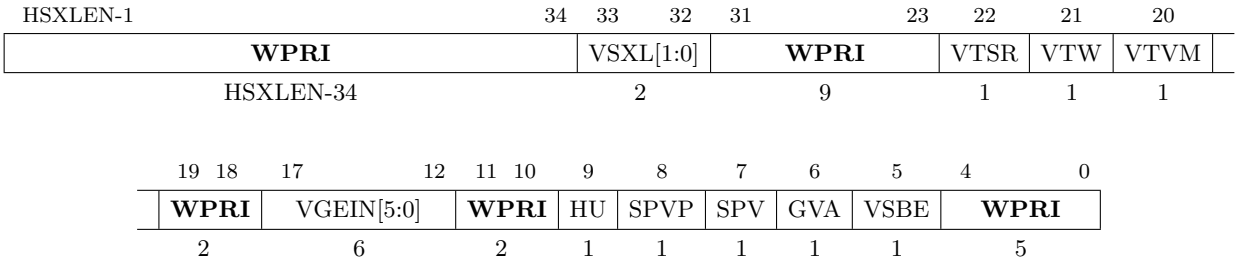


图 8.2: 当 HSXLEN=64 时，超级监管器状态寄存器（hstatus）。Hypervisor status register (hstatus) when HSXLEN=64.

VSXL 域控制 VS 模式下有效的 XLEN（VSXLEN），VSXLEN 可能与 HS 模式下的 XLEN（HSXLEN）不同。当 HSXLEN=32 时，VSXL 域不存在，并且 VSXLEN=32。当 HSXLEN=64 时，VSXL 域是一个 **WARL** 域，其被编码为与 misa 的 MXL 相同的域，在第 18页，表 3.1中显示。特别地，一种实现可能会让 VSXL 成为一个只读的域，而它的值总是保证 VSXLEN=HSXLEN。

如果 HSXLEN 从 32 位变成更宽的宽度，并且如果 VSXL 域没有被限制为一个单一的值，那么它得到的值对应于最宽的支持长度，但不超过这个新的 HSXLEN。

hstatus 的 VTSR，VTW 和 VTVM 域并定义为类似 mstatus 的 TSR，TW 和 TVM 域，但其只在 VS 模式下影响执行，和引起虚拟指令异常而不是非法指令异常。当 VTSR=1 时，试图在 VS 模式执行 SRET 引起一个虚拟指令异常。当 VTW=1 时（假设 mstatus.TW=0），如果 WFI 试图在 VS 模式下执行 WFI 会引起一个虚拟指令异常 When VTW=1 (and assuming mstatus.TW=0), an attempt in VS-mode to execute WFI raises a virtual instruction exception if the WFI does not complete within an implementation-specific, bounded time limit. 当 VTVM=1 时，试图在 VS 模式下执行 SFENCE.VMA、SINVAL.VMA 或访问 CSRsatp 会引起一个虚拟指令异常。

VGEIN（虚拟宾客外部中断号）域为 VS 级外部中断选择一个宾客外部中断源。VGEIN 是一个 **WLRL** 域，其必须能够保持在 0 和最大宾客外部中断号（被称为 GEILEN）之间的所有值。当 VGEIN=0 时，没有宾客外部中断源为 VS 级外部中断选中。GEILEN 可能是 0，着这种情况下，VGEIN 是一个只读的 0。宾客外部中断将在第 8.2.4节被解释，VGEIN 的用法将在第 8.2.3节阐述。

HU 域（在 U 模式下的超级监管器）控制是否虚拟机的加载/存储指令，HLV，HLVX 和 HSV 也可以在 U 模式下被使用。当 HU=1 时，这些指令可以在 U 模式下执行，就像在 HS 模式下执行一样。当 HU=0 时，在 U 模式下的所有超级监管级指令都会引起一个非法指令陷入。

HU 位允许超级监管器的一部分运行在 U 模式下，为软件漏洞提供额外的保护，同时仍保持对虚拟机内存的访问。

在 HS 模式下，任何时候 trap 的执行，将导致 SPV 位（监管级上一个虚拟模式）被写入。就像 `sstatus` 中的 SPP 位被设置为陷入发生时（名义上的）特权模式，`hstatus` 中的 SPV 位在发生陷入时，被设置为虚拟模式 V 值。当在 V=0，执行 SRET 指令时，V 被这是给 SPV。

当 V=1 和一个陷入发生在 HS 模式，SPVP 位（监管级上一个虚拟特权）被设置为陷入发生时的名义上的特权模式，就像 `sstatus`.SPP。但是如果在陷入之前 V=0，SPVP 在陷入实体中不做修改。SPVP 控制由虚拟机的加载/存储指令、HLV、HLVX 和 HSV 引起的显示内存访问的有效权限。

没有 SPVP，如果指令 HLV、HLVX 和 HSV 转而去 `sstatus`.SPP 寻找它们内存访问的有效权限，那么即使 HU=1，U 模式仍旧不能在 VS 级别访问虚拟机内存，因为通过 SRET 进入 U 模式总是让 SPP=0。不像 SPP，在 HS 模式和 U 模式之间转换不会影响 SPVP 域。

在任何时候，发生在 HS 模式下的陷入，使 GVA（Guest Virtual Address）域被写入。Field GVA (Guest Virtual Address) is written by the implementation whenever a trap is taken into HS-mode. 对于任何将一个宾客虚拟地址写入 `stval` 的陷入（断点、地址非对齐、访问错误、页错误或宾客页错误（guest-page flaut），GVA 被置 1。对于在 HS 模式下的其他陷入，GVA 被置 0。

对于将一个非零值写入 `stval` 的断点和内存访问陷入，对于 SPV 域来说，GVA 是多余的（这两个位是一致的）。而当 HLV、HLVX 或 HSV 指令引起错误时，这两个域是不同的（SPV=0 但是 GVA=1）。

VSBE 位是一个 **WARL** 域，其控制 VS 模式下的显示内存访问的端序问题。如果 VSBE=0，VS 模式下的显示的加载和存储内存访问是小端序的。如果 VSBE=1，其是大端序的。VSBE 也控制所有对 VS 级别内存管理数据结构的隐式访问的端序问题，比如页表。一个具体的实现可能让 VSBE 成为一个只读的域，总是指示与 HS 模式相同的端序。

8.2.2 超级监管级陷入委托寄存器 (hedeleg 和 hideleg)

寄存器 hedeleg 和 hideleg 是 HSXLEN 位读/写寄存器，其格式如图8.3和图8.4所示。默认，任意特权级别的所有陷入都在 M 模式下被处理，尽管 M 模式经常用 medeleg 和 mideleg 控制状态寄存器委托一些陷入给 HS 模式。hedeleg 和 hideleg 控制状态寄存器允许一些陷入委托到一个 VS 模式的宾客；他们的布局与 medeleg 和 mideleg 一致。

图 8.3: 超级监管器异常委托寄存器（hedeleg）Hypervisor exception delegation register (hedeleg).

图 8.4: 超级监管器中断委托寄存器（hideleg）Hypervisor interrupt delegation register (hideleg).

在陷入和相应的 hedeleg 被设置之前，一个已经被委托给 HS 模式（用 medeleg）的同步陷入，被进一步委托给 VS 模式（如果 V=1)。每个 hedeleg 位应该要么是可写的要么是只读的零。hedeleg 的许多位需要明确说明是可写的还是 0，列举在表 8.2中。如果 IALIGN=32，那么 0 位（对应于指令地址未对齐异常）必须是可写的。

---

hedeleg 的一些位需要是可写的减少了一些超级监管器处理不同实现的压力。

如果 hideleg 相应位被设置，一个已经被委托给 HS 模式的中断 (using mideleg) 将被进一步委托给 VS 模式，在 hideleg 的 0-15 位中，10、6 和 2 位（与标准 VS 级别中断相关）是可写的，12、9、5 和 1 位（与标准的 S 级别中断相关）是只读的 0。

当一个虚拟监管级外部中断（代码为 10）被委托给 VS 模式时，它自动被机器翻译成 VS 模式下的监管级外部中断（代码为 9），包括在中断陷入中写入 vscause 中的值。类似地，一个虚拟的监管级时钟中断（6）被翻译成 VS 模式下的监管器时钟中断（5），和一个虚拟监管级软件中断（2）被翻译成 VS 模式下的一个监管级软件中断（1)。类似的，翻译过程可能或可能不平台或自定义中断触发（代码为 16 和更高）。Similar translations may or may not be done for platform or custom interrupt causes (codes 16 and above).

Bit	Attribute	Corresponding Exception
0	(See text)	Instruction address misaligned
1	Writable	Instruction access fault
2	Writable	Illegal instruction
3	Writable	Breakpoint
4	Writable	Load address misaligned
5	Writable	Load access fault
6	Writable	Store/AMO address misaligned
7	Writable	Store/AMO access fault
8	Writable	Environment call from U-mode or VU-mode
9	Read-only 0	Environment call from HS-mode
10	Read-only 0	Environment call from VS-mode
11	Read-only 0	Environment call from M-mode
12	Writable	Instruction page fault
13	Writable	Load page fault
15	Writable	Store/AMO page fault
20	Read-only 0	Instruction guest-page fault
21	Read-only 0	Load guest-page fault
22	Read-only 0	Virtual instruction
23	Read-only 0	Store/AMO guest-page fault

表 8.2: Bits of `hedeleg` that must be writable or must be read-only zero.

8.2.3 超级监管器中断寄存器 (`hvip`，`hip` 和 `hie`)

寄存器 `hvip` 是一个 HSXLEN 位读/写寄存器。一个超级监管器可以写入 `hvip`，去指示用于 VS 模式的虚拟中断。不可写的 `hvip` 的位是只读的 0。

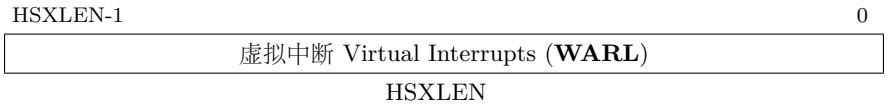


图 8.5: 超级监管器虚拟中断等待寄存器（`hvip`）Hypervisor virtual-interrupt-pending register (`hvip`).

`hvip` 的标准部分（0-15 位）被格式化为图 8.6形式。`hvip` 的 VSEIP、VSTIP 和 VSSIP 位是可写的。在 `hvip` 中，设置 VSEIP=1 表示一个 VS 级外部中断。设置 VSTIP 表示一个 VS 级时钟中断。设置 VSSIP 表示一个 VS 级软件中断。Bits VSEIP, VSTIP, and VSSIP of `hvip` are writable.

Setting VSEIP=1 in `hvip` asserts a VS-level external interrupt; setting VSTIP asserts a VS-level timer interrupt; and setting VSSIP asserts a VS-level software interrupt.

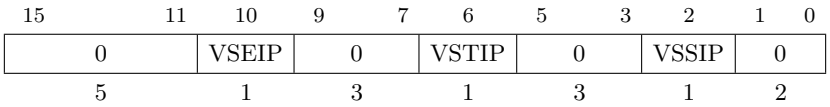


图 8.6: `hvip` 的标准部分（0-15 位）Standard portion (bits 15:0) of `hvip`.

寄存器 `hip` 和 `hie` 是 HSXLEN 位读/写寄存器。他们分别提供了 HS 级别的 `sip` 和 `sie`。`hip` 寄存器表明了等待的 VS 级和超级监管级特有的中断，而 `hie` 包含这些中断的使能位。

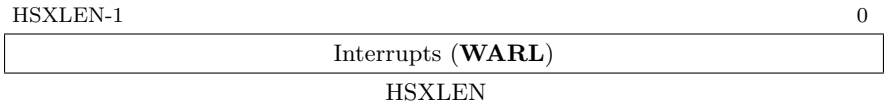


图 8.7: 超级监管器中断等待寄存器（`hip`）Hypervisor interrupt-pending register (`hip`).

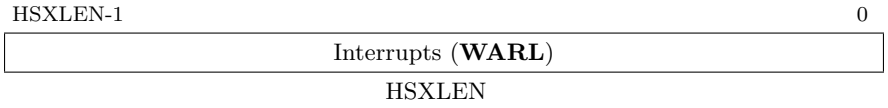


图 8.8: 超级监管级中断使能寄存器（`hie`）Hypervisor interrupt-enable register (`hie`).

在 `sie` 中每个可写的位，在 `hip` 和 `hie` 中相对应的位应该是只读的零。因此，在 `sie` 和 `hie` 中非零位总是互补的，`sip` 和 `hip` 也是一样。

`hip` 和 `hie` 的有效位不能设置在 *HS* 级的 `sip` 和 `sie` 中，因为这样做就会让那些没有在硬件上实现超级监管器拓展的平台，不能用软件模拟超级监管器拓展。

无论在何时，中断 i 都应陷入 HS 模式的情形如下：（1）要么是当前操作模式是 HS 模式，并且 `sstatus` 寄存器中的 SIE 位被设置；要么是当前操作模式拥有低于 HS 模式的权限。（2）在 `sip` 和 `sie` 或在 `hip` 和 `hie` 中，i 位被设置；并且（3）在 `hideleg` 中 i 没有被设置。

如果 `sie` 的 i 位是只读的零，在 `hip` 中相同的位可能是可写的或可能是只读的。当 `hip` 的 i 位是可写的，一个等待的中断 i 可以通过为这个位写零被清除。如果中断 i 可以在 `hip` 中等待，但 `hip` 中的 i 是只读的，那么要么这个中断可以通过清除 `hvip` 的第 i 位被清除，要么这个实现必然提供了其他技术，去清除这个等待的中断（其可能包含一个对执行环境的调用）。

如果在 `hip` 中，中断可以变成等待着的，那么在 `hie` 中的对应位应该是可写的。`hie` 中不可写的位应该是只读的零。

`hip` 和 `hie` 寄存器的标准部分（0-15 位）的格式分别显示在图8.9和8.10。

15 13	12	11	10	9 7	6	5 3	2	1 0
0	SGEIP	0	VSEIP	0	VSTIP	0	VSSIP	0
3	1	1	1	3	1	3	1	2

图 8.9: `hip` 的标准部分（0-15 位）Standard portion (bits 15:0) of `hip`.

15 13	12	11	10	9 7	6	5 3	2	1 0
0	SGEIE	0	VSEIE	0	VSTIE	0	VSSIE	0
3	1	1	1	3	1	3	1	2

图 8.10: `hie` 的标准部分（0-15 位）Standard portion (bits 15:0) of `hie`.

`hip`.SGEIP 和 `hie`.SGEIE 位是为处于监管级（HS 级）宾客外部中断使用的中断等待位和中断使能位。在 `hip` 中 SGEIP 是只读的，并且当且仅当控制寄存器 `hgeip` 和 `hgeie` 的位级逻辑与在每一位都为非零时，其值为 1。(见第 8.2.4 节 8.2.4)

`hip`.VSEIP 位和 `hie`.VSEIE 位是 VS 级外部中的的中断等待位和中断使能位。在 `hip` 中 VSEIP 位是只读的，是以下中断源的逻辑或：

- `hvip` 的 VSEIP 位；
- `hstatus`.VGEIN 选中的 `hgeip` 位；和
- 任何其他具体平台针对 VS 级别的外部中断信号。

`hip`.VSTIP 位和 `hie`.VSTIE 位是 VS 级时钟中断的中断等待和中断使能位。在 `hip` 中 VSTIP 是只读的，并且是 `hvip`.VSTIP 和其他具体平台针对 VS 级别时钟中断信号的逻辑或。

`hip`.VSSIP 位和 `hie`.VSSIE 位是 VS 级软件中断的中断等待和中断使能位。`hip` 的 VSSIP 位是 `hvip` 的相同位的别名（可写）。

为 HS 模式指定的多个类似的中断被按照如下降序方式处理：SEI, SSI, STI, SGEI, VSEI, VSSI, VSTI。

8.2.4 超级监管级宾客外部中断寄存器（`hgeip` 和 `hgeie`）

`hgeip` 寄存器是一个 HSXLEN 位只读寄存器，其格式如下图 8.11所示，其指示当前硬件线程的等待宾客外部中断。`hgeie` 寄存器是一个 HSXLEN 位读/写寄存器，其格式如下图 8.12所示，其包含当前硬件线程的宾客外部中断的使能位。宾客外部中断号 i 对应于 `hgeip` 和 `hgeie` 中的 i 位。Guest external interrupt number i corresponds with bit i in both `hgeip` and `hgeie`.

宾客外部中断代表针对 VS 级别的各个虚拟机的中断。如果一个 RISC-V 平台支持使一个物理设备直接被一个宾客操作系统通过与超级监管器最小交互的方式控制（被称为在一个虚拟机和物理设备

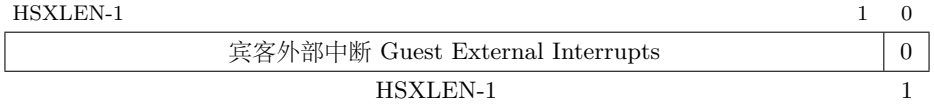


图 8.11: 超级监管器宾客外部中断等待寄存器（hgeip）Hypervisor guest external interrupt-pending register (hgeip).

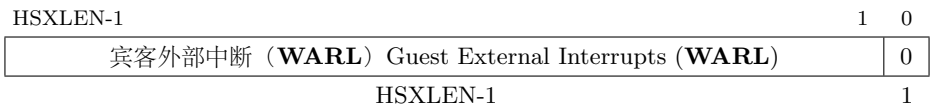


图 8.12: 超级监管器宾客外部中断使能寄存器（hgeie）Hypervisor guest external interrupt-enable register (hgeie).

之间的直通（*pass-through*）或直接分配（*direct assignment*）（译者注：这里需要注一下这两种方式），那么，在这种环境下，从这个设备发出的中断仅供一个特殊的虚拟机。hgeip 的每一位总结了针对一个虚拟硬件线程的所有等待中断，由中断控制器收集并报告。为了区分从多个设备发出的具体的等待的中断，软件必须询问中断控制器。

为了支持宾客外部中断，需要一个中断控制器，其会将针对虚拟机的中断和其他中断分开收集。

hgeip 和 hgeie 中为宾客外部中断实现的位数是未指定的 ，并且可能是零。这个数字被称为 *GEILEN*。除了第 0 位，最低点被最先实现。因此如果 GEILEN 不是零，1-GEILEN 位应该是在 hgeie 中是可写的，并且 hgeip 和 hgeie 的其他位应该是只读的零。

在一个物理硬件线程中被接收和处理的宾客外部中断集合可能与在其他物理硬件线程接收的不同。在一个物理硬件线程中，宾客外部中断号 i 不应该与其他硬件线程上的宾客外部中断 i 相同。对于任何一个物理硬件线程，可能直接接受宾客外部中断的最大虚拟硬件线程数被 *GEILEN* 限制。对于 *RV32* 的任何实现而言，每个物理硬件线程的最大值为 *31*；对于 *RV64* 而言，是 *63*。

超级监管器总是可以为任何数量的虚拟硬件线程模拟设备，而不受 *GEILEN* 的限制。只有直接传递（*pass-through*）*(*直接分配*)* 的中断受到 *GEILEN* 限制的影响，并且限制是接收这种中断的虚拟硬件线程的数量，而不是接收到的不同中断的数量。一个单独的虚拟硬件线程上接收到的不同线程的数量被中断控制器所决定。

hgeie 寄存器选择可以引起监管级（HS 级）中断的宾客外部中断的子集。hgeie 中的使能位不会影响通过 hstatus.VGEIN 被 hgeip 选出的 VS 级外部中断信号。

8.2.5 超级监管器环境配置寄存器（henvcfg 和 henvcfgh）

henvcfg CSR 是一个 HSXLEN-bit 读/写寄存器，格式化为 HSXLEN=64，如图 8.13所示，它控制虚拟模式 V=1 时执行环境的某些特征。

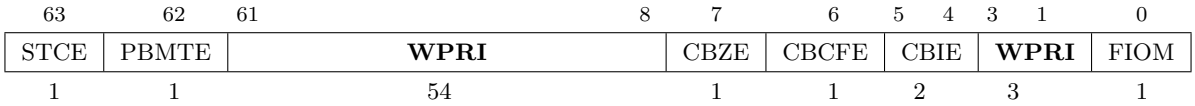


图 8.13: HSXLEN=64 下，超级监管器环境配置寄存器（henvcfg）。

如果在 henvcfg 中将位 FIOM (I/O 的 Fence 意味着内存) 设置为 1，那么当 V=1 时执行的 Fence 指令将被修改，因此对设备 I/O 的访问顺序要求也意味着对主存访问顺序的要求。表 8.3详细描述了 FIOM=1 和 V=1 时 FENCE 指令位 PI、PO、SI 和 SO 的修改解释。

类似地，当 FIOM=1 和 V=1 时，如果访问作为设备 I/O 排序的区域的原子指令有它的 *aq* 和/或 *rl* 位设置，那么该指令的排序就像它访问设备 I/O 和内存一样。

Instruction bit	Meaning when set
PI	Predecessor device input and memory reads (PR implied)
PO	Predecessor device output and memory writes (PW implied)
SI	Successor device input and memory reads (SR implied)
SO	Successor device output and memory writes (SW implied)

表 8.3: 修改了 FIOM=1 和虚拟模式 V=1 时的 FENCE 前集合和后集合的解释。

PBMTE 位控制 Svpbmt 扩展是否可用于 VS 级地址转换。当 PBMTE=1 时，Svpbmt 可用于 VS 级地址转换。当 PBMTE=0 时，实现行为就好像 Svpbmt 没有实现 VS 阶段地址转换。如果没有实现 Svpbmt，则 PBMTE 为只读零。

STCE 字段的定义将由即将到来的 Sstc 扩展提供。它在 henvcfg 中的分配可能在该扩展批准之前发生变化。

CBZE 字段的定义将由即将到来的 Zicboz 扩展提供。它在 henvcfg 中的分配可能在该扩展批准之前发生变化。

CBCFE 和 CBIE 字段的定义将由即将到来的 Zicbom 扩展提供。它们在 henvcfg 内的分配可能在批准该扩展之前发生变化。

当 HSXLEN=32 时，henvcfg 包含与 HSXLEN=64 时 henvcfg 的 31:0 位相同的字段。此外，当 HSXLEN=32 时，henvcfgh 是一个 32 位的读/写寄存器，它包含与 HSXLEN=64 时 henvcfg 的 63:32 位相同的字段。HSXLEN=64 时，寄存器 henvcfgh 不存在。

8.2.6 超级监管器计数器使能寄存器（hcounteren）

计数器使能寄存器 hcounteren 是一个 32 位寄存器，它控制硬件性能监视计数器对宾客虚拟机的可用性。

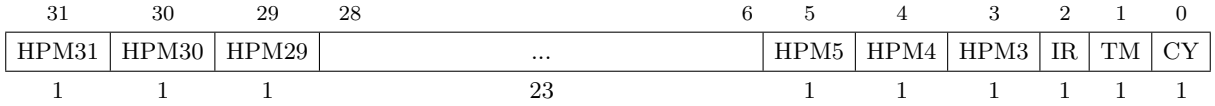


图 8.14: 超级监管器计数器使能寄存器（hcounteren）

当 hcounteren 寄存器中的 CY、TM、IR 或 HPMn 位为空时，当 V=1 时试图读取 cycle、time、instret 或 hpmcounter$n$ 寄存器时，如果 mcounteren 中的同一位为 1，则将导致虚拟指令异常。当这些位中的一个被设置，当 V=1 时，访问相应的寄存器是被允许的，除非有其他原因阻止。在 VU 模式下，除非在 hcounteren 和 scounteren 中同时设置了适用的位，否则计数器是不可读的。

hcounteren 必须执行。然而，任何位都可能是只读零，这表明当 V=1 时，对相应计数器的读取将导致异常。因此，它们是有效的 **WARL** 字段。

8.2.7 超级监管器时间偏移（Delta）寄存器（htimedelta，htimedeltah）

htimedelta CSR 是一个读/写寄存器，它包含 time CSR 的值与 vs 模式或 vu 模式下返回值之间的差值。也就是说，在 VS 或 VU 模式下读取 time CSR 将返回 htimedelta 的内容和 time 的实际值之和。

因为在计算 htimedelta 和 time 时，会忽略 *overflow*，所以可以使用 htimedelta 的大值来表示负的时间偏移量。

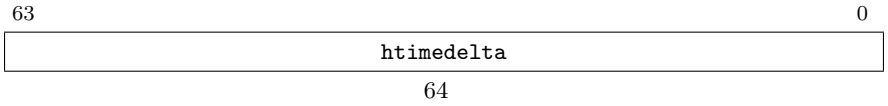


图 8.15: 超级监管器时间偏移（Delta）寄存器，HSXLEN=64

仅对于 HSXLEN=32，htimedelta 保存增量的下 32 位，htimedeltah 保存增量的上 32 位。

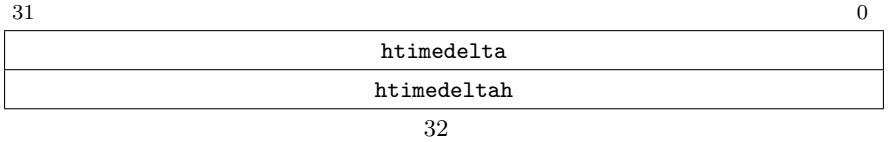


图 8.16: 超级监管器时间偏移寄存器，HSXLEN=32。

8.2.8 超级监管器陷入值寄存器（htval）

htval 寄存器是一个 HSXLEN-bit 的读/写寄存器，其格式如图 8.17所示。当一个陷入进入 HS 模式时，htval 与 stval 一起被写入额外的特定于异常的信息，以协助软件处理该陷入。

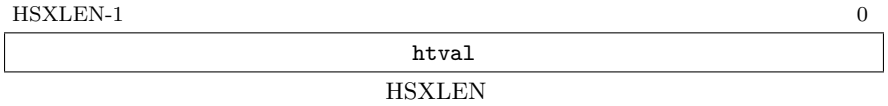


图 8.17: 超级监管器陷入值寄存器（htval）。

当客户页面错误陷入进入 HS 模式时，htval 用零或出错的客户物理地址右移 2 位写入。对于其他的陷入，htval 被设置为零，但是未来的标准或扩展可能会为其他的陷阱重新定义 htval 的设置。

宾客页错误可能是由于在第一阶段 (VS 阶段) 地址翻译期间的隐式内存访问而引起的，在这种情况下，写入 htval 的宾客物理地址就是发生错误的隐式内存访问的地址——例如，无法读取的 VS 级页表项的地址。(当 VS 阶段翻译失败时，与原始虚拟地址对应的来宾物理地址是未知的。)CSR htinst 中提供了其他信息来消除这种情况的歧义。

否则，对于导致来宾页面错误的加载和存储错误，htval 中的非零宾客物理地址对应于由 stval 中的虚拟地址指示的访问错误部分。对于具有变长指令的系统上的指令宾客页面错误，非零 htval 对应于由 stval 中的虚拟地址表示的指令的错误部分。

写入 htval 的宾客物理地址右移 *2* 位，以容纳比当前 *XLEN* 更宽的地址。对于 *RV32*，超级管理器拓展允许宾客物理地址宽至 *34* 位，htval 报告地址的第 *33:2* 位。宾客物理地址的这种按 *2* 移位编码与 *PMP* 地址寄存器 *(*第 **??**节*)* 和页表条目 *(*第*4.3*节，第*4.4*节，第*4.5*节和第 *4.6*节*)* 中的物理地址编码相匹配。

如果需要出现故障的宾客物理地址的最低有效位的两个位，这些位通常与 stval 中出错的虚拟地址的最低有效位的两个位相同。对于由于 *VS* 阶段地址翻译的隐式内存访问而导致的错误，最低有效位的两位改为零。可以使用寄存器 htinst 中提供的值来区分这些情况。

htval 是一个 **WARL** 寄存器，它必须能够容纳零，并且可能只能容纳其他 2 位移位的宾客物理地址的任意子集 (如果有的话)。

除非有其他理由假设 *(*例如平台标准*)*，将值写入 htval 的软件应该从 htval 回读以确认存储的值。

8.2.9 超级监管器陷入指令寄存器（htinst）

htinst 寄存器是一个 HSXLEN-bit 的读/写寄存器，其格式如图 8.18所示。当一个陷入进入 HS 模式时，htinst 被写入一个值，如果该值非零，则提供发生陷入指令的信息，以协助软件处理该陷入。在发生陷入时，可能写入 htinst 的值记录在第 8.6.3节中。

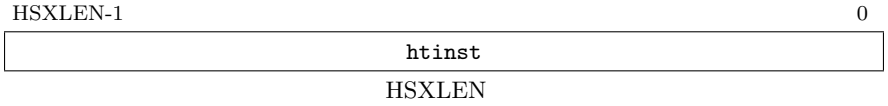


图 8.18: 超级监管器陷入指令寄存器（htinst）。

htinst 是一个 **WARL** 寄存器，它只需要能够保存在实现中可能在发生陷入时自动写入它的值。

8.2.10 超级监管器宾客地址翻译和保护寄存器（hgatp）

hgatp 寄存器是一个 HSXLEN 位读/写寄存器，其 HSXLEN=32 格式如图 8.19所示，其 HSXLEN=64 格式如图 8.20所示。它控制 G 阶段地址翻译和保护–宾客虚拟地址两阶段翻译的第二个阶段（见第 8.5节）。类似于 satp 控制状态寄存器，这个寄存器保存了宾客物理根页表的物理页号（PPN）；一个虚拟机标识符（VMID），它在每虚拟机（per-virtual-machine）的基础上提供地址翻译屏障。Similar to CSR satp, this register holds the physical page number (PPN) of the guest-physical root page table; a virtual machine identifier (VMID), which facilitates address-translation fences on a per-virtual-machine basis; and the MODE field, which selects the address-translation scheme for guest physical addresses. 当 mstatus.TVM=1 时，在 HS 模式下试图去读或写 hgatp 会引起一个非法指令异常。

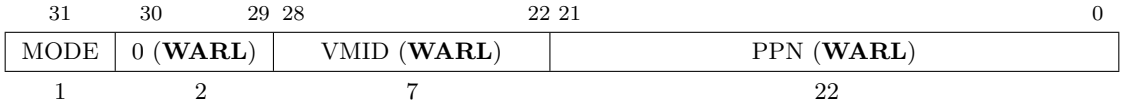


图 8.19: 当 HSXLEN=32 时，超级监管器宾客地址翻译和保护寄存器 hgatp

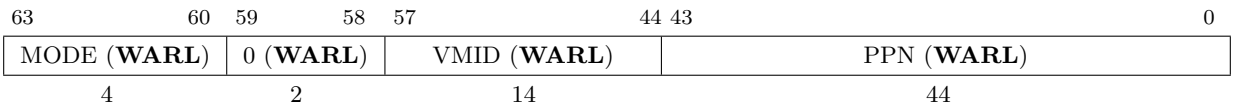


图 8.20: 当 HSXLEN=64 时，模式值可能为裸机、Sv39x4、Sv48x4、Sv57x4 的，模式超级监管器宾客地址翻译和保护寄存器 hgatp

表 8.4展示了，当 HSXLEN=32 和 HSXLEN=64 时，MODE 域的编码。当 MODE=Bare 时，宾客物理地址与监管级物理地址一致。并且，对于一个宾客虚拟机也无需用超过物理内存保护方案（在第 3.7节描述）去保护内存。在这种情况下，hgatp 的剩余域必须被设置为 0。

当 HSXLEN=32 时，MODE 的唯一值是 Sv32x4。Sv32x4 是平常的 Sv32 页虚拟内存方案的修改版，其拓展支持了 34 位宾客物理地址。当 HSXLEN=64 使，Sv39x4，Sv48x4，和 Sv57x4 被定义为 Sv39，Sv48，和 Sv57 页虚拟内存方案的修改版。所有这些页虚拟内存方案在第 8.5.1节描述。

HSXLEN=64 下剩余的 MODE 设置为将来使用所保留，可能在 `hgatp` 的其他域中定义不同的解释。

HSXLEN=32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32x4	Page-based 34-bit virtual addressing (2-bit extension of Sv32).

HSXLEN=64		
Value	Name	Description
0	Bare	No translation or protection.
1–7	—	*Reserved*
8	Sv39x4	Page-based 41-bit virtual addressing (2-bit extension of Sv39).
9	Sv48x4	Page-based 50-bit virtual addressing (2-bit extension of Sv48).
10	Sv57x4	Page-based 59-bit virtual addressing (2-bit extension of Sv57).
11–15	—	*Reserved*

表 8.4: `hgatp` 中 MODE 域的编码。

当 HSXLEN=64 时，不需要去实现所有定义的所有 MODE 设定。

带有不支持 MODE 值的对 `hgatp` 的写操作不会像 `satp` 一样将其忽略。而是，`hgatp` 在正常 A write to `hgatp` with an unsupported MODE value is not ignored as it is for `satp`. Instead, the fields of `hgatp` are **WARL** in the normal way, when so indicated.

就像第 8.5.1节解释的那样，对于分页虚拟机策略（Sv32x4，Sv39x4，Sv48x4，和 Sv57x4），根页表的大小是 16KiB，而且必须 16KiB 对齐。在这种模式下，`hgatp` 中的物理页号（PPN）的最低两位总是可读的零。仅支持定义的分页虚拟内存方案和/或裸机的实现可能使 PPN[1:0] 成为只读的零。

VMID 位数是未指定的 ，并且可能是零。通过向 VMID 域中每一位写入 1，然后再从 `hgatp` 中读出，查看 VMID 域中哪些位保持 1，可以决定 VMID 被实现的位数（用术语*VMIDLEN*表示）。VMID 的最低位被最先实现：这就意味着，如果 VMIDLEN $>$ 0，VMID[VMIDLEN-1:0] 是可写的。VMIDLEN 的最大值（用术语 VMIDAX 表示）是 Sv32x4 的 7 或 Sv39x4，Sv48x4，和 Sv57x4 的 14。The least-significant bits of VMID are implemented first: that is, if VMIDLEN $>$ 0, VMID[VMIDLEN-1:0] is writable. The maximal value of VMIDLEN, termed VMIDMAX, is 7 for Sv32x4 or 14 for Sv39x4, Sv48x4, and Sv57x4.

为了地址翻译算法的目的，hgatp 寄存器被认为是*活跃的*，除非有效的权限模式是 U 并且 hstatus.HU=0。The hgatp register is considered *active* for the purposes of the address-translation algorithm *unless* the effective privilege mode is U and hstatus.HU=0.

这个定义简化了 *HLV*，*HLVX*，和 *HSV* 指令推测执行的实现。

注意，在页表更新和随后的 G 阶段地址翻译之间，写 hgatp 并不会暗示任何顺序约束。如果新的虚拟机的宾客物理地址已经被修改了，或者一个 VMID 被使用，必须在写 hgatp 之前或之后执行 HFENCE.GVMA 指令（见第 8.3.2章）。

8.2.11 虚拟监管级状态寄存器（vsstatus）

vsstatus 寄存器是一个 VSXLEN 位的读/写寄存器，是 sstatus 在 VS 模式下的版本，当 VSXLEN=32 时，其格式如图 8.21所示，当 VSXLEN=64 时，其格式如图 8.22所示。当 V=1 时，vsstatus 代替 sstatus，所以平时读或修改 sstatus 的指令实际上会访问 vsstatus。

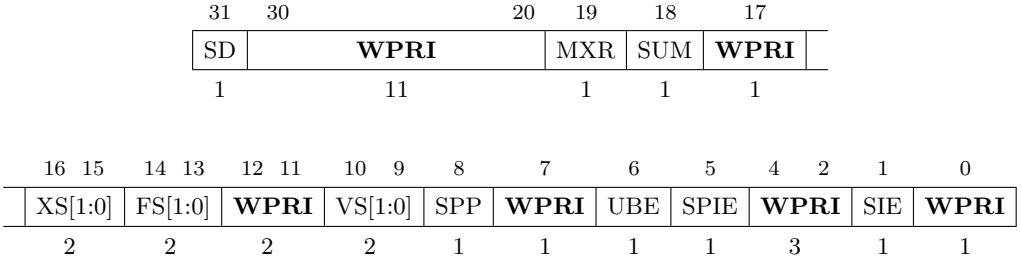


图 8.21: 当 VSXLEN=32 时，虚拟监管级状态寄存器（vsstatus）

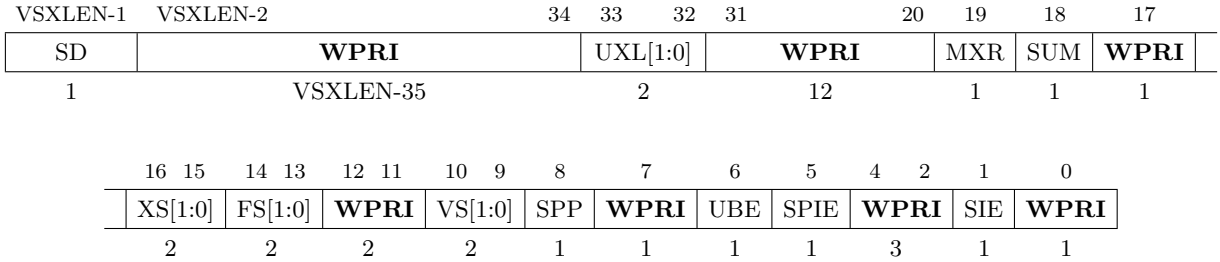


图 8.22: 当 VSXLEN=64 时，虚拟监管级状态寄存器（vsstatus）

UXL 域控制 VU 模式下有效的 XLEN，它可能不同于 VS 模式下的 XLEN（VSXLEN）。当 VSXLEN=32 时，UXL 域不会存在，并且 VU 模式的 XLEN=32。当 VSXLEN=64 时，UXL 是一个 **WARL** 域，其编码方式与 misa 的 MXL 域相同，如在 18的表 3.1所示。特别地，一种实现可能让 UXL 是 hstatusVSXL 域的只读副本，强制使 VU 模式的 XLEN=VSXLEN。

如果 VSXLEN 从 32 位变为一个更宽的宽度，并且如果 UXL 域并没有被限制成一个单一的值，那么 UXL 域会得到一个不超过这个新的 VSXLEN 的支持的最宽的长度。

当 V=1 时，`vsstatus`.FS 和 HS 级的 `sstatus`.FS 是同时有效的。当它们同时为 0（关闭）时，试图执行一个浮点指令会引发一个非法指令异常。当 V=1 时修改浮点状态会引起它们都被设置为 3（脏的）（译者注:）

对于一个受益于 *the extension context status* 的超级监管器来说，它必须有在 *HS* 级 `sstatus` 的副本，使其能够维持 *VS* 模式下独立的运行一个宾客操作系统。

While a version of the extension context status obviously must exist in `vsstatus` *for VS-mode, a hypervisor cannot rely on this version being maintained correctly, given that VS-level software can change* `vsstatus`*.FS arbitrarily. If the HS-level* `sstatus`*.FS were not independently active and maintained by the hardware in parallel with* `vsstatus`*.FS while V=1, hypervisors would always be forced to conservatively swap all floating-point state when context-switching between virtual machines.*

类似地，当 V=1 时，`vsstatus`.VS 和 HS 级的 `sstatus`.VS 同时有效。当它们同时为 0（关闭）时，试图执行一个向量指令会引发一个非法指令异常。当 V=1 时修改浮点状态会引起它们都被设置为 3（脏的）

只读域 SD 和 XS 总结了 the extension context status，因为它只对 VS 模式可见。例如：HS 级 `sstatus`.FS 的值不会影响 `vsstatus`.SD 的值。

一种实现可能会让 UBE 域是 `hstatus`.VSBE 的只读副本。

当 V=0 时，`vsstatus` 不会直接影响机器的行为，除非一个虚拟机加载/存储（HLV, HLVX, or HSV）或 `mstatus` 寄存器的 MPRV 特性被用来执行一个加载或者存储（就像 V=1 一般）。

8.2.12 虚拟监管级中断寄存器（`vsip` 和 `vsie`）

`vsip` 和 `vsie` 寄存器是 VSXLEN 位读/写寄存器，它是监管级 CSR`sip` 和 `sie` 的 VS 模式的版本，其格式分别如图8.23和8.24所示。当 V=1 时，`vsip` 和 `vsie` 代替了通常的 `sip` 和 `sie`，所以通常读或修改 `sip`/`sie` 的指令实际上会访问 `vsip`/`vsie`。然而，当 V=1 时，指向 HS 级的中断继续在 HS 级 `sip` 寄存器中指示，而不是在 `vsip` 中指示。

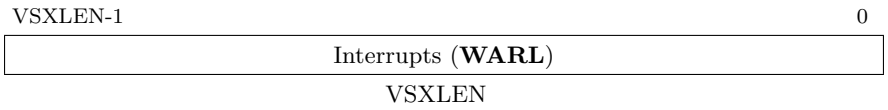


图 8.23: 虚拟监管级中断等待寄存器（`vsip`）。

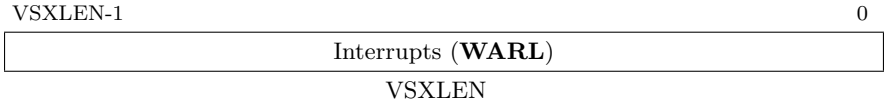


图 8.24: 虚拟监管级中断使能寄存器（vsie）。

vsip 和 vsie 寄存器的标准部分（0-15 位）的格式分别如图8.25和图8.26所示。

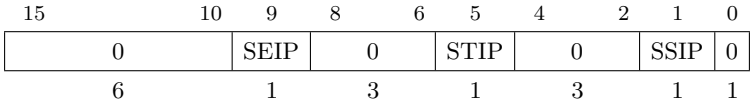


图 8.25: vsip 的标准部分（0-15 位）。

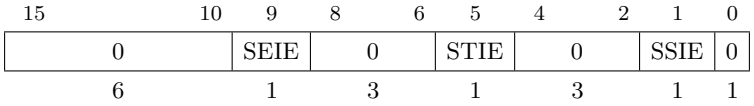


图 8.26: vsie 的标准部分（0-15 位）。

当 hideleg 的第 10 位为零时，vsip.SEIP 和 vsie.SEIE 是只读零。其他时候，vsip.SEIP 和 vsie.SEIE 是 hip 和 hie.vseie 的别名。

当 hideleg 的第 6 位为零时，vsip.STIP 和 vsie.STIE 是只读零。其他时候，vsip.STIP 和 vsie.STIE 是 hip.VSTIP 和 hie.VSTIE 的别名。

当 hideleg 的第 2 位为零时，vsip.SSIP 和 vsie.SSIE 是只读零。其他时候，vsip.SSIP 和 vsie.SSIE 是 hip.VSSIP 和 hie.VSSIE 的别名。

8.2.13 虚拟监管级陷入向量基地址寄存器（vstvec）

vstvec 寄存器是一个 VSXLEN 位读/写寄存器，它是监管级寄存器 stvec 的 VS 模式的版本，其格式如图 8.27所示。当 V=1 时，vstvec 代替了通常的 stvec，所以通常读或修改 stvec 的指令实际上会访问 vstvec。当 V=0 时，vstvec 不会直接影响机器的行为。

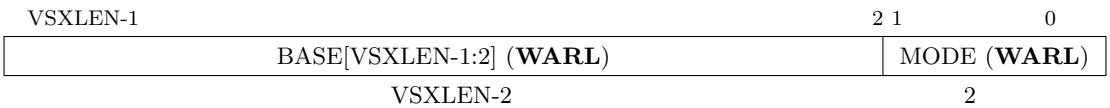


图 8.27: 虚拟监管级陷入向量基地址寄存器（vstvec）。

8.2.14 虚拟监管级 scratch 寄存器（vsscratch）

vsscratch 寄存器是一个 VSXLEN 位读/写寄存器，它是监管级寄存器 sscratch 的 VS 模式的版本，其格式如图 8.28所示。当 V=1 时，vsscratch 代替了通常的 sscratch，所以通常读或修改 sscratch 的指令实际上会访问 vsscratch。vsscratch 的内容从不会直接影响机器的行为。

图 8.28: 虚拟监管级 scratch 寄存器（vsscratch）

8.2.15 虚拟监管级异常程序计数器（vsepc）

vsepc 寄存器是一个 VSXLEN 位读/写寄存器，它是监管级寄存器 sepc 的 VS 模式的版本，其格式如图 8.29所示。当 V=1 时，vsepc 代替了通常的 sepc，所以通常读或修改 sepc 的指令实际上会访问 vsepc。当 V=0 时，vsepc 不会直接影响机器的行为。

vsepc 是一个 **WARL** 寄存器，其必须能够保持与 sepc 可以保持的相同值的集合。

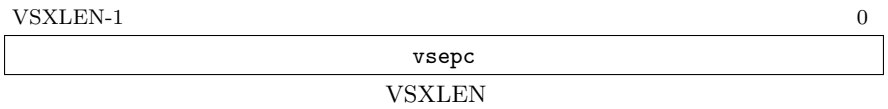


图 8.29: 虚拟监管级异常程序计数器（vsepc）。

8.2.16 虚拟监管级原因寄存器（vscause）

vscause 寄存器是一个 VSXLEN 位读/写寄存器，它是监管级寄存器 scause 的 VS 模式的版本，其格式如图 8.30所示。当 V=1 时，vscause 代替了通常的 scause，所以通常读或修改 scause 的指令实际上会访问 vscause。当 V=0 时，vscause 不会直接影响机器的行为。

vscause 是一个 **WARL** 寄存器，其必须能够保持与 scause 可以保持的相同值的集合。

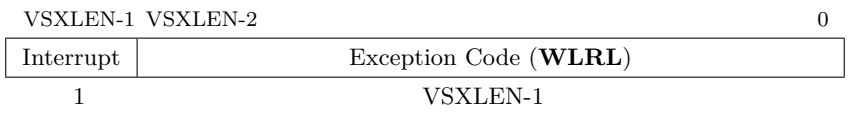


图 8.30: 虚拟监管级原因寄存器（vscause）。

8.2.17 虚拟超级监管器陷入值寄存器（vstval）

vstval 寄存器是一个 VSXLEN 位读/写寄存器，它是监管级寄存器 stval 的 VS 模式的版本，其格式如图 8.31所示。当 V=1 时，vstval 代替了通常的 stval，所以通常读或修改 stval 的指令实际上会访问 vstval。当 V=0 时，vstval 不会直接影响机器的行为。

vstval 是一个 **WARL** 寄存器，其必须能够保持与 stval 可以保持的相同值的集合。

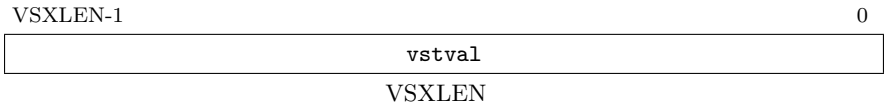


图 8.31: 虚拟超级监管器陷入值寄存器（vstval）。

8.2.18 虚拟监管级地址翻译和保护寄存器（vsatp）

vsatp 寄存器是一个 VSXLEN 位读/写寄存器，它是监管级寄存器 satp 在 VS 模式下的版本，其 VSXLEN=32 格式如图 8.32，VSXLEN=64 格式如图 8.33所示。当 V=1 时，vsatp 代替 satp，所以读或修改 satp 的指令实际上会访问 vsatp。vsatp 控制 VS 阶段宾客虚拟地址地址翻译–两阶段翻译中的第一阶段（见第 8.5节）。

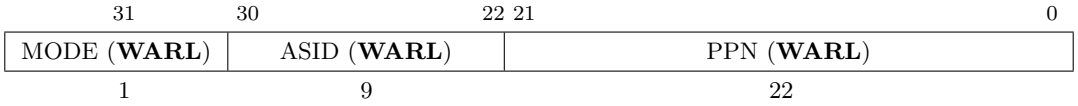


图 8.32: 当 VSXLEN=32 时，虚拟监管级地址翻译和保护寄存器 vsatp

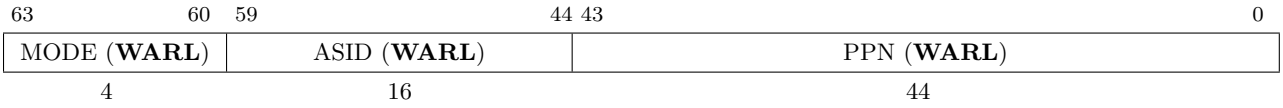


图 8.33: 当 VSXLEN=64 时，裸机、Sv39、Sv48 和 Sv57 模式下，虚拟监管级地址翻译和保护寄存器 vsatp

为了地址翻译算法的目的，vsatp 寄存器被认为是*活跃的*，除非有效的权限模式是 U 并且 hstatus.HU=0。然而，即使当 vsatp 是活跃的，VS 阶段页表实体的 A 位不可以作为推测执行的结构而被设置，除非有效的权限模式是 VS 或 VU。

特别地，虚拟机加载/存储（*HLV*, *HLVX*, 或 *HSV*）指令 *(misspeculatively) In particular, virtual-machine load/store (HLV, HLVX, or HSV) instructions that are misspeculatively executed must not cause VS-stage A bits to be set.*

当 V=0 时，对 `vsatp` 的带有不支持的权限模式值的写操作要么被忽略（就像 `satp` 一样），要么 `vsatp` 的域在平常的方式下被当做 **WARL**。然而，如果 V=1，对 `satp` 的带有不支持的权限模式值的写操作会被忽略，并且不会对 `vsatp` 生效。

当 V=0 时，`vsatp` 不会直接影响机器的行为，除非一个虚拟机加载/存取指令（HLV, HLVX, or HSV）或在 `mstatus` 中的 MPRV 特性被用来执行一个加载或存储操作（就像在 V=1 时）。

8.3 超级监管器指令

超级监管器拓展添加了虚拟机加载和存储指令和两个特权屏障指令。

8.3.1 超级监管级虚拟机加载和存储指令

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
HLV.*width*	[U]	addr	PRIVM	dest	SYSTEM
HLVX.HU/WU	HLVX	addr	PRIVM	dest	SYSTEM
HSV.*width*	src	addr	PRIVM	0	SYSTEM

超级监管器虚拟机加载和存储指令仅在 M 模式或 HS 模式下有效，或在 U 模式下当 `hstatus`.HU=1 时有效。每条指令执行一个显式的内存访问，尽管 V=1; 即，具有地址转换、保护和端序，适用于 vs 模式或 vu 模式下的内存访问。字段 SPVP 的 `hstatus` 控制访问的权限级别。当 SPVP=0 时，显式内存访问就像在 VU 模式下完成，当 SPVP=1 时，就像在 VS 模式下完成。和往常一样，当 V=1 时，应用两级地址转换，HS 级 `sstatus`.SUM 将被忽略。HS 级 `sstatus`.MXR 使得仅执行的页面对于地址转换的两个阶段 (VS 阶段和 G 阶段) 都是可读的，而 `vsstatus`.MXR 只影响第一个翻译阶段 (VS 阶段)。

对于每个 RV32I 或 RV64I 加载指令 LB、LBU、LH、LHU、LW、LWU 和 LD，都有一个对应的虚拟机加载指令：HLV.B、HLV.BU、HLV.H、HLV.HU、HLV.W、HLV.WU 和 HLV.D。对于每个 RV32I 或 RV64I 存储指令 SB、SH、SW 和 SD，都有一个相应的虚拟机存储指令：HSV.B、HSV.H、HSV.W 和 HSV.D。当然，指令 HLV.WU、HLV.D 和 HSV.D 对 RV32 无效。

HLVX.HU 和 HLVX.WU 指令与 HLV.HU 和 HLV.WU 是一样的，除了在地址转换过程中，执行权限取代了读权限。也就是说，被读取的内存必须在地址转换的两个阶段都是可执行的，但不需要读取权限。对于由地址翻译产生的监管级物理地址，监管级物理内存属性必须同时授予执行和写权限。(超级监管器物理内存属性是由监管级物理内存保护（第 3.7节）修改的机器物理内存属性。)

HLVX 不能覆盖机器级物理内存保护 (PMP)，因此试图读取 PMP 指定为仅执行的内存仍然会导致访问错误异常。

尽管 HLVX 指令的显式内存访问需要执行权限，但它们仍然会引发与其他加载指令相同的异常，而不是引发获取异常。

HLVX.WU 对于 RV32 是有效的，即使 LWU 和 HLV 是无效的。(对于 RV32，HLVX.WU 可以被认为是 HLV.W 的变体，因为符号扩展与 32 位值无关。)

当 V=1 时，试图执行虚拟机加载/存储指令 (HLV、HLVX 或 HSV) 会引起虚拟指令陷入。当 `hstatus`.HU=0，试图在 U 模式执行上述指令会引起非法指令陷入。

8.3.2 超级监管器内存管理屏障指令

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
HFENCE.VVMA	asid	vaddr	PRIV	0	SYSTEM
HFENCE.GVMA	vmid	gaddr	PRIV	0	SYSTEM

超级监管器内存管理屏障指令 HFENCE.VVMA 和 HFENCE.GVMA，执行类似 SFENCE.VMA 的功能（第 4.2.1节），但应用于由 CSR `vsatp`（HFENCE.VVMA）控制的 VS 级内存管理数据结构或由 CSR `hgatp`（HFENCE.GVMA）控制的宾客物理内存管理数据结构除外。指令 SFENCE.VMA 只应用于当前 `satp` 控制的内存管理数据结构（当 V=0 时是 HS 级 `satp`，当 V=1 时是 `vsatp`）。

HFENCE.VVMA 仅在 M 模式或 HS 模式下有效。它的效果与临时进入 VS 模式并执行 SFENCE.VMA 非常相似。执行 HFENCE.VVMA 可以确保，当前硬件线程已经可见的任何先前的存储操作，在所有隐式读取指令的 VS 阶段地址翻译之前已经被完成。这些指令如下：

- 在 HFENCE.VVMA 之后的指令，和
- 当 `hgatp`.VMID 有与执行 HFENCE.VVMA 相同效果的设置时，执行的指令。

当 `hgatp`.VMID 与 HFENCE.VVMA 执行的时候不同时，隐式读取不需要排序。如果操作数 *rs1*≠`x0`，它指定一个宾客虚拟地址，如果操作数 *rs2*≠`x0`，它指定一个宾客地址空间标识符 (ASID)。

一个 HFENCE.VVMA 指令只应用于一个虚拟机，当 HFENCE.VVMA 执行时，通过设置 `hgatp`.VMID 标识当前虚拟机。

当 *rs2*≠x0 时，*rs2* 中保存的值的位 XLEN-1:ASIDMAX 保留以供将来标准使用。在标准拓展定义它们的使用之前，它们应该被软件归零，并被当前的实现忽略。此外，如果 ASIDLEN < ASIDMAX，实现将忽略 *rs2* 中保存的值的 ASIDMAX-1:ASIDLEN 位。

HFENCE.VVMA 简单的实现可以忽略 rs1 中的宾客虚拟地址和 rs2 中的宾客 ASID 值，以及 hgatp.VMID，并始终为所有虚拟机的 VS 级内存管理执行全局屏障，甚至为所有内存管理数据结构执行全局屏障。

mstatus.TVM 和 hstatus.VTVM 都不会引起陷入。

HFENCE.GVMA 仅在 HS 模式下当 mstatus.TVM=0 时，或在 M 模式下（不考虑 mstatus.TVM）有效。执行一个 HFENCE.GVMA 指令保证当前硬件线程已经可见的任何先前的存储，在该硬件线程为遵循 HFENCE.GVMA 的指令进行 G 阶段地址翻译的所有隐式读取之前被排序。如果操作数 *rs1*≠x0，它指定一个宾客物理地址，并右移 2 位；如果操作数 *rs2*≠x0，它指定一个虚拟机标识符 (VMID)。

从概念上讲，一个实现可能包含两个地址转换缓存：一个将宾客虚拟地址映射到宾客物理地址，另一个将宾客物理地址映射到监管级物理地址。HFENCE.GVMA 不需要刷新前一个缓存，但它必须从后一个缓存中刷新与 HFENCE.GVMA 地址和 VMID 参数相匹配的条目。

更常见的是，实现包含地址翻译的缓存，它将宾客虚拟地址直接映射到监管级物理地址，消除了某种程度的间接。对于这种实现，其宾客虚拟地址映射到与 HFENCE.GVMA 的地址和 VMID 参数匹配的宾客物理地址的任何条目必须被刷新。以这种方式有选择地刷新条目需要用宾客物理地址标记它们，这代价很高。因此常用的技术是刷新与 HFENCE.GVMA 的 VMID 参数匹配的所有条目，而不理会地址参数。

就像在陷入中写入 htval 的宾客物理地址一样，在 rs1 中指定的宾客物理地址将右移 2 位，以容纳比当前 XLEN 更宽的地址。

当 *rs2*≠x0 时，*rs2* 中保存的值的位 XLEN-1:VMIDMAX 保留以供将来标准使用。在标准扩展定义它们的使用之前，它们应该被软件归零，并被当前的实现忽略。此外，如果 VMIDLEN < VMIDMAX，则实现将忽略 *rs2* 中保存的值的位 VMIDMAX-1:VMIDLEN。

HFENCE.GVMA 的简单实现可以忽略 rs1 中的宾客物理地址和 rs2 中的 VMID 值，并始终为所有虚拟机的宾客物理内存管理执行全局隔离，甚至为所有内存管理数据结构执行全局隔离。

如果 hgatp.MODE 对于给定的 VMID 被更改，HFENCE.GVMA 必须执行带有 *rs1*=x0(以及 *rs2* 设置为 x0 或 VMID) 的 GVMA，以便命令带有 MODE 更改的后续宾客翻译——即使旧 MODE 或新 MODE 为 Bare。

试图执行 HFENCE.VVMA，或在 V=1 时执行 HFENCE.GVMA 会引起一个虚拟指令陷入，而在 U 模式下尝试做同样的事情会导致非法指令陷入。在 HS 模式下并且 `mstatus`.TVM=1 时，试图执行 HFENCE.GVMA 也会导致非法指令陷入。

8.4 机器级 CSR

超级监管器拓展增加或修改了机器级 CSR `mstatus`、`mstatush`、`mideleg`、`mip` 和 `mie`，并添加了 CSR `mtval2` 和 `mtinst`。

8.4.1 机器级状态寄存器（`mstatus` 和 `mstatush`）

超级监管器拓展将两个字段 MPV 和 GVA 添加到机器级别的 `mstatus` 或 `mstatush` CSR 中，并修改几个现有的 `mstatus` 字段的行为。图 8.34显示了在实现了超级监管器拓展并且 MXLEN=64 时，修改的 `mstatus` 寄存器。当 MXLEN=32 时，超级监管器拓展将 MPV 和 GVA 添加到 `mstatush` 而不是 `mstatus`。图 8.35显示了在实现了超级监管器拓展并且 MXLEN=32 时的 `mstatush` 寄存器。

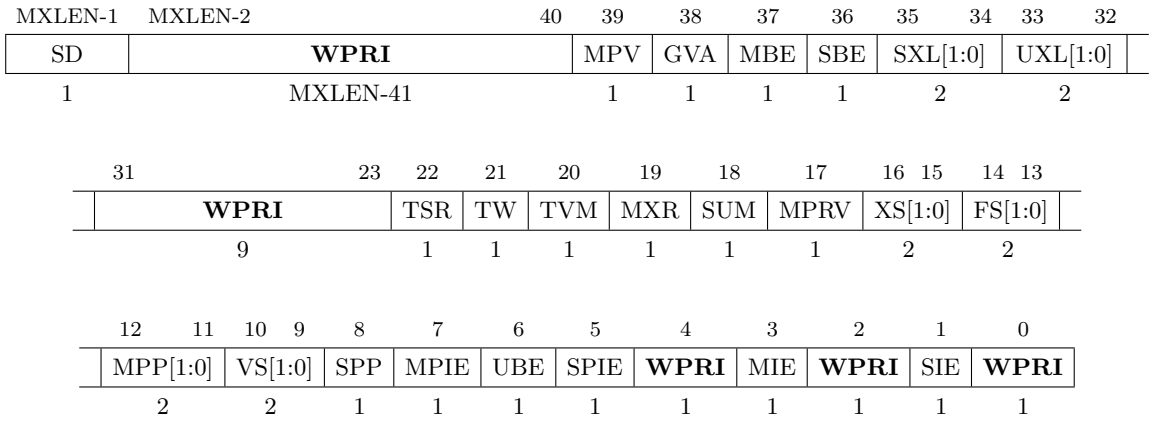


图 8.34: 当实现了超级监管器拓展时，RV64 下的机器状态寄存器（`mstatus`）。

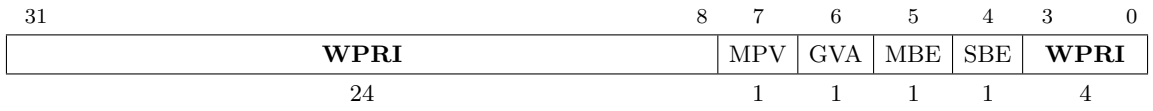


图 8.35: 当实现了超级监管器拓展时，RV32 下的额外的机器状态寄存器（`mstatush`）。RV32 下的 `mstatus` 格式并未改变。

MPV 位 (机器以前的虚拟化模式) 在陷入进入 M 模式时写入。就像 MPP 字段在陷入发生时被设置为 (名义上的) 特权模式一样，MPV 位在陷入发生时被设置为虚拟模式 V 的值。当 MRET 指令被执行时，虚拟模式 V 被设置为 MPV，除非 MPP=3，在这种情况下 V 保持为 0。

字段 GVA (Guest Virtual Address) 在陷入进入 M 模式时由实现写入。对于将宾客虚拟地址写入 `mtval` 的任何陷入 (断点、地址不对齐、访问错误、页面错误或宾客页面错误)，GVA 被设置为 1。对于进入 M 模式的其他陷入, GVA 被设置为 0。

`mstatus` 的 TSR 和 TVM 字段只影响 HS 模式下的执行，而不影响 VS 模式下的执行。TW 字段影响除 M 模式外的所有模式的执行。

设置 TVM=1 可防止 HS 模式访问 `hgatp` 或执行 HFENCE.GVMA 或 HINVAL.GVMA，但对访问 `vsatp` 或指令 HFENCE.VVMA 或 HINVAL.VVMA 没有影响。

TVM 存在于 `mstatus` 中，允许机器级软件修改由监管级操作系统管理的地址翻译过程，通常是为了在操作系统控制的地址翻译下面插入另一个阶段的地址转换。由 TVM=1 启用的指令陷入允许机器级合并 `satp` 和 `hgatp`，并替换影子页表 (将操作系统选择的页翻译与 M 级的低级阶段 (lower-stage) 翻译合并)，这些操作系统都不会感知到。M 级软件不仅需要这种能力来模拟尚未被支持的管理程序扩展，而且还需要模拟任何可能修改或添加地址翻译阶段的未来RISC-V扩展，例如：对嵌套超级监管器程序的改进支持，即在其他超级监管器之上运行超级监管器。

然而，设置 TVM=1 不会导致访问 `vsatp` 或着 HFENCE.VVMA 或 HINVAL.VMA 指令的陷入，或者在 VS 模式下采取的任何操作，因为 M 级软件不需要涉及 VS 阶段地址转换。对于虚拟机来说，不去管 VS 阶段的地址翻译，而将所有其他的翻译阶段合并到由 `hgatp` 控制的 G 阶段影子页表中，应该就足够了，而且很可能更快。这种假设确实对当前机器能够有效模拟的未来可能的RISC-V扩展施加了一些限制。

超级监管器拓展更改 `mstatus` 的修改权限字段 MPRV 的行为。当 MPRV=0 时，翻译和保护行为正常。当 MPRV=1 时，显式内存访问被转换和保护，并应用端序，就好像当前虚拟模式被设置为 MPV，当前名义特权模式被设置为 MPP。表 8.5列举了这些例子。

MPRV 不会影响虚拟机加载/存储指令、HLV、HLVX 和 HSV。这些指令的显式加载和存储总是像 V=1，并且名义上的特权模式是 `hstatus`.SPVP 一样，覆盖了 MPRV。

`mstatus` 寄存器是 HS 级 `sstatus` 寄存器的超集，但不是 `vsstatus` 的超集。

MPRV	MPV	MPP	影响
0	–	–	一般访问; 当前特权模式应用。Normal access; current privilege mode applies.
1	0	0	只带有 HS 级别翻译和保护的 U 级访问。U-level access with HS-level translation and protection only.
1	0	1	只带有 HS 级别翻译和保护的 HS 级访问。HS-level access with HS-level translation and protection only.
1	–	3	没有翻译的 M 级别访问。M-level access with no translation.
1	1	0	具有两级转换和保护的 VU 级访问。HS 级 MXR 位使任何可执行页面都可读。`vsstatus`.MXR 使那些在 VS 翻译阶段标记为可执行的页面具有可读性，但只有在宾客物理翻译阶段才具有可读性。VU-level access with two-stage translation and protection. The HS-level MXR bit makes any executable page readable. `vsstatus`.MXR makes readable those pages marked executable at the VS translation stage, but only if readable at the guest-physical translation stage.
1	1	1	具有两级转换和保护的 VS 级访问。HS 级 MXR 位使任何可执行页面都可读。`vsstatus`.MXR 使那些在 VS 翻译阶段标记为可执行的页面具有可读性，但只有在宾客物理翻译阶段才具有可读性。`vsstatus`.SUM 代替 HS 级 SUM 位。VS-level access with two-stage translation and protection. The HS-level MXR bit makes any executable page readable. `vsstatus`.MXR makes readable those pages marked executable at the VS translation stage, but only if readable at the guest-physical translation stage. `vsstatus`.SUM applies instead of the HS-level SUM bit.

表 8.5: 在显示内存访问的翻译和保护中，MPRV 的影响。

8.4.2 机器中断代理寄存器（`mideleg`）

当实现超级监管器拓展时，`mideleg` 的第 10、6 和 2 位 (对应于标准 VS 级中断) 都是只读的。此外，如果实现了任何宾客外部中断 (GEILEN 是非零)，`mideleg` 的 12 位 (对应于监管级宾客外部中断) 也是只读的。VS 级中断和宾客外部中断总是从 M 模式委托到 HS 模式。

对于 `mideleg` 为零的位，`hideleg`、`hip` 和 `hie` 中相应的位为只读零。

8.4.3 机器中断寄存器（mip 和 mie）

超级监管器拓展为超级监管器添加的中断提供了寄存器 mip 和 mie 额外的活动位。图8.36和8.36显示了实现了超级监管器拓展时，寄存器 mip 和 mie 的标准部分 (位 15:0)。

15 — 13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SGEIP	MEIP	VSEIP	SEIP	0	MTIP	VSTIP	STIP	0	MSIP	VSSIP	SSIP	0
3	1	1	1	1	1	1	1	1	1	1	1	1	1

图 8.36: mip 的标准部分（0-15 位）。

15 — 13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SGEIE	MEIE	VSEIE	SEIE	0	MTIE	VSTIE	STIE	0	MSIE	VSSIE	SSIE	0
3	1	1	1	1	1	1	1	1	1	1	1	1	1

图 8.37: mie 的标准部分（0-15 位）。

mip 中的 SGEIP、VSEIP、VSTIP 和 VSSIP 位是超级监管器 CSR hip 中相同位的别名，而 mie 中的 SGEIE、VSEIE、VSTIE 和 VSSIE 位是 hie 中相同位的别名。

8.4.4 机器第二陷入值寄存器（mtval2）

mtval2 寄存器是一个 MXLEN 位的读/写寄存器，其格式如图 8.38所示。当一个陷入进入 M 模式时，mtval2 与 mtval 一起被写入额外的异常相关的信息，以协助软件处理该陷入。

MXLEN-1 — 0
mtval2
MXLEN

图 8.38: 机器第二陷入值寄存器（mtval2）。

当宾客页面错误异常进入 M 模式时，mtval2 用 0 或出错的宾客物理地址右移 2 位写入。对于其他的陷入，mtval2 被设置为零，但是未来的标准或扩展可能会为其他的陷入重新定义 mtval2 的设置。

如果宾客页面错误是由于第一阶段 (VS 阶段) 地址翻译期间的隐式内存访问造成的，那么写入 mtval2 的宾客物理地址就是发生错误的隐式内存访问的物理地址。CSR mtinst 中提供了其他信息来消除这种情况的歧义。

否则，对于导致宾客页面错误的加载和存储错误，mtval2 中的非零宾客物理地址对应于由 mtval 中的虚拟地址指示的访问错误部分。对于具有变长指令的系统上的指令宾客页面错误，非零 mtval2 对应于由 mtval 中的虚拟地址表示的指令的错误部分。

mtval2 是一个 **WARL** 寄存器，它必须能够容纳零，并且可能只能容纳其他动移 2 位的宾客物理地址的任意子集 (如果有的话)。

8.4.5 机器陷入指令寄存器（mtinst）

mtinst 寄存器是一个 MXLEN 位的读/写寄存器，其格式如图 8.39所示。当一个陷入进入 M 模式时，mtinst 被写入一个值，如果该值非零，则提供陷入指令的信息，以协助软件处理该陷入。可能在陷入中写入 mtinst 的值记录在 Section 8.6.3中。

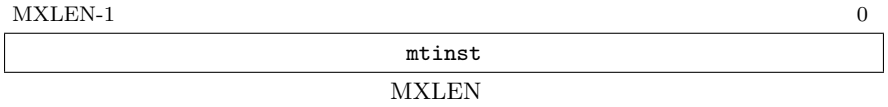


图 8.39: 机器陷入指令寄存器（mtinst）。

mtinst is a **WARL** register that need only be able to hold the values that the implementation may automatically write to it on a trap.

8.5 两阶段地址翻译

如果当前的虚拟模式 V 是 1 时，两阶段地址翻译和保护就开始生效。对于任何虚拟内存访问，原始的虚拟地址是在第一阶段被转化为*宾客物理地址*。这个 VS 级地址翻译过程被被 vsatp 寄存器控制。宾客物理地址是在第二阶段被转化为监管级物理地址。这个宾客物理地址翻译的过程被 hgatp 寄存器控制。这两个阶段也被称为 VS 阶段翻译和 G 阶段翻译。尽管当 V=1 时没有机会可以关掉两阶段地址翻译，但是可以通过为相应的控制寄存器（vsatp 或 hgatp）置零，有效的关掉相对应的翻译阶段。

vsstatus 的 MXR 域（让只是可执行的页可读）只会覆盖 VS 阶段页保护。在 VS 级别设置 MXR 不会覆盖宾客物理页保护。然而，在 HS 级别设置 MXR，会覆盖 VS 阶段和 G 阶段的只可执行权限。

当 V=1 时，平常绕过地址翻译的内存访问只受 G 阶段内存翻译的约束。这包含支持 VS 阶段地址翻译的内存访问，例如读写 VS 级别页表。

机器级物理内存保护应用于监管级物理地址，并且在任何虚拟模式下都生效。

8.5.1 宾客物理地址翻译

从宾客物理地址到监管级物理地址的映射被 `hgatp` 控制状态寄存器所控制。（第 8.2.10节）。

当被 `hgatp` 的 MODE 域选择的地址翻译方案是裸机（Bare）时，宾客物理地址与监管级物理地址一致，没有任何修改，并且没有内存保护应用在这个微小的从宾客物理地址到监管级物理地址的翻译过程。

当 `hgatp`.MODE 指定了一个翻译模式（Sv32x4、Sv39x4、Sv48x4 或 Sv57x4），G 阶段地址翻译是普通基于页的虚拟地址翻译方案（Sv32、Sv39、Sv48 或 Sv57）的一个变体。在这种情况下，输入地址的大小被拓宽两位（变成 34，41，50，或 59 位）。为了容纳额外的 2 位，根页表 (仅) 扩展了 4 倍，为 16KiB，而不是通常的 4KiB。为了匹配其更大的尺寸，根页表还必须对齐到 16KiB 边界，而不是通常的 4KiB 页面边界。除特别说明外，所有其他方面的 Sv32、Sv39、Sv48 或 Sv57 均不变地用于 G 阶段翻译。非根页表和所有页表实体（PTE）有相同的格式，描述在第4.3，4.4，4.5，和 4.6节。

对于 Sv32x4 而言，传入的宾客物理地址被划分为一个虚拟页号（VPN）和页偏移，如图 8.40所示。这种划分方式与图 4.31（页 100）描述的 Sv32 虚拟地址相同，除了在 VPN[1] 高位的多了两位。(注意：宾客物理地址划分的域也与 Sv32 分配给物理地址的结构一一对应，如图 4.32所示。)

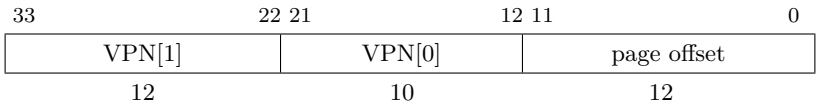


图 8.40: Sv32x4 虚拟地址（宾客物理地址）。

对于 Sv39x4 而言，传入的宾客物理地址被划分为如图 8.41所示。这种划分方式与图 4.37（页 112）描述的 Sv39 虚拟地址相同，除了在 VPN[2] 高位的多了两位。地址 63:41 位必须全部为零，否则一个宾客页错误异常将会发生。

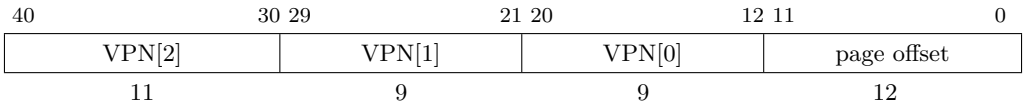


图 8.41: Sv39x4 虚拟地址（宾客物理地址）。

对于 Sv48x4 而言，传入的宾客物理地址被划分为如图 8.42所示。这种划分方式与图 4.43（页 114）描述的 Sv39 虚拟地址相同，除了在 VPN[3] 高位的多了两位。地址 63:50 位必须全部为零，否则一个宾客页错误异常将会发生。

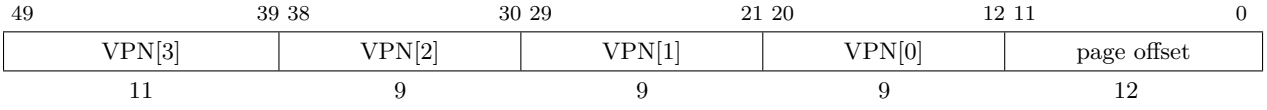


图 8.42: Sv48x4 虚拟地址（宾客物理地址）。

对于 Sv57x4 而言，传入的宾客物理地址被划分为如图 8.43所示。这种划分方式与图 4.49（页 116）描述的 Sv39 虚拟地址相同，除了在 VPN[4] 高位的多了两位。地址 63:59 位必须全部为零，否则一个宾客页错误异常将会发生。

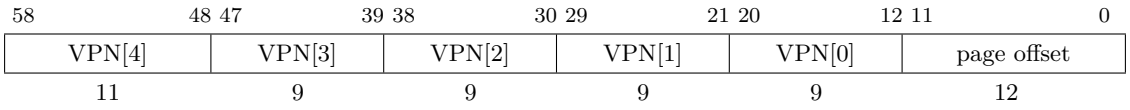


图 8.43: Sv57x4 虚拟地址（宾客物理地址）。

RV32 的基于页面的 G 阶段地址转换方案–Sv32x4，被定义为支持 34 位宾客物理地址，因此 RV32 超级监管器虚拟化真实 32 位 RISC-V 计算机的能力不需要受到限制，即使是那些具有 33 位或 34 位物理地址的计算机。如果它碰巧使用 33 位或 34 位物理地址，这可能包括机器虚拟化自身的可能性。将根页表的大小和对齐方式乘以 4 是扩展 Sv32 以覆盖 34 位地址的最简便的方法。对于一个不必要的大根页表，可能的 12KiB 的浪费对于大多数 (可能所有) 实际使用的后果是微不足道的。

对于拥有四倍于虚拟地址空间的物理地址空间的虚拟机，一致的虚拟化能力被认为对 RV64 也有一定的用处。例如，对于实现 39 位虚拟地址 (Sv39) 的机器，这允许超级监管器拓展支持最多 41 位的宾客物理地址空间，而不需要对 48 位虚拟地址 (Sv48) 提供硬件支持，也不需要使用影子页表来模拟更大的地址空间。

Sv32x4、Sv39x4、Sv48x4 或 Sv57x4 宾客物理地址的转换，使用与 Sv32、Sv39、Sv48 或 Sv57 相同的算法来完成，如第 4.3.2节所示，除了:

- `hgatp` 替代了通常的 `satp`;
- 只有当有效的特权级别为 VS 模式或 VU 模式时，地址翻译才可以开始。
- 在检查 U 位时，当前特权模式始终被为 U 模式；而且
- 引发宾客页面错误异常，而不是普通的页面错误异常。

对于 G 级地址转换，所有内存访问 (包括为 VS 阶段地址转换而访问数据结构的访问) 都被认为是用户级访问，就像在 U 模式下执行一样。访问类型权限——可读、可写或可执行——在 G 阶段的翻译中被检查（与 VS 阶段的翻译相同）。对于支持 VS 阶段地址转换 (例如读/写 VS 级页表) 的内存访问，检查权限就像检查加载或存储，而不是检查原始访问类型。但是，对于原始访问类型 (指令、加载或存储/AMO)，总是报告任何异常。

所有 G 阶段的 PTE 中的 G 位都是为将来的标准使用而保留的。在它的使用被标准扩展定义之前，为了向前兼容，它应该被软件清除，并且必须被硬件忽略。

G 级地址转换使用与常规地址转换相同的 PTE 格式，甚至包括 U 位，因为 G 阶段翻译和常规 HS 级地址转换之间可能共享一些 (或所有) 页表。不管这种用法是否会变得普遍，我们选择不排除它。

8.5.2 宾客页错误

宾客页错误陷入在 `medeleg` 的控制可以从 M 模式代理到 HS 模式，但是不能代理到其他特权模式。在宾客页错误中，CSR `mtval` 或 `stval` 像往常一样将出错的宾客虚拟地址写入，而 `mtval2` 或 `htval` 则将零或出错客户物理地址写入后，右移两位。CSR `mtinst` 或 `htinst` 同时也可能将出错指令或访问的其他原因的信息写入，就像在第 8.6.3节解释的那样。

当指令获取或未对齐的内存访问跨越页面边界时，将会涉及到两种不同的地址转换。当客户页错误出现在这种情况下时，写入 `mtval/stval` 的错误虚拟地址与普通页面错误所需的虚拟地址相同。因此，如果页边界上的字节在被访问的字节中，则出错的虚拟地址可能是高于指令原始虚拟地址的页边界地址。

当一个宾客页错误不是由于在 VS 阶段地址翻译的隐式内存访问所导致时，写入 `mtval2/htval` 的非零宾客物理地址应与写入 `mtval/stval` 的确切虚拟地址相对应。

8.5.3 内存管理屏障

SFENCE.VMA 指令的行为受当前虚拟模式 V 的影响。当 V=0 时，虚拟地址参数是一个 HS 级虚拟地址，ASID 参数是一个 HS 级 ASID。指令序列只存储到带有随后的 HS 级地址翻译的 HS 级地址翻译结构中。

当 V=1 时，SFENCE.VMA 的虚拟地址参数是当前虚拟机的宾客虚拟地址，其 ASID 参数是在当前虚拟机的 VS 级 ASID。当前虚拟机由 CSR `hgatp` 的 VMID 字段标识，有效的 ASID 可以认为是该 VMID 与 vs 级 ASID 的组合。SFENCE.VMA 指令序列仅存储到，在同一虚拟机中，具有后续的 VS 阶段地址的，VS 级地址转换结构中，例如：仅当 `hgatp`.VMID 与 SFENCE.VMA 被执行时才相同。

超级监管器指令 HFENCE.VVMA 和 HFENCE.GVMA 提供额外的内存管理屏障，去补充 SFENCE.VMA。这些指令在第 8.3.2节中描述。

第 3.7.2节讨论了物理内存保护（PMP）和基于页地址翻译的交互过程。需要注意的是，当 PMP 设置的修改影响到保存页表的物理内存或页表所指向的物理内存时，M 模式软件必须将 PMP 设置与虚拟内存系统同步。对于 HS 级别的地址转换，是通过在 M 模式下执行 SFENCE.VMA 指令（使 *rs1*=x0 和 *rs2*=x0）来完成的。还需要同步 G 阶段和 VS 阶段的数据结构。执行一个 HFENCE.GVMA 指令（*rs1*=x0 和 *rs2*=x0）足以刷新所有 G 阶段和 VS 阶段地址翻译缓存项，这些缓存项已经缓存了与最终已翻译的监管级物理地址对应的 PMP 设置。一个 HFENCE.VVMA 指令不是必需的。

8.6 陷入

8.6.1 陷入原因编码

超级监管器拓展扩展了陷入原因编码。表 8.6列出了在实现超级监管器拓展时可能的 M 模式和 HS 模式陷入原因的代码。为 VS 级中断 (中断 2、6、10)、监管级宾客外部中断 (中断 12)、虚拟指令异常 (异常 22) 和宾客页面错误 (异常 20、21、23) 添加了代码。此外，来自 VS 模式的环境调用被分配为原因 10，而来自 HS 模式或 S 模式的环境调用通常使用原因 9。

HS 模式和 VS 模式的 ECALL 用不同的原因值，所以他们就可以被分别代理。

当 V=1 时，如果尝试的指令是 *HS-qualified*，则通常会引发一个虚拟指令异常 (代码 22)，而不是一个非法指令异常，但当 V=1 时，由于特权不足或由于指令被监管器或超级监管器 CSR(例如 `scounteren` 或 `hcounteren` 明确禁用，该指令将无法执行。假设 CSR `mstatus` 的 TSR 和 TVM 字段都为零，如果一条指令在 HS 模式下执行 (对于指令的寄存器操作数的某些值) 是有效的，那么它就是 *HS-qualified*。

特殊规则适用于访问32-位高半分 CSR 的 CSR 指令，如 `cycleh` 和 `htimedeltah`。当 V=1 和 XLEN>32 时，尝试访问 high-half 监管级 CSR、high-half 超级监管级 CSR、high-halfVS 级 CSR 或 high-half 非特权 CSR 总是会引发非法指令异常。在 vs 模式中，如果 VU 模式的 XLEN 大于 32，则尝试访问 high-half 用户级的 CSR(与非特权的 CSR 不同) 总是会引发非法指令异常。另一方面，当 V=1 和 XLEN=32 时，如果其另一半 *low-half* CSR(例如 `cycle` 或 `htimedelta`) 的相同 CSR 指令是 HS-qualified，那么访问 high-half 的 S 级、超级监管器、VS 或非特权 CSR 的无效尝试将引发虚拟指令异常，而不是非法指令异常。同样，在 VS 模式中，如果 VU 模式的 XLEN 为 32，如果其另一半 low-half 的 CSR 的相同 CSR 指令是 HS-qualified，则访问 high-half 用户级 CSR 的无效尝试将引发虚拟指令异常，而不是非法指令异常。

RISC-V 特权体系结构目前没有定义用户级 CSR，但它们可能会通过该标准的未来版本或扩展添加。

中断	异常编码	描述
1	0	保留的
1	1	监管级软件中断 Supervisor software interrupt
1	2	虚拟监管级软件中断 Virtual supervisor software interrupt
1	3	机器软件中断 Machine software interrupt
1	4	保留的
1	5	监管级时钟中断 Supervisor timer interrupt
1	6	虚拟监管级时钟中断 Virtual supervisor timer interrupt
1	7	机器时钟中断 Machine timer interrupt
1	8	保留的
1	9	监管级外部中断 Supervisor external interrupt
1	10	虚拟监管级外部中断 Virtual supervisor external interrupt
1	11	机器外部中断 Machine external interrupt
1	12	监管级宾客外部中断 Supervisor guest external interrupt
1	13–15	保留的
1	≥16	指定用于平台或定制使用
0	0	指令地址未对齐 Instruction address misaligned
0	1	指令访问错误 Instruction access fault
0	2	非法指令 Illegal instruction
0	3	断点 Breakpoint
0	4	加载地址未对齐 Load address misaligned
0	5	加载访问错误 Load access fault
0	6	存储/AMO 地址未对齐 Store/AMO address misaligned
0	7	存储/AMO 访问错误 Store/AMO access fault
0	8	在 U 模式或 VU 模式发生的环境调用 Environment call from U-mode or VU-mode
0	9	在 HS 模式发生的的环境调用 Environment call from HS-mode
0	10	在 VS 模式发生的环境调用 Environment call from VS-mode
0	11	在 M 模式发生的环境调用 Environment call from M-mode
0	12	指令页错误 Instruction page fault
0	13	加载页错误 Load page fault
0	14	保留的
0	15	存储/AMO 页错误 Store/AMO page fault
0	16–19	保留的
0	20	指令宾客页面错误 Instruction guest-page fault
0	21	加载宾客页面错误 Load guest-page fault
0	22	虚拟指令 Virtual instruction
0	23	存储/AMO 宾客页面错误 Store/AMO guest-page fault
0	24–31	为自定义指定
0	32–47	保留的
0	48–63	为自定义指定
0	≥64	保留的

具体来说，在以下情况下引发虚拟指令异常:

- 在 VS 模式下，当 `hcounteren` 中的相应位为 0 且 `mcounteren` 中的相同位为 1 时，尝试访问 non-high-half 计数器 CSR；
- 在 VS 模式下，如果 XLEN=32，当 `hcounteren` 中的相应位为 0 且 `mcounteren` 中的相同位为 1 时，尝试访问 high-half 计数器 CSR；
- 在 VU 模式下，当 `hcounteren` 中的相应位为 0 且 `mcounteren` 中的相同位为 1 时，尝试访问 non-high-half 计数器 CSR；
- 在 VU 模式下，如果 XLEN=32，当 `hcounteren` 中的相应位为 0 且 `mcounteren` 中的相同位为 1 时，尝试访问 high-half 计数器 CSR；
- 在 VS 模式或 VU 模式下,试图去执行一个超级监管级指令(HLV,HLVX,HSV,或 HFENCE);
- 在 VS 模式或 VU 模式下，假设 `mstatus`.TVM=0，当在 HS 模式下允许相同的访问 (读/写) 时，尝试访问一个已被实现的 non-high-half 超级监管级 CSR 或 VS 级 CSR；
- 在 VS 模式或 VU 模式下，假设 `mstatus`.TVM=0，如果 XLEN=32，当在 HS 模式下允许对 CSR 的 low-half 的另一半进行相同的访问 (读/写) 时，尝试访问已实现的 high-half 超级监管器 CSR 或 high-halfVS 级 CSR；
- 在 VU 模式下,当 `mstatus`.TV=0 时尝试执行 WFI 或执行监管级指令 (SRET 或 SFENCE)；
- 在 VU 模式下，假设 `mstatus`.TVM=0，当 HS 模式下允许相同的访问 (读/写) 时，尝试访问一个已被实现的 non-high-half 监管级 CSR；
- 在 VU 模式下，假设 `mstatus`.TVM=0，如果 XLEN=32，当在 HS 模式下允许访问 CSR 的 low-half 的另一半时，试图访问实现的 high-half 监管级 CSR;
- 在 VS 模式下，当 `hstatus`.VTW=1 和 `mstatus`.TW=0 时，试图执行 WFI（除非指令在特定于实现的限定时间内完成)；
- 在 VS 模式下，当 `hstatus`.VTSR=1 时，尝试执行 SRET；并且 in VS-mode, attempts to execute SRET when `hstatus`.VTSR=1; and
- 在 VS 模式下，当 `hstatus`.VTVM=1 时，尝试执行 SFENCE.VMA、SINVAL.VMA 或访问 `satp`。

对RISC-V特权体系结构的其他扩展可能会增加在 V=1 时导致虚拟指令异常的环境集。

在虚拟指令陷入中，`mtval` 或 `stval` 的编写方式与非法指令陷入相同。

为了支持嵌套管理程序或出于其他原因，超级监管器必须模拟引发虚拟指令异常的指令，这种情况并不罕见。通常将虚拟指令陷入直接委托给 *HS* 级的应该是机器级，因此非法指令陷入可能首先在 *M* 模式下处理，然后再有条件地（通过软件）委托给 *HS* 级。因此，虚拟指令陷入通常应该比非法指令陷入处理得更快。

当不模拟陷入指令时，超级监管器应该将虚拟指令陷入转换为宾客虚拟机的非法指令异常。

因为 `mstatus` 中的 *TSR* 和 *TVM* 只影响 *S* 模式 *(HS* 模式*)*，所以在 *VS* 模式中确定的异常会忽略它们。

优先级	Exc. Code	描述
Highest	3	指令地址中断 Instruction address breakpoint
		在指令地址翻译过程中：During instruction address translation:
	12, 20, 1	第一个遇到的页错误，First encountered page fault, 宾客页错误，或访问错误
		带有物理地址的指令 With physical address for instruction:
	1	指令访问错误 Instruction access fault
	2	非法指令 Illegal instruction
	22	虚拟指令 Virtual instruction
	0	指令地址未对齐 Instruction address misaligned
	8, 9, 10, 11	环境调用 Environment call
	3	环境断点 Environment break
	3	加载/存储/AMO 地址断点 Load/store/AMO address breakpoint
		可选地：Optionally:
	4, 6	加载/存储/AMO 地址未对齐 Load/store/AMO address misaligned
		在显示内存访问的地址翻译过程中：During address translation for an explicit m
	13, 15, 21, 23, 5, 7	第一次遇到页错误，First encountered page fault, 宾客页错误，或访问错误 gu
		带有显示内存访问的物理地址：With physical address for an explicit memory ac
	5, 7	加载/存储/AMO 访问错误 Load/store/AMO access fault
		如果不是更高的优先级：If not higher priority:
Lowest	4, 6	加载/存储/AMO 地址未对齐 Load/store/AMO address misaligned

表 8.7: 当超级监管器拓展被实现时，同步异常的优先级。

如果一条指令可能引发多个同步异常，则表 8.7递减优先级顺序表示，在 `mcause` 或 `scause` 中采用并报告哪个异常。

8.6.2 陷入实体

当陷入在 HS 模式或 U 模式下发生时，它将进入 M 模式，除非经过 `medeleg` 或 `mideleg` 的委托进入 HS 模式。当陷入发生在 VS 模式或 VU 模式时，它将进入 M 模式，除非经过 `medeleg` 或 `mideleg` 的委托进入 HS 模式，除非经过 `hedeleg` 或 `hideleg` 的进一步委托进入 VS 模式。

当陷入进入 M 模式时，虚拟模式 V 被设置为 0，`mstatus`(或 `mstatush`) 中的字段 MPV 和 MPP 根据表 8.8设置。陷入到 M 模式还会在 `mstatus`/`mstatush` 中写入字段 GVA、MPIE 和 MIE，并写入 CSR `mepc`、`mcause`、`mtval`、`mtval2` 和 `mtinst`。

Previous Mode	MPV	MPP
U-mode	0	0
HS-mode	0	1
M-mode	0	3
VU-mode	1	0
VS-mode	1	1

表 8.8: 在陷入到 M 模式之后，`mstatus`/`mstatush` 的 MPV 和 MPP 字段。当陷入返回，并且 MPP=3 时，忽略 MPV。

当陷入进入 HS 模式时，虚拟模式 V 设置为 0，`hstatus`.SPV 和 `sstatus`.SPP 根据表 8.9设置。如果在发生陷入之前 V 为 1，`hstatus` 中的 SPVP 字段被设置为 `sstatus`.SPP；否则，保持 SPVP 不变。进入 HS 模式的陷入还将字段 GVA 写入 `hstatus`，字段 SPIE 和 SIE 写入 `sstatus`、CSR `sepc`、`scause`、`stval`、`htval` 和 `htinst`。

Previous Mode	SPV	SPP
U-mode	0	0
HS-mode	0	1
VU-mode	1	0
VS-mode	1	1

表 8.9: 在陷入到 HS 模式之后，`hstatus` 的 SPV 字段和 `sstatus` 的 SPP 字段的值。

当陷入进入 VS 模式时，`vsstatus`.SPP 根据表 8.10设置。寄存器 `hstatus` 和 HS 级别 `sstatus` 未被修改，虚拟模式 V 保持 1。进入 VS 模式的陷入还会在 `vsstatus` 中写入字段 SPIE 和 SIE，并写 CSRs `vsepc`、`vscause` 和 `vstval`。

Previous Mode	SPP
VU-mode	0
VS-mode	1

表 8.10: 在陷入到 VS 模式之后，`vsstatus` 的 SPP 字段的值。

8.6.3 为 `mtinst` 或 `htinst` 的转换指令或伪指令

在任何陷入到 M 模式或 HS 模式的陷入中，下面的一个值会被自动写入合适的陷入指令 CSR `mtinst` 或 `htinst` 中:

- 零;
- 陷入指令的转换;
- 一个自定义的值（只在陷入指令是不标准的情况下允许）；或
- 一个特殊的伪指令。

除了当一个伪指令的值是需要的（后面会描述），写入 `mtinst` 或 `htinst` 的值总是可是是零。这表明硬件不会在寄存器中为这个特殊的陷入提供任何信息。

写入陷入指令 *CSR* 的值有两个目的。第一个目的是提高陷入处理程序中指令模拟的速度，一种方式是通过允许处理程序（*handler*）跳过从内存中加载陷入指令，另一种方式是通过避免解码和执行指令的一些工作。第二个目的是通过伪指令提供关于宾客页面错误异常的附加信息，这些异常是由为 *VS* 阶段地址转换而执行的隐式内存访问引起的。

为了最小化硬件的负担，陷入指令的转换被写入而不是原有指令的复制，然而仍旧为陷入处理程序提供了模拟指令所需要的信息的。

在一个中断中，写入到陷入指令寄存器中的值总是零。在一个同步异常中，如果写入一个非零值，则该值应符合以下条件之一:

- 0 位是 `1`，用 `1` 替换 1 位使该值成为标准指令的有效编码。

  在这种情况下，发生陷入的指令与寄存器值指示的指令是同一种类型，并且寄存器值是陷入指令的转换，如后面所定义的。例如，如果 0-1 位是二进制 `11`，寄存器值是标准 LW(加载字) 指令的编码，那么陷入指令就是 LW，寄存器值是发生陷入的 LW 指令的转换。

- 0 位为 `1`，用 `1` 替换 1 位将使该值变为为自定义指令显式指定的指令编码 (不是为未使用的保留编码)。

  这是一个 定制值。发生陷入的指令是非标准指令。本标准对自定义值的解释没有另行规定。This is a *custom value*.

- 该值是稍后定义的一个特殊伪指令，所有这些伪指令的位 0-1 位都等于 00。

这三种情况排除了大量其他可能的值，例如所有 0-1 位等于二进制 10 的值。未来的标准或扩展可能定义额外的情况，从而允许当前被排除的值。软件可以安全地将陷入指令寄存器中不可识别的值视为零。

为了与本标准的未来修订版向前兼容，从 mtinst 或 htinst 解释非零值的软件必须完全验证该值符合上面列出的情况之一。例如，对于 *RV64*，发现 mtinst 的 *0-6* 位是 0000011，*12-14* 位是 010 并不足以证实此情形适用于第一种情况，并且捕获指令是一个标准的 *LW* 指令；相反，软件还必须确认 mtinst 的 *32-63* 位都是零。

未来的标准可能会为 *64* 位 mtinst 定义新的值，这些值在 *32-63* 位中非零，但在 *0-31* 位中可能具有与标准 *RV64* 指令相同的位模式。

与标准指令不同的是，不要求自定义值的指令编码与发生陷入的指令具有相同的“类型”(甚至不要求与陷入指令有任何关联)。

表 **??**显示了可能会针对每个标准异常原因自动写入陷入指令寄存器的值。对于阻止获取指令的异常，只能写入零或伪指令值。只有当发生陷入的指令是非标准的时候，才会自动写入自定义值。未来的标准或扩展可能允许编写其他值，从前面建立的允许值集中选择这些值。

如表中所列举的，同步异常向陷入指令寄存器中写入，只针对显式内存访问 (从 load、store 和 AMO 指令) 引起的异常的陷入指令的标准转换。因此，目前仅为这些内存访问指令定义了标准转换。如果同步陷入发生在没有定义转换的标准指令上，那么陷入指令寄存器应该写为零 (或者，在某些情况下，写为一个特殊的伪指令值)。

Exception	Zero	Transformed Standard Instruction	Custom Value	Pseudo-instruction Value
Instruction address misaligned	Yes	No	Yes	No
Instruction access fault	Yes	No	No	No
Illegal instruction	Yes	No	No	No
Breakpoint	Yes	No	Yes	No
Virtual instruction	Yes	No	Yes	No
Load address misaligned	Yes	Yes	Yes	No
Load access fault	Yes	Yes	Yes	No
Store/AMO address misaligned	Yes	Yes	Yes	No
Store/AMO access fault	Yes	Yes	Yes	No
Environment call	Yes	No	Yes	No
Instruction page fault	Yes	No	No	No
Load page fault	Yes	Yes	Yes	No
Store/AMO page fault	Yes	Yes	Yes	No
Instruction guest-page fault	Yes	No	No	Yes
Load guest-page fault	Yes	Yes	Yes	Yes
Store/AMO guest-page fault	Yes	Yes	Yes	Yes

表 8.11: 在异常陷入中，可能被自动写入陷入指令寄存器（`mtinst` 或 `htinst`）的值。

对于不是压缩指令，而是 LB、LBU、LH、LHU、LW、LWU、LD、FLW、FLD、FLQ 或 FLH 之一的标准加载指令，转换后的指令格式如图 **??**所示。

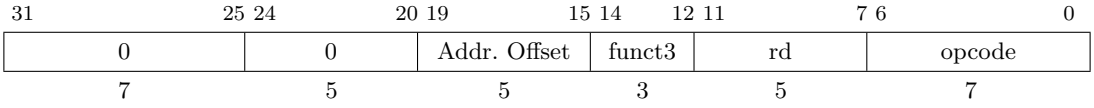


图 8.44: Transformed noncompressed load instruction (LB, LBU, LH, LHU, LW, LWU, LD, FLW, FLD, FLQ, or FLH). Fields funct3, rd, and opcode are the same as the trapping load instruction.

对于不是压缩指令，而是 SB、SH、SW、SD、FSW、FSD、FSQ 或 FSH 之一的标准存储指令，转换后的指令格式如图 **??**所示。

对于标准原子指令 (保留加载、条件存储或 AMO 指令)，转换后的指令格式如图 **??**所示。

对于标准的虚拟机加载/存储指令 (HLV、HLVX 或 HSV)，转换后的指令的格式如图 8.47所示。

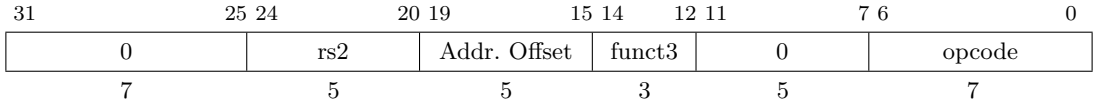


图 8.45: Transformed noncompressed store instruction (SB, SH, SW, SD, FSW, FSD, FSQ, or FSH). Fields rs2, funct3, and opcode are the same as the trapping store instruction.

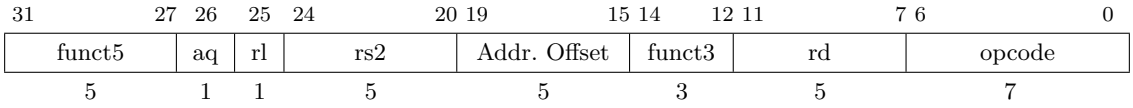


图 8.46: Transformed atomic instruction (load-reserved, store-conditional, or AMO instruction). All fields are the same as the trapping instruction except bits 19:15, Addr. Offset.

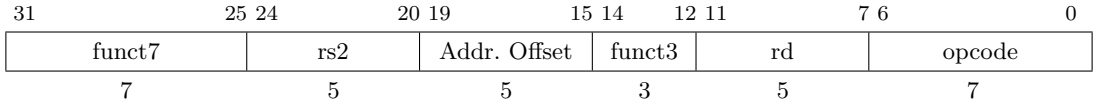


图 8.47: Transformed virtual-machine load/store instruction (HLV, HLVX, HSV). All fields are the same as the trapping instruction except bits 19:15, Addr. Offset.

在以上所有转换的指令中，Addr. Offset 字段是错误虚拟地址（写入 `mtval` 或 `stval`）与原始虚拟地址之间的正差值，它代替了指令 15-19 位的 rs1 字段。只有在非对齐内存访问的情况下，此差异才可能是非零的。还要注意，对于基本的加载和存储，转换将指令的直接偏移字段替换为零。

对于标准压缩指令（16 位长），转换指令如下所示：

1. 拓展压缩指令为等价的 32 位指令。
2. 转换这个 32 位等价指令。
3. 用 a `0` 代替第 1 位。

如果陷入指令被压缩，则转换后的标准指令的第 0-1 位将是二进制 `01`，否则将是 `11`。

在解码 `mtinst` 或 `htinst` 的内容时，一旦软件确定了寄存器包含标准基本加载（*LB, LBU, LH, LHU, LW, LWU, LD, FLW, FLD, FLQ*，或 *FLH*）或基本存储（*SB, SH, SW, SD, FSW, FSD, FSQ*，或 *FSH*）的编码，就不需要确认立即偏移字段（*31:25*，和 *24:20* 或 *11:7*）为零。知道寄存器的值是基本加载/存储的编码就足以证明陷入指令是同类指令。

该标准的未来版本可能会向当前为零的字段添加信息。但是，为了向后兼容，任何此类信息都只用于性能目的，可以安全地忽略。

对于宾客页错误，陷入指令寄存器在以下情况下用一个特殊的伪指令值写入：（a）在 VS 阶段地址翻译过程中，隐式内存访问引起的错误，并且（b）向 `mtval2` 或 `htval` 写入非零值（错误宾客物理地址）。如果两个条件都符合，那么写入 `mtinst` 或 `htinst` 的值必须从表 8.12选择；零是不被允许的。

Value	Meaning
0x00002000	32-bit read for VS-stage address translation (RV32)
0x00002020	32-bit write for VS-stage address translation (RV32)
0x00003000	64-bit read for VS-stage address translation (RV64)
0x00003020	64-bit write for VS-stage address translation (RV64)

表 8.12: Special pseudoinstruction values for guest-page faults. The RV32 values are used when VSXLEN=32, and the RV64 values when VSXLEN=64.

所定义的伪指令值被设计为与基本加载和存储的编码密切对应，如 Table 8.13所示。

Encoding	Instruction
0x00002003	`lw x0,0(x0)`
0x00002023	`sw x0,0(x0)`
0x00003003	`ld x0,0(x0)`
0x00003023	`sd x0,0(x0)`

表 8.13: Standard instructions corresponding to the special pseudoinstructions of Table 8.12.

write 伪指令 (0x00002020 或 0x00003020) 用于机器试图自动更新 VS 级页表中的位 A 和/或 D 的情况。所有其他用于 VS 阶段地址翻译的隐式内存访问都将被读取。如果机器从不自动更新 VS 级页表中的 A 位或 D 位 (将此留给软件)，则永远不会出现 *write* 情况。这样的页表更新实际上必须是原子的，而不仅仅是一个简单的写入，这一事实在伪指令中被忽略了。

如果需要伪指令值的条件可以出现在 *M* 模式中，那么 `mtinst` 不可能完全只读为零；*HS-mode* 和 `htinst` 也是如此。然而，在这种情况下，陷入指令寄存器在只需要一个或两个硬件触发器，就可能最低限度地支持值 *0* 和 0x00002000 或 0x00003000，可能还支持 0x00002020 或 0x00003020，

忽略页表更新的原子性要求在这里没有什么坏处，因为在这些情况下，超级监管器不希望模拟失败的隐式内存访问。相反，在通过重试错误指令恢复执行之前，超级监管器被提供了足够的关于错误访问的信息，以便能够使内存被访问（例如通过恢复虚拟内存的一个缺失页）。

8.6.4 陷入返回

MRET 指令用于进入 M 模式的陷入的返回。MRET 首先根据 `mstatus` 或 `mstatush` 中的 MPP 和 MPV 的值（其编码在表 8.8中）确定新的特权模式是什么。然后，MRET 在 `mstatus`/`mstatush` 中设置 MPV=0, MPP=0, MIE=MPIE, MPIE=1。最后，MRET 将特权模式设置成先前决定的特权模式，并设置 `pc`=`mepc`。

SRET 指令用于进入 HS 模式或 VS 模式的陷入返回。它的行为取决于当前的虚拟模式。

当在 M 模式或 HS 模式 (即 V=0) 执行时，SRET 首先根据 `hstatus`.SPV 和 `sstatus`.SPP 中的值（其编码在表 8.9中）确定新的特权模式是什么。然后 SRET 设置 `hstatus`.SPV=0，在 `sstatus` 中设置 SPP=0, SIE=SPIE, SPIE=1。最后，SRET 将特权模式设置成先前决定的特权模式，并设置 `pc`=`sepc`。

当 SRET 以 VS 模式 (即 V=1) 执行时，根据表 8.10设置特权模式，在 `vsstatus` 中设置 SPP=0, SIE=SPIE, SPIE=1，最后设置 `pc`=`vsepc`。

第九章　RISC-V 特权指令集列表

本章介绍了 RISC-V 特权架构中定义的所有指令的指令集列表。

本手册第一卷提供了非特权指令的指令集列表，包括 ECALL 和 EBREAK 指令。

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-类型
imm[11:0]			rs1	funct3	rd	opcode	I-类型

陷入-返回指令

0001000	00010	00000	000	00000	1110011	SRET
0011000	00010	00000	000	00000	1110011	MRET

中断-管理指令

0001000	00101	00000	000	00000	1110011	WFI

监管器内存-管理指令

0001001	rs2	rs1	000	00000	1110011	SFENCE.VMA

超级监管器内存-管理指令

0010001	rs2	rs1	000	00000	1110011	HFENCE.VVMA
0110001	rs2	rs1	000	00000	1110011	HFENCE.GVMA

超级监管器虚拟-机器加载和保存指令

0110000	00000	rs1	100	rd	1110011	HLV.B
0110000	00001	rs1	100	rd	1110011	HLV.BU
0110010	00000	rs1	100	rd	1110011	HLV.H
0110010	00001	rs1	100	rd	1110011	HLV.HU
0110100	00000	rs1	100	rd	1110011	HLV.W
0110010	00011	rs1	100	rd	1110011	HLVX.HU
0110100	00011	rs1	100	rd	1110011	HLVX.WU
0110001	rs2	rs1	100	00000	1110011	HSV.B
0110011	rs2	rs1	100	00000	1110011	HSV.H
0110101	rs2	rs1	100	00000	1110011	HSV.W

超级监管器虚拟-机器加载和存储指令，仅 RV64

0110100	00001	rs1	100	rd	1110011	HLV.WU
0110110	00000	rs1	100	rd	1110011	HLV.D
0110111	rs2	rs1	100	00000	1110011	HSV.D

***Svinval* 内存-管理拓展**

0001011	rs2	rs1	000	00000	1110011	SINVAL.VMA
0001100	00000	00000	000	00000	1110011	SFENCE.W.INVAL
0001100	00001	00000	000	00000	1110011	SFENCE.INVAL.IR
0010011	rs2	rs1	000	00000	1110011	HINVAL.VVMA
0110011	rs2	rs1	000	00000	1110011	HINVAL.GVMA

表 9.1: RISC-V 特权指令

第十章　历史

10.1　加州大学伯克利分校的研究经费

RISC-V 架构和实现的开发部分由以下赞助商资助。

- **Par 实验室**：研究由微软（Award #024263）和英特尔（Award #024894）赞助，并由 U.C.Discovery（Award #DIG07-10227）提供匹配资助。额外的支持来自于 Par 实验室附属的诺基亚、英伟达、甲骨文和三星。
- **项目 Isis**：DoE Award DE-SC0003624。
- **ASPIRE 实验室**：DARPA PERFECT 工程，Award HR0011-12-2-0016。DARPA POEM 工程 Award HR0011-11-C-0100。未来架构研究中心（C-FAR），一个由半导体研究公司资助的 STARnet 中心。额外的支持来自于 ASPIRE 工业赞助者，英特尔，和 ASPIRE 附属，谷歌，惠普企业，华为，诺基亚，英伟达，甲骨文，和三星。

本文的内容并不能必然地反映出美国政府的立场和政策，并且不应被推断出官方的认可。

参考文献

[1] Robert P. Goldberg. Survey of virtual machine research. *Computer*, 7(6):34–45, June 1974.

[2] Juan Navarro, Sitaram Iyer, Peter Druschel, and Alan Cox. Practical, transparent operating system support for superpages. *SIGOPS Oper. Syst. Rev.*, 36(SI):89–104, December 2002.