RISC-V 指令集手册
第 2 卷
特权架构

版本：20240411

（注：实际内容为 2025 年 5 月 1 日发布版）

目　　次

序（中文版）

该版本由中国电子工业标准化技术协会 RISC-V 工作委员会技术委员会组织翻译。

该版本的主要贡献人包括：颉靖、唐旖浓、陈振宇、陈俊岳、刘永艳、许文琪。

序（英文版）

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..

前　言

本文档描述了RISC-V的特权架构。本版本（20240213版）包含以下版本的RISC-V ISA模块：

模块	版本	状态
Machine ISA	1.13	草案
Smstateen Extension	1.0.0	批准
Smcrsrind/Sscsrind Extension	1.0.0	批准
Smepmp	1.0.0	批准
*Smcntrpmf	1.0.0	批准
Smrnmi Extension	1.0.0	批准
Smcdeleg	1.13	草案
Supervisor ISA	1.0.0	批准
Svade Extension	0.1	草案
Synapot Extension	1.0	批准
Svpbmt Extension	1.0	批准
Svinval Extension	1.0	批准
Svadu Extension	1.0	批准
Sstc	1.0	批准
Sscofpmf	1.0	批准
Hypervisor ISA	1.0	批准

自1.13版本以来进行了以下变更，尽管这些变更在严格意义上不完全向后兼容，但预计在实际应用中不会导致软件移植性问题：

- 纳入了截至2024年3月所有已批准的扩展标准。

注：原文中涉及20240213版及之前版本，以及1.11、1.10和1.9.1版的前言未在本文件中翻译，若需要了解，请阅读原文。

RISC-V 指令集手册
第二卷：特权架构

1 引言

本文档描述了 RISC-V 的特权架构，涵盖了 RISC-V 系统中除非特权指令集架构（ISA）之外的各个方面，包括特权指令以及运行操作系统和连接外部设备所需的附加功能。

关于设计决策的评论以本段格式（注的形式）呈现。仅关注规范本身的读者可跳过这些非规范性内容。

本文档中描述的整个特权级别设计可以在不改变非特权ISA的情况下被完全不同的特权级别设计所取代，甚至可能在不改变应用二进制接口（ABI）的情况下实现。特别是，此特权规范旨在运行现有的流行操作系统，因此体现了传统的基于级别的保护模型，其他特权规范可能体现更灵活的保护域模型。为了表达简洁，本文档假设这是唯一可能的特权架构。

1.1 RISC-V 特权软件栈术语

本节阐述了在描述 RISC-V 可能的各种特权软件栈组件时使用的术语。

图 1.1 展示了 RISC-V 架构可以支持的一些可能的软件栈。左侧展示了一个简单的系统，该系统仅支持在应用程序执行环境（AEE）上运行的单个应用程序。应用程序编码为使用特定的应用二进制接口（ABI）运行。ABI 包括支持的用户级 ISA 以及一组与 AEE 交互的 ABI 调用。ABI 隐藏了 AEE 的细节，以便在实现 AEE 时具有更大的灵活性。相同的 ABI 可以在多个不同的主机操作系统上本地实现，或者可以在具有不同本地 ISA 的机器上通过用户模式仿真环境支持。

图形约定使用带有白色文本的黑色方框表示抽象接口，以将其与实现接口的具体组件实例分开。

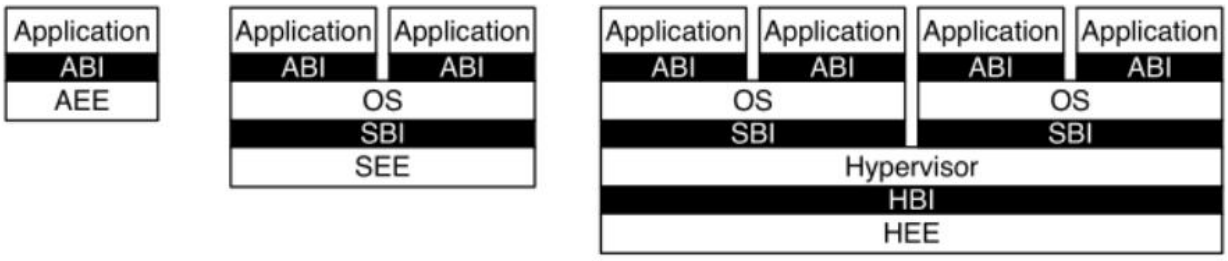


图1.1 支持多种特权执行形式的不同实现技术栈

中间的配置展示了一个传统的操作系统（OS），它能够支持多道程序执行的多个应用程序。每个应用程序通过 ABI 与 OS 通信，OS 提供了应用程序执行环境（AEE）。正如应用程序通过 ABI 与 AEE 接口一样，RISC-V 操作系统通过监管者二进制接口（SBI）与监管者执行环境（SEE）接口。SBI 包括用户级

和监管者级的 ISA 以及一组 SBI 函数调用。在所有 SEE 实现中使用单一的 SBI 允许单一的 OS 二进制映像在任何 SEE 上运行。SEE 可以是一个低端硬件平台中的简单引导加载程序和 BIOS 风格的 IO 系统，也可以是高端服务器中由虚拟机监视器提供的虚拟机，或者是架构仿真环境中主机操作系统上的一个薄翻译层。

大多数监管者级ISA定义并没有将SBI从执行环境和/或硬件平台中分离出来，这使得虚拟化和新硬件平台的启动变得复杂。

最右边的配置展示了一个虚拟机监视器配置，其中单个虚拟机监视器支持多个多道程序操作系统。每个 OS 通过 SBI 与虚拟机监视器通信，虚拟机监视器提供了 SEE。虚拟机监视器通过虚拟机监视器二进制接口（HBI）与虚拟机监视器执行环境（HEE）通信，以将虚拟机监视器与硬件平台的细节隔离。

ABI、SBI和HBI仍在开发中，但目前优先支持Type-2虚拟机监视器，其中SBI由S模式的OS递归提供。

RISC-V ISA 的硬件实现通常需要特权 ISA 之外的附加功能来支持各种执行环境（AEE、SEE 或 HEE）。

1.2 特权级别

在任何时候，RISC-V 硬件线程都以某种特权级别运行，该级别编码为一个或多个控制和状态寄存器（CSR）中的模式。目前定义了三个 RISC-V 特权级别，如表 1.1 所示。

表 1.1 RISC-V 特权级别

级别	代码	名称	简称
0	00	用户	U
1	01	监管	S
2	10	保留	
3	11	机器	M

特权级别用于提供软件栈中不同组件之间的保护，尝试执行当前特权模式不允许的操作将导致引发异常。这些异常通常会引发陷入底层执行环境的陷阱。

在描述中，尽可能将代码编写的特权级别与它运行的特权模式分开，尽管这两者通常是绑定的。例如，一个监管者级别的操作系统可以在具有三种特权模式的系统上以监管者模式运行，但也可以在具有两种或更多特权模式的系统上以用户模式在经典虚拟机监视器下运行。在这两种情况下，可以使用相同的监管者级别操作系统二进制代码，编码为监管者级别的SBI，因此期望能够使用监管者级别的特权指令和CSR。当在用户模式下运行客户操作系统时，所有监管者级别的操作将被运行在更高特权级别的SEE捕获和模拟。

机器级别具有最高的特权，是 RISC-V 硬件平台唯一强制要求的特权级别。在机器模式（M-mode）下运行的代码通常是固有可信的，因为它具有对机器实现的低级访问权限。M-mode 可以用于管理 RISC-V 上的安全执行环境。用户模式（U-mode）和监管者模式（S-mode）分别用于传统的应用程序和操作系统使用。

每个特权级别都有一组核心的特权 ISA 扩展，以及可选的扩展和变体。例如，机器模式支持一个可选的用于内存保护的标准扩展。此外，监管者模式可以扩展以支持如 hypervisor 中所述的 Type-2 虚拟机监视器执行。

实现可能提供从 1 到 3 种特权模式的任意组合，以降低实现成本为代价减少隔离，如表 1.2 所示。

表 1.2 支持的特权模式组合

等级数	支持的模式	使用目的
1	M	简单的嵌入式系统
2	M,U	安全的嵌入式系统
3	M,U,S	运行类 unix 操作系统的系统

所有硬件实现都必须提供机器模式（M-mode），因为这是唯一能够无限制访问整个机器的模式。最简单的 RISC-V 实现可能只提供 M-mode，尽管这将无法防止不正确或恶意的应用程序代码。

即使只实现了M-mode，可选的内存保护设施（PMP）的锁定功能也可以提供一些有限的保护。

许多 RISC-V 实现还将至少支持用户模式（U-mode），以保护系统的其余部分免受应用程序代码的影响。可以添加监管者模式（S-mode）以提供监管者级别操作系统与 SEE 之间的隔离。

硬件线程通常以用户模式（U-mode）运行应用程序代码，直到某些陷阱（例如，监管者调用或定时器中断）强制切换到陷阱处理程序，该处理程序通常在更高特权的模式下运行。然后，硬件线程将执行陷阱处理程序，最终将在 U-mode 中的原始陷阱指令处或之后恢复执行。增加特权级别的陷阱称为垂直陷阱，而保持在同一特权级别的陷阱称为水平陷阱。RISC-V 特权架构提供了灵活的陷阱路由到不同特权层。

水平陷阱可以实现为垂直陷阱，将控制返回到较低特权模式中的水平陷阱处理程序。

1.3 调试模式

实现中可能还包括调试模式以支持片外调试和/或制造测试。调试模式（D-mode）可以被视为一种额外的特权模式，具有比 M-mode 更多的访问权限。单独的调试规范提案描述了 RISC-V 硬件线程在调试模式下的操作。调试模式保留了一些仅在 D-mode 下可访问的 CSR 地址，并且可能还保留了平台上物理地址空间的一些部分。

2 控制与状态寄存器

在 RISC-V 指令集架构中，SYSTEM 主操作码用于编码所有特权指令。这些指令可以分为两大类：一类是原子性地读取-修改-写入控制与状态寄存器（CSR）的指令，这些指令在 Zicsr 扩展中定义；另一类是所有其他特权指令。特权架构要求实现 Zicsr 扩展；而其他特权指令的要求则取决于特权架构的功能集。

除了本手册第一卷中描述的非特权状态外，实现中可能还包含额外的 CSR，这些 CSR 可以通过第一

卷中描述的 CSR 指令在某些特权级别下访问。在本章中，将对 CSR 地址空间进行映射。接下来的章节将根据特权级别描述每个 CSR 的功能，以及其他通常与特定特权级别密切相关的特权指令。需要注意的是，尽管 CSR 和指令与一个特权级别相关联，但它们也可以在更高的特权级别下访问。标准 CSR 在读取时没有副作用，但在写入时可能会产生副作用。

2.1 CSR 地址映射约定

标准的 RISC-V 指令集架构预留了一个 12 位的编码空间（csr[11:0]），用于最多 4,096 个 CSR。按照约定，CSR 地址的高 4 位（csr[11:8]）用于根据特权级别编码 CSR 的读写访问权限，如表 2.1 所示。最高的两位（csr[11:10]）表示寄存器是可读写的（00、01 或 10）还是只读的（11）。接下来的两位（csr[9:8]）编码可以访问该 CSR 的最低特权级别。

CSR地址约定使用CSR地址的高位来编码默认的访问权限。这简化了硬件中的错误检查，并提供了更大的CSR空间，但也限制了CSR在地址空间中的映射。

实现可能允许更高特权级别捕获较低特权级别本应允许的CSR访问，以便拦截这些访问。这种更改对较低特权级别的软件应该是透明的。

执行访问不存在的 CSR 的指令是非法的。在没有适当特权级别的情况下访问 CSR 会引发非法指令异常，或者引发虚拟指令异常。写入只读寄存器会引发非法指令异常。一个可读写的寄存器也可能包含一些只读位，在这种情况下，对只读位的写入将被忽略。

表 2.1 还显示了在标准用途和自定义用途之间分配 CSR 地址的约定。为自定义用途指定的 CSR 地址不会被未来的标准扩展重新定义。

机器模式的标准读写 CSR 0x7A0-0x7BF 保留给调试系统使用。在这些 CSR 中，0x7A0-0x7AF 对机器模式可访问，而 0x7B0-0x7BF 仅对调试模式可见。实现应在机器模式访问后一组寄存器时引发非法指令异常。

有效的虚拟化要求在虚拟化环境中尽可能多地原生运行指令，同时任何特权访问都会陷入虚拟机监视器（VMM）。在某些较低特权级别下为只读的CSR，如果在更高特权级别下被设置为可读写，则会被映射到单独的CSR地址中。这样可以避免捕获允许的低特权访问，同时仍然对非法访问引发陷入。目前，计数器是唯一被映射的CSR。

2.2 CSR 列表

下列表中列出了当前已分配 CSR 地址的 CSR。计时器、计数器和浮点 CSR 是标准的非特权 CSR。其他寄存器由特权代码使用，如下文章节所述。需要注意的是，并非所有寄存器在所有实现中都是必需的。

表 2.1 RISC-V 控制状态寄存器地址空间分配

CSR 地址			十六进制	用途
[11:10]	[9:8]	[7:4]		
非特权 csr 和用户级 csr				
00	00	XXXX	0x000-0x0FF	标准读/写
01	00	XXXX	0x400-0x4FF	标准读/写
10	00	XXXX	0x800-0x8FF	定制读/写
11	00	0XXX	0xC00-0xC7F	标准读
11	00	10XX	0xC80-0xCBF	标准读

11	00	11XX	0xCC0-0xCFF	定制读
主管级别 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	定制读/写
10	01	0XXX	0x900-0x97F	标准读/写
10	01	10XX	0x980-0x9BF	标准读/写
10	01	11XX	0x9C0-0x9FF	定制读/写
11	01	0XXX	0xD00-0xD7F	标准读
11	01	10XX	0xD80-0xDBF	标准读
11	01	11XX	0xDC0-0xDFF	定制读
虚拟化扩展相关控制状态寄存器				
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	定制读/写
10	10	0XXX	0xA00-0xA7F	定制读/写
10	10	10XX	0xA80-0xABF	定制读/写
10	10	11XX	0xAC0-0xAFF	定制读/写
11	10	0XXX	0xE00-0xE7F	标准读
11	10	10XX	0xE80-0xEBF	标准读
11	10	11XX	0xEC0-0xEFF	定制读
机器级控制状态寄存器				
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	Debug 模式
01	11	11XX	0x7C0-0x7FF	定制读/写
10	11	0XXX	0xB00-0xB7F	标准读/写
10	11	10XX	0xB80-0xBBF	标准读/写
10	11	11XX	0xBC0-0xBFF	定制读/写
11	11	0XXX	0xF00-0xF7F	标准读
11	11	10XX	0xF80-0xFBF	标准读
11	11	11XX	0xFC0-0xFFF	定制读

表 2.2 当前分配的 RISC-V 主管级 CSR 地址

序号	特权	名称	描述
监控器陷阱设置			
0x100	SRW	sstatus	主管状态登记

0x104 0x105 0x106	SRW SRW SRW	sie stvec scounteren	监视器中断使能寄存器 监控器陷阱处理程序的基址 启用监控计数器
主管配置			
0x10A	SRW	senvcfg	主管环境配置寄存器
监管者模式计数器配置寄存器			
0x120	SRW	scountinhibit	监管者模式计数器禁用寄存器
监管者陷阱处理			
0x140 0x141 0x142 0x143 0x144 0xDA0	SRW SRW SRW SRW SRW SRO	sscratch sepc scause stval sip scountovf	主管陷阱处理程序的暂存寄存器 主管异常程序计数器。 监控器 trap 原因 主管没有地址或指示 主管中断待定 监控器计数溢出
监管者保护与翻译			
0x180	SRW	satp	主管地址的翻译和保护
调试/跟踪注册			
0x5A8	SRW	scontext	管理器模式上下文寄存器
管理器状态使能寄存器			
0x10C 0x10D 0x10E 0x10F	SRW SRW SRW SRW	sstateen0 sstateen1 sstateen2 sstateen3	Supervisor State 使能 0 寄存器 Supervisor State 使能 1 注册 Supervisor State 使能 2 注册 Supervisor State 启用 3 Register

表 2.3 当前已分配的 RISC-V 机器级控制状态寄存器(CSR)地址

序列	特权	名称	描述
虚拟机监视器			
0x600 0x602 0x603 0x604 0x606 0x607 0x612	HRW HRW HRW HRW HRW HRW HRW	hstatus hedeleg hideleg hie hcounteren hgeie hedelegh	虚拟机状态寄存器 虚拟机异常委托寄存器 虚拟机中断委托寄存器 虚拟机中断启用寄存器 虚拟机计数器启用 虚拟机来宾部中断启用寄存器 hedelegate 的上 32 位，仅支持 RV32
虚拟机监视器陷阱			
0x643 0x644 0x645 0x64A	HRW HRW HRW HRW	htval hip hvip htinst	监管者错误的来宾物理地址 管理程序中断挂起 监管者虚拟中断挂起 管理程序陷阱指令

0xE12	HRO	hgeip	Hypervisor 来宾外部中断待定
虚拟机配置机制			
0x60A 0x61A	HRW HRM	henvcfg henvcfgh	Hypervisor 环境配置寄存器 henvcfg 的上 32 位，仅支持 RV32
虚拟化环境保护和转换			
0x680	HRW	hgatp	监管者来宾地址的转换和保护
调试/跟踪注册			
0x6A8	HRW	hcontext	管理程序模式上下文寄存器
监管者计数器/定时器虚拟化寄存器			
0x605 0x615	HRW HRW	htimedelta htimedeltah	VS/VU 模式定时器增量值
监管者状态启用寄存器			
0x60C 0x60D 0x60E 0x60F 0x61C 0x61D 0x61E 0x61F	HRW HRW HRW HRW HRW HRW HRW HRW	hstateen0 hstateen1 hstateen2 hstateen3 hstateen0h hstateen1h hstateen2h hstateen3h	监管者状态启用 0 寄存器 监管者状态启用 1 注册 监管者状态启用 2 注册 监管者状态启用 3 注册 监管者状态上 32 位使能 0 寄存器，仅 RV32 Hypervisor 状态上 32 位使能 1 寄存器，仅 RV3 Hypervisor 状态上 32 位使能 2 寄存器，只支持 RV32 Hypervisor 状态上 32 位使能 3 寄存器，仅 RV32
虚拟主管寄存器			
0x200 0x204 0x205 0x240 0x241 0x242 0x243 0x244 0x280	HRW HRW HRW HRW HRW HRW HRW HRW HRW	vsstatus vsie vstvec vsscratch vsepc vscause vstval vsip vsatp	虚拟主管状态寄存器 虚拟管理器 interrupt-enable 寄存器 虚拟监控器 trap 处理程序基址 虚拟主管转换寄存器 虚拟机监视器异常计数器 虚拟管理员陷阱原因 虚拟主管地址或指令 虚拟监控中断挂起 虚拟主管地址的翻译和保护

2.3 CSR 字段规范

2.3.1 保留字段：写入保留值，读取忽略值（WPRI）

某些完整的读/写字段保留供未来使用。软件应忽略从这些字段读取的值，并在写入同一寄存器的其他字段时保留这些字段中的值。为了实现向前兼容性，未提供这些字段的实现必须将其设置为只读 0。这些字段在寄存器描述中标记为 WPRI。

为了简化软件模型，任何对CSR中先前保留字段的向后兼容的未来定义必须处理使用非原子读/修改/写序列更新CSR中其他字段的可能性。或者，原始CSR定义必须指定子字段只能以原子方式更新，这可能需要两条指令的清除位/设置位序列，如果中间值不合法，则可能会出现问题。

2.3.2 仅写入/读取合法值（WLRL）

某些读/写 CSR 字段仅指定部分可能位编码的行为，其他位编码保留。软件不应向此类字段写入非法值，并且不应假设读取会返回合法值，除非最后一次写入的是合法值，或者自其他操作（例如复位）将寄存器设置为合法值以来未写入该寄存器。这些字段在寄存器描述中标记为 WLRL。

硬件实现仅需实现足够的状态位以区分支持的值，但在读取时必须始终返回任何支持值的完整指定位编码。

如果指令尝试向 WLRL 字段写入不支持的值，实现可以但不必须引发非法指令异常。当最后一次写入的是非法值时，实现可以在读取 WLRL 字段时返回任意位模式，但返回的值应确定性地依赖于非法写入值和写入前字段的值。

2.3.3 写入任意值，读取合法值（WARL）

某些读/写 CSR 字段仅对部分位编码定义，但允许写入任何值，同时保证读取时始终返回合法值。假设写入 CSR 没有其他副作用，可以通过尝试写入所需设置然后读取以查看是否保留该值来确定支持的值范围。这些字段在寄存器描述中标记为 WARL。

实现不会在向 WARL 字段写入不支持的值时引发异常。当最后一次写入的是非法值时，实现可以在读取 WARL 字段时返回任何合法值，但返回的合法值应确定性地依赖于非法写入值和硬件的架构状态。

2.4 CSR 字段调制

如果对一个 CSR 的写入更改了第二个 CSR 字段允许的合法值集合，则第二个 CSR 的字段立即从其新的合法值中获取一个未指定值，即使写入前字段的值在写入后仍然合法，字段的值也可能因写入控制 CSR 而更改。

作为此规则的特殊情况是，当写入一个CSR的值时，该值能够决定另一个CSR中相应字段的访问属性，即该字段是可写（此时该字段可取多个合法值）还是只读。当对控制CSR的写入导致第二个CSR的字段从先前只读变为可写时，该字段立即获取一个未指定但合法的值，除非另有说明。

某些 CSR 字段在可写时被定义为其他 CSR 字段的别名。设 x 为这样的 CSR 字段，设 y 为它在可写时别名的 CSR 字段。如果对控制 CSR 的写入导致字段 x 从先前只读变为可写，则 x 的新值不是未指定，而是立即反映其别名 y 的现有值，这是必需的。

由于此原因对 CSR 值的更改不是对受影响 CSR 的写入，因此不会触发为该 CSR 指定的任何副作用。

2.5 CSR 的隐式读取

实现有时会执行 CSR 的隐式读取。（例如，所有 S 模式指令获取隐式读取 satp CSR。）除非另有说明，否则 CSR 的隐式读取返回的值与在足够特权模式下使用 CSR 访问指令显式读取 CSR 返回的值相同。

2.6 CSR 宽度调制

如果 CSR 的宽度发生变化（例如，通过更改 SXLEN 或 UXLEN，如 3.1.6.2 中所述），则新宽度 CSR 的可写字段和位的值（除非另有说明）将从先前宽度的 CSR 中确定，具体算法如下：

将先前宽度 CSR 的值复制到一个具有相同宽度的临时寄存器中。

对于先前宽度 CSR 的只读位，临时寄存器中相同位置的位设置为 0。将临时寄存器的宽度更改为新宽度。如果新宽度 W 比先前宽度窄，则保留临时寄存器的最低有效 W 位，并丢弃更高有效位。如果新宽度比先前宽度宽，则将临时寄存器 0 扩展到更宽的宽度。新宽度 CSR 的每个可写字段取临时寄存器中相同位置的位的值。更改 CSR 的宽度不是对 CSR 的读取或写入，因此不会触发任何副作用。

2.7 对宽度大于 XLEN 的 CSR 的显式访问

如果标准 CSR 的宽度大于 XLEN 位，则显式读取 CSR 将返回寄存器的最低有效 XLEN 位，而显式写入 CSR 仅修改寄存器的最低有效 XLEN 位，高位保持不变。

某些标准 CSR（例如扩展 Zicntr 的计数器 CSR）始终为 64 位，即使 XLEN=32（RV32）。对于每个这样的 64 位 CSR（例如计数器 time），通常会定义一个对应的 32 位高半部分 CSR，其名称相同但末尾附加字母“h“（timeh）。高半部分 CSR 别名为其同名 64 位 CSR 的 63:32 位，从而为 RV32 软件提供了一种读取和修改否则无法访问的 32 位的方式。

标准高半部分 CSR 仅在基础 RISC-V 指令集为 RV32（XLEN=32）时可访问。对于 RV64（当 XLEN=64 时），所有标准高半部分 CSR 的地址均保留，因此尝试访问高半部分 CSR 通常会引发非法指令异常。

3 机器级 ISA，V1.13

本章描述了在机器模式（M 模式）下可用的机器级操作，M 模式是 RISC-V 硬件线程中的最高特权模式。M 模式用于对硬件平台进行低级访问，并且是复位时进入的第一个模式。M 模式还可用于实现那些在硬件中直接实现过于困难或昂贵的功能。RISC-V 机器级 ISA 包含一个通用核，该核根据支持的其他特权级别和硬件实现的其他细节进行扩展。

3.1 机器级 CSR

除了本节描述的机器级 CSR 外，M 模式代码还可以访问较低特权级别的所有 CSR。

3.1.1 机器 ISA（misa）寄存器

misa CSR 是一个写任意读合法（WARL）读写寄存器，用于报告硬件线程支持的 ISA。在任何实现中，该寄存器都必须可读。如果 misa 寄存器尚未实现，它可能会返回零值。在这种情况下，需要通过单独的非标准机制来确定 CPU 功能。

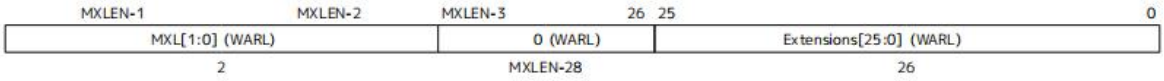


图 3.1 机器 ISA 寄存器

MXL（Machine XLEN）字段编码了原生的基础整数指令集架构（ISA）宽度，如下表所示。MXL 字段是只读的。如果 misa 不为 0，MXL 字段表示在 M 模式下的有效 XLEN，这个常量称为 MXLEN。XLEN 永远不会大于 MXLEN，但在较低权限模式下，XLEN 可能小于 MXLEN。

表 3.1 主管间接寄存器选择

MXL	XLEN
1	32
2	64
3	128

可以通过对返回的misa值的符号进行分支判断，以及可能的一次左移和第二次符号分支判断，快速确定基础宽度。这些检查可以在不知道硬件线程寄存器宽度（MXLEN）的情况下用汇编代码编写。基础宽度由公式MXLEN=2^{MXL+4}给出。

如果misa为0，也可以通过将立即数4放入寄存器，然后每次将寄存器左移31位来确定基础宽度。如果一次移位后为0，则硬件线程为RV32。如果两次移位后为0，则硬件线程为RV64，否则为RV128。

扩展字段编码了标准扩展的存在情况，每个字母对应一个位（位 0 编码扩展“A“，位 1 编码扩展“B“，依此类推，到位 25 编码“Z”）。对于 RV32I、RV64I 和 RV128I 基础 ISA，“I“位将被设置；对于 RV32E 和 RV64E，“E“位将被设置。扩展字段是一个 WARL 字段，可以包含可写位，如果实现允许修改支持的 ISA。在复位时，扩展字段应包含支持的最大扩展集，并且如果“I“和“E“都可用，则选择“I”。

当通过清除 misa 中的位来禁用标准扩展时，该扩展定义或修改的指令和 CSR 将恢复为其定义或保留的行为，就像该扩展未实现一样。

对于给定的RISC-V执行环境，RISC-V ISA的指令、扩展或其他功能通常根据该环境中的可观察执行行为来判断是否实现。例如，如果RISC-V非特权ISA为F定义的指令按指定执行，则F扩展被认为在该执行环境中实现。

根据实现的这一定义，通过清除misa中的位来禁用扩展会导致该扩展在M模式下被视为未实现。例如，设置misa.F=0会导致F扩展在M模式下未实现，因为F扩展的指令不会按照非特权ISA的要求执行，而是可能引发非法指令异常。

严格基于可观察行为定义术语实现可能会与其他常见理解相冲突。具体而言，尽管通常用法可能允许“实现但禁用“的组合，但在本文档中，这被认为是术语的矛盾，因为禁用意味着执行不会按照要求的行为进行，因此该功能不被视为实现。同样，“实现并启用“在这里是多余的；“实现“就足够了。

misa 中扩展字段的编码。所有保留供未来使用的位在读取时必须返回 0。

表 3.2 特征对应表

比特	特征	描述
0	A	原子扩展
1	B	B 扩展
2	C	压缩扩展
3	D	双精度浮点扩展
4	E	RV32E/64E 基 ISA
5	F	单精度浮点扩展
6	G	保留
7	H	虚拟机监视器扩展
8	I	RV32I/64I/128I 基 ISA

9	J	保留
10	K	保留
11	L	保留
12	M	整数乘法/除法扩展
13	N	暂时保留给用户级中断扩展
14	O	保留
15	P	暂定用于打包 simd 扩展
16	Q	四精度浮点扩展
17	R	保留
18	S	实现主管模式
19	T	保留
20	U	实现用户模式
21	V	向量扩展
22	W	保留
23	X	非标准扩展
24	Y	保留
25	Z	保留

RV128I 基础 ISA 的设计尚未完成，尽管本规范的其余部分预计将适用于 RV128，但本文档的当前版本仅关注 RV32 和 RV64。

如果分别支持 U 模式和监管模式，则“U“和“S“位将被设置。

如果有任何非标准扩展，则“X“位将被设置。

当“B“位为 1 时，实现支持 Zba、Zbb 和 Zbs 扩展提供的指令。当“B“位为 0 时，表示实现可能不支持 Zba、Zbb 或 Zbs 扩展中的一个或多个。

misa CSR向机器模式代码暴露了一个基本的CPU功能目录。更广泛的信息可以通过探测其他机器寄存器以及在系统启动过程中检查其他ROM存储来在机器模式下获取。

要求较低权限级别执行环境调用，而不是读取CPU寄存器来确定每个权限级别可用的功能。这使得虚拟化层能够更改在任何级别观察到的ISA，并支持更丰富的命令接口，而不会增加硬件设计的负担。

“E“位是只读的。除非 misa 全部为只读 0，否则“E“位始终读取为“I“位的补码。如果一个执行环境同时支持 RV32E 和 RV32I，软件可以通过清除“I“位来选择 RV32E。如果一个 ISA 特性 x 依赖于 ISA 特性 y，则尝试启用特性 x 但禁用特性 y 会导致两个特性都被禁用。例如，设置“F“=0 和“D“=1 会导致“F“和“D“都被清除。

实现可能会对两个或多个 misa 字段的集体设置施加额外的约束，在这种情况下，它们作为一个单一的 WARL 字段共同起作用。尝试写入不受支持的组合会导致这些位被设置为某个受支持的组合。

写入 misa 可能会增加 IALIGN，例如通过禁用“C“扩展。如果一条指令将写入 misa 并增加 IALIGN，并且后续指令的地址未按 IALIGN 位对齐，则对 misa 的写入将被抑制，misa 保持不变。当软件启用之前禁用的扩展时，除非该扩展另有规定，否则与该扩展唯一相关的所有状态都是未指定的。

尽管misa中25--0位中的某一位设置为1意味着相应的功能已实现，但反之则不一定成立：这些位中的某一位被清除并不一定意味着相应的功能未实现。这是因为当某个功能未实现时，相应的操作码和CSR变为保留，不一定是非法的。

3.1.2 机器供应商 ID（mvendorid）寄存器

mvendorid CSR 是一个 32 位的只读寄存器，用于提供核供应商的 JEDEC 制造商 ID。在任何实现中，该寄存器必须可读，但可以返回 0 值以表示该字段未实现或这是一个非商业实现。

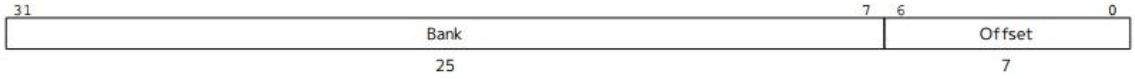


图 3.2 供应商 ID 寄存器

JEDEC 制造商 ID 通常编码为一个字节的延续码序列 0x7f，以一个不等于 0x7f 的单字节 ID 终止，每个字节的最高有效位包含一个奇校验位。mvendorid 在 Bank 字段中编码单字节延续码的数量，并在 Offset 字段中编码最后一个字节，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（十二个延续码后跟 0x8a）将在 mvendorid CSR 中编码为 0x60a。

在JEDEC的术语中，银行号比延续码的数量多一；因此，mvendorid的Bank字段编码的值比JEDEC银行号少一。

过去，供应商ID是由RISC-V国际分配的号码，但这与JEDEC维护制造商ID标准的工作重复。截至撰写时，向JEDEC注册制造商ID的一次性费用为500美元。

3.1.3 机器架构 ID（marchid）寄存器

marchid CSR 是一个 MXLEN 位的只读寄存器，用于编码硬件线程的基本微架构。在任何实现中，该寄存器必须可读，但可以返回 0 值以表示该字段未实现。mvendorid 和 marchid 的组合应唯一标识所实现的硬件线程微架构类型。

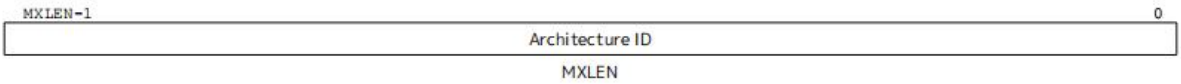


图 3.3 机器体系结构 ID 寄存器

开源项目架构 ID 由 RISC-V 国际全局分配，具有非 0 架构 ID 且最高有效位（MSB）为 0。商业架构 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 0。

意图是让架构ID代表围绕开发发生的仓库相关的微架构，而不是特定组织。开源设计的商业制造应（并且可能根据许可证要求）保留原始架构ID。这将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国际管理，应仅分配给已发布、功能正常的开源项目。商业架构ID可以由任何注册供应商独立管理，但必须具有与开源架构ID不相交的ID（设置MSB），以防止供应商希望同时使用闭源和开源微架构时发生冲突。

以下实现字段中采用的约定可用于隔离同一架构设计的分支，包括按组织隔离。misa寄存器也有助于区分设计的不同变体。

3.1.4 机器实现 ID（mimpid）寄存器

mimpid CSR 提供了处理器实现版本的唯一编码。在任何实现中，该寄存器必须可读，但可以返回

0 值以表示该字段未实现。实现值应反映 RISC-V 处理器本身的设计，而不是任何周围系统。

图 3.4 机器实现 ID 寄存器

该字段的格式由架构源代码的提供者决定，但通常会由标准工具以十六进制字符串的形式打印，且不包含任何前导或尾随0。因此，实现值可以左对齐（即从最高有效半字节开始填充），并且子字段在半字节边界上对齐，以便于人类阅读。

3.1.5 硬件线程 ID（m 硬件线程 id）寄存器

m 硬件线程 id CSR 是一个 MXLEN 位的只读寄存器，包含运行代码的硬件线程的整数 ID。在任何实现中，该寄存器必须可读。在多处理器系统中，硬件线程 ID 不一定连续编号，但至少必须有一个硬件线程的 ID 为 0。硬件线程 ID 在执行环境中必须是唯一的。

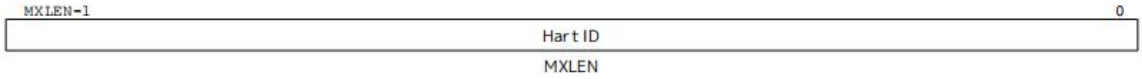


图 3.5 硬件线程 ID 寄存器

在某些情况下，必须确保只有一个硬件线程运行某些代码（例如，在复位时），因此需要一个硬件线程具有已知的硬件线程 ID为0。

为了提高效率，系统实现者应尽量减少系统中使用的最大硬件线程 ID的幅度。

3.1.6 机器状态（mstatus 和 mstatush）寄存器

mstatus 寄存器是一个 MXLEN 位的读/写寄存器，其 RV32 和 RV64 的格式如的下图所示。mstatus 寄存器跟踪并控制硬件线程的当前操作状态。mstatus 的受限视图在 S 级 ISA 中显示为 sstatus 寄存器。

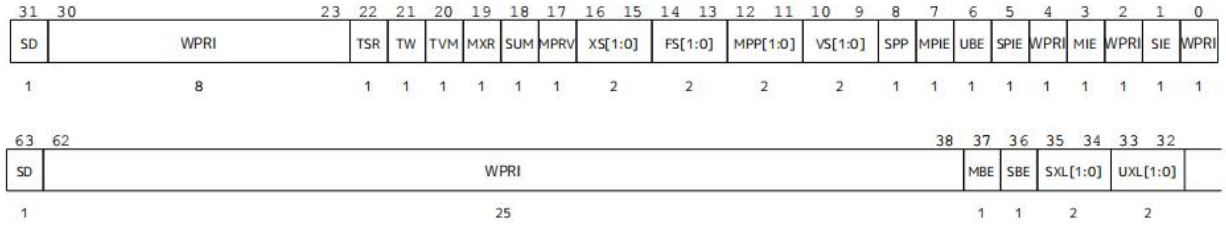


图 3.6 RV32 和 RV64 机器模式状态寄存器

仅适用于 RV32，mstatush 是一个 32 位的读/写寄存器，其格式如下图所示。mstatush 的位 30:4 通常包含与 RV64 中 mstatus 的位 62:36 相同的字段。mstatush 中不存在 SD、SXL 和 UXL 字段。

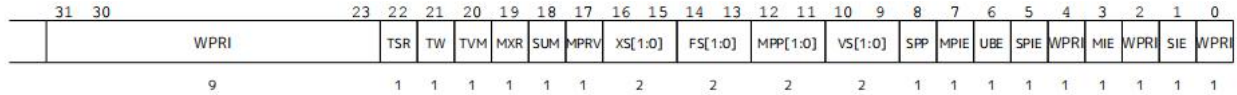


图 3.7 RV32 的附加机器模式状态寄存器（mstatush）

3.1.6.1 寄存器中的特权模式和全局中断使能栈

全局中断使能位 MIE 和 SIE 分别用于 M 模式和 S 模式。这些位主要用于保证当前特权模式下中断处理程序的原子性。

全局 xIE 位位于 mstatus 的低位，允许通过一条 CSR 指令原子地设置或清除它们。

当 硬件线程 在特权模式 x 下执行时，如果 xIE=1，则中断全局使能；如果 xIE=0，则中断全局禁用。对于较低特权模式 w<x 的中断，无论较低特权模式的全局 wIE 位如何设置，始终全局禁用。对于较高特权模式 y>x 的中断，无论较高特权模式的全局 yIE 位如何设置，始终全局使能。较高特权级别的代码可以在将控制权交给较低特权模式之前，使用单独的中断使能位来禁用选定的较高特权模式中断。

较高特权模式 y 可以在将控制权交给较低特权模式之前禁用其所有中断，但这并不常见，因为这将导致只能通过同步陷阱、不可屏蔽中断或复位来重新获得 硬件线程 的控制权。

为了支持嵌套陷阱，每个能够响应中断的特权模式 x 都有一个两级的中断使能位和特权模式栈。xPIE 保存陷阱发生前活动的中断使能位的值，xPP 保存先前的特权模式。xPP 字段只能保存最高为 x 的特权模式，因此 MPP 为两位宽，SPP 为一位宽。当从特权模式 y 进入特权模式 x 的陷阱时，xPIE 设置为 xIE 的值；xIE 设置为 0；xPP 设置为 y。

对于较低特权模式，任何陷阱（同步或异步）通常会在进入时以禁用中断的较高特权模式处理。较高级别的陷阱处理程序将要么服务陷阱并使用栈中的信息返回，要么如果不立即返回到中断上下文，则会在重新启用中断之前保存特权栈，因此每个栈只需要一个条目。

MRET 或 SRET 指令分别用于从 M 模式或 S 模式的陷阱返回。当执行 xRET 指令时，假设 xPP 保存值 y，则 xIE 设置为 xPIE；特权模式更改为 y；xPIE 设置为 1；xPP 设置为支持的最低特权模式（如果实现了 U 模式则为 U，否则为 M）。如果 y≠M，xRET 还会将 MPRV 设置为 0。

在 xRET 上将 xPP 设置为支持的最低特权模式有助于识别两级特权模式栈管理中的软件错误。

陷阱处理程序必须设计为在处理阶段既不启用中断也不引发异常，该阶段保存处理陷阱和从中恢复所需的关键状态信息。在陷阱处理的关键阶段发生异常或中断可能会导致覆盖此类关键状态的陷阱。这可能导致丢失从初始陷阱恢复所需的数据。此外，如果在处理陷阱所需的代码路径中发生异常，则这种情况可能导致无限循环的陷阱。为了防止这种情况，陷阱处理程序必须精心设计，以识别并安全地管理其操作流程中的异常。

xPP 字段是 WARL 字段，只能保存特权模式 x 和任何低于 x 的实现特权模式。如果特权模式 x 未实现，则 xPP 必须为只读 0。

M 模式软件可以通过将该模式写入 MPP 然后读回来确定是否实现了某个特权模式。

如果机器仅提供 U 和 M 模式，则只需要一个硬件存储位来表示 MPP 中的 00 或 11。

3.1.6.2 mstatus 寄存器中的基础 ISA 控制

对于 RV64 的 硬件线程，SXL 和 UXL 字段是 WARL 字段，分别控制 S 模式和 U 模式的 XLEN 值。这些字段的编码与 misa 的 MXL 字段相同，如表 3.1 所示。S 模式和 U 模式中的有效 XLEN 分别称为 SXLEN 和 UXLEN。

当 MXLEN=32 时，SXL 和 UXL 字段不存在，且 SXLEN=32 和 UXLEN=32。

当 MXLEN=64 时，如果不支持 S 模式，则 SXL 为只读 0。否则，它是一个 WARL 字段，编码当前 SXLEN 的值。具体而言，实现可以使 SXL 成为只读字段，其值始终确保 SXLEN=MXLEN。

当 MXLEN=64 时，如果不支持 U 模式，则 UXL 为只读 0。否则，它是一个 WARL 字段，编码当前 UXLEN 的值。具体而言，实现可以使 UXL 成为只读字段，其值始终确保 UXLEN=MXLEN 或 UXLEN=SXLEN。

如果实现了 S 模式，UXL 字段可能假设的合法值集合不包括那些会导致 UXLEN 大于 SXLEN 的值。

每当任何模式中的 XLEN 设置为小于支持的最大 XLEN 的值时，所有操作必须忽略源操作数寄存器中高于配置 XLEN 的位，并且必须将结果符号扩展以填充目标寄存器中支持的最大 XLEN。同样，pc 中高于 XLEN 的位被忽略，当写入 pc 时，它被符号扩展以填充支持的最大 XLEN。

要求操作始终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。

为了减少硬件复杂性，架构不检查较低特权模式的 XLEN 设置是否小于或等于下一个较高特权模式。实际上，这样的设置几乎总是软件错误，但即使在这种情况下，机器的操作也是定义良好的。

一些 HINT 指令被编码为整数计算指令，这些指令用当前值覆盖其目标寄存器，例如 c.addi x8, 0。当使用 XLEN < MXLEN 执行此类 HINT 时，如果目标寄存器的 MXLEN..XLEN 位不全等于 XLEN-1 位，则目标寄存器的 MXLEN..XLEN 位是保持不变还是被 XLEN-1 位的副本覆盖是实现定义的。

此定义允许实现省略某些 HINT 的寄存器写回，同时允许它们以与其他整数计算指令相同的方式执行其他 HINT。

实现选择仅对 XLEN 设置大于当前 XLEN 的特权模式可观察；对当前特权模式不可见。

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，即加载和存储执行的特权级别。当 MPRV=0 时，加载和存储行为正常，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的转换和保护以及字节序应用，就像当前特权模式设置为 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，MPRV 为只读 0。

将特权模式更改为比 M 模式更低特权的 MRET 或 SRET 指令也会将 MPRV 设置为 0。

MXR（使可执行可读）位修改加载访问虚拟内存的特权。当 MXR=0 时，只有从标记为可读（R=1）的页面加载才会成功。当 MXR=1 时，从标记为可读或可执行（R=1 或 X=1）的页面加载都会成功。当基于页面的虚拟内存未生效时，MXR 无效。如果不支持 S 模式，MXR 为只读 0。

MPRV 和 MXR 机制旨在提高模拟缺失硬件功能的 M 模式例程的效率，例如未对齐的加载和存储。MPRV 避免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被视为当前 XLEN 设置为 MPP 的 XLEN，遵循3.1.6.2中的规则。

SUM（允许监管模式访问用户内存）位修改了 S 模式下加载和存储访问虚拟内存的特权。当 SUM=0 时，S 模式对 U 模式可访问的页面（ U=1）的内存访问将引发故障。当 SUM=1 时，这些访问被允许。当基于页面的虚拟内存未生效时,SUM 无效。请注意,虽然 SUM 在非 S 模式下通常被忽略,但当 MPRV=1 且 MPP=S 时，SUM 仍然有效。如果不支持 S 模式或 satp.MODE 为只读 0，则 SUM 为只读 0。

MXR 和 SUM 机制仅影响页表条目中编码的权限解释。具体而言,它们不会影响由于 PMA 或 PMP 引发的访问故障异常。

3.1.6.4 mstatus 和 mstatush 寄存器中的字节序控制

mstatus 和 mstatush 中的 MBE、SBE 和 UBE 位是 WARL 字段，用于控制除指令取指外的内存访问的字节序。指令取指始终为小端序。

MBE 控制从 M 模式（假设 mstatus.MPRV=0）进行的非指令取指内存访问是小端序（MBE=0）还是大端序（MBE=1）。

如果不支持 S 模式，则 SBE 为只读 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序（SBE=0）还是大端序（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序（UBE=0）还是大端序（UBE=1）。

对于对监管级内存管理数据结构（如页表）的 隐式 访问，字节序始终由 SBE 控制。由于更改 SBE 会改变实现对这些数据结构的解释，如果在更改 SBE 后仍在使用任何此类数据结构，则 M 模式软件必须在更改 SBE 后执行一条 SFENCE.VMA 指令，其中 rs1=x0 且 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同时解释为小端序和大端序。实际上，SBE 只会在世界切换时在运行时更改，在这种情况下，旧的和新的内存管理数据结构都不会以不同的字节序重新解释。在这种情况下，除了通常在世界切换时所需的 SFENCE.VMA 外，不需要额外的 SFENCE.VMA。

如果支持 S 模式，实现可以使 SBE 成为 MBE 的只读副本。如果支持 U 模式，实现可以使 UBE 成为 MBE 或 SBE 的只读副本。

如果字段 MBE、SBE 和 UBE 均为只读 0，则实现仅支持小端序内存访问。如果 MBE 为只读 1，并且在支持 S 模式和U模式时 SBE 和 UBE 均为只读 1，则实现仅支持大端序内存访问（除指令取指外）。

第一卷将 硬件线程 的地址空间定义为连续的 2^{XLEN} 字节的循环序列。地址与字节位置之间的对应关系是固定的，不受任何字节序模式的影响。相反，适用的字节序模式决定了内存字节与多字节量（半字、字等）之间的映射顺序。

标准的 RISC-V ABI 预计为纯小端序或纯大端序，不支持混合字节序。然而，字节序控制已被定义，以允许例如一种字节序的操作系统执行相反字节序的U模式程序。还考虑了非标准用途的可能性，即软件根据需要翻转内存访问的字节序。

RISC-V 指令统一为小端序，以将指令编码与当前字节序设置解耦，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字节序，尽管字节序模式可能在执行期间动态更改。相比之下，通过为指令提供固定的字节序，有时可以精心编写的软件在二进制形式中与字节序无关，就像位置无关代码一样。

然而，选择仅使指令为小端序确实对编码或解码机器指令的 RISC-V 软件有影响。在大端序模式下，此类软件必须考虑显式加载和存储的字节序与指令的字节序相反，例如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（陷阱虚拟内存）位是一个 WARL 字段，用于支持拦截监管虚拟内存管理操作。当 TVM=1 时，在 S 模式下执行时尝试读取或写入 satp CSR 或执行 SFENCE.VMA 或 SINVAL.VMA 指令将引发非法指令异常。当 TVM=0 时，这些操作在 S 模式下被允许。如果不支持 S 模式，则 TVM 为只读 0。

TVM 机制通过允许客户操作系统在 S 模式下执行，而不是在U模式下进行经典虚拟化，提高了虚拟化效率。这种方法避免了对大多数 S 模式 CSR 访问的陷阱。

捕获 satp 访问以及 SFENCE.VMA 和 SINVAL.VMA 指令提供了延迟填充影子页表所需的钩

TW（超时等待）位是一个 WARL 字段，用于支持拦截 WFI 指令（见 wfi）。当 TW=0 时，WFI 指令可以在较低特权模式下执行，除非因其他原因被阻止。当 TW=1 时，如果在任何较低特权模式下执行 WFI，并且它在实现特定的有限时间内未完成，则 WFI 指令将引发非法指令异常。实现可以使 WFI 在 TW=1 时始终在较低特权模式下引发非法指令异常，即使在执行指令时有挂起的全局禁用中断。如果没有比 M 模式更低的特权模式，则 TW 为只读 0。

捕获 WFI 指令可以触发切换到另一个客户操作系统的世界切换，而不是在当前客户中浪费性地空闲。

当实现 S 模式时，在 U 模式下执行 WFI 会导致非法指令异常，除非它在实现特定的有限时间内完成。本规范的未来修订可能会添加一个功能，允许 S 模式选择性地允许在 U 模式下执行 WFI。此功能仅在 TW=0 时激活。

TSR（陷阱 SRET）位是一个 WARL 字段，用于支持拦截监管异常返回指令 SRET。当 TSR=1 时，在 S 模式下执行时尝试执行 SRET 将引发非法指令异常。当 TSR=0 时，此操作在 S 模式下被允许。如果不支持 S 模式，则 TSR 为只读 0。

捕获 SRET 对于在不提供虚拟机监视器扩展的实现上模拟虚拟机监视器扩展（见 hypervisor）是必要的。

3.1.6.6 mstatus 寄存器中的扩展上下文状态

支持大量扩展是 RISC-V 的主要目标之一，因此定义了一个标准接口，以允许未更改的特权模式代码（特别是监管级操作系统）支持任意的 U 模式状态扩展。

迄今为止，V 扩展是唯一一个定义了超出浮点 CSR 和数据寄存器的附加状态的标准扩展。

FS[1:0] 和 VS[1:0] WARL 字段以及 XS[1:0] 只读字段用于通过设置和跟踪浮点单元和任何其他 U 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 fflags。VS 字段编码向量扩展状态的状态，包括向量寄存器 v0－v31 和 CSR vcsr、vxrm、vxsat、vstart、vl、vtype 和 vlenb。XS 字段编码附加 U 模式扩展和相关状态的状态。这些字段可以由上下文切换例程检查，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，通常需要额外的指令和 CSR 来执行和优化该过程。

设计预期大多数上下文切换不需要保存/恢复浮点单元或其他扩展中的状态，因此通过 SD 位提供了快速检查。

FS、VS 和 XS 字段使用与 fsxsencoding 中所示相同的状态编码，四种可能的状态值为关闭（Off）、初始（Initial）、干净（Clean）和脏（Dirty）。

表 3.3 FS[1:0]、VS[1:0]、XS[1:0]状态字段的编码

状态	FS 和 VS 含义	XS 含义
0	关闭	全部关闭
1	初始	部分开启，无脏数据，无干净数据
2	干净	无脏数据，部分干净数据
3	脏	部分脏数据

如果实现了 F 扩展，则 FS 字段不得为只读 0。

如果既未实现 F 扩展也未实现 S 模式，则 FS 为只读 0。如果实现了 S 模式但未实现 F 扩展，则 FS 可以选择为只读 0。

允许但不要求具有 S 模式但未实现 F 扩展的实现将 FS 字段设置为只读0。一些此类实现可能会选择 不 将 FS 字段设置为只读0，以便通过不可见的 M 模式陷阱来模拟 S 模式和U模式的 F 扩展。

如果实现了 v 寄存器，则 VS 字段不得为只读 0。如果既未实现 v 寄存器也未实现 S 模式，则 VS 为只读 0。如果实现了 S 模式但未实现 v 寄存器，则 VS 可以选择为只读 0。

在没有需要新状态的附加用户扩展的 硬件线程 中，XS 字段为只读 0。每个具有状态的附加扩展都提供一个 CSR 字段，用于编码与 XS 状态等效的状态。XS 字段表示所有扩展状态的摘要，如 表 3.3 所示。

XS 字段有效地报告所有用户扩展状态字段中的最大状态值，尽管各个扩展可以使用与 XS 不同的编码。

SD 位是一个只读位，用于总结 FS、VS 或 XS 字段是否指示存在某些脏状态，这些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 均为只读 0，则 SD 也始终为 0。

当扩展的状态设置为关闭（Off）时，任何尝试读取或写入相应状态的指令都将引发非法指令异常。当状态为初始（Initial）时，相应的状态应具有初始常量值。当状态为干净（Clean）时，相应的状态可能与初始值不同，但与上次上下文交换时存储的值匹配。当状态为脏（Dirty）时，相应的状态可能自上次上下文保存以来已被修改。

在上下文保存期间，负责的特权代码只需在状态为脏（Dirty）时写出相应的状态，然后可以将扩展的状态重置为干净（Clean）。在上下文恢复期间，只需在状态为干净（Clean）时从内存加载上下文（在恢复时状态不应为脏（Dirty））。如果状态为初始（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，浮点寄存器可以全部初始化为立即值 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 字段通过写入各个扩展的状态寄存器间接设置。状态字段也会在执行指令期间更新，无论特权模式如何。

U 模式 ISA 的扩展通常包括附加的 U 模式状态，并且这些状态可能比基础整数寄存器大得多。这些扩展可能仅用于某些应用程序，或者可能仅在单个应用程序的短阶段内需要。为了提高性能，U 模式扩展可以定义附加指令，以允许 U 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首次使用前进行配置，并在使用后“取消配置”。取消配置状态将表示为上下文保存的初始（Initial）状态。如果同一应用程序在取消配置和下一次配置之间保持运行（这将状态设置为脏（Dirty）），则无需在取消配置指令处实际重新初始化状态，因为所有状态都是用户进程本地的，即初始（Initial）状态可能仅在上下文恢复时将协处理器状态初始化为常量值，而不是在每次取消配置时。

执行 U 模式指令以禁用单元并将其置于关闭（Off）状态时，如果任何后续指令在单元重新打开之前尝试使用该单元，则将引发非法指令异常。U 模式指令打开单元还必须确保单元的状态已正确初始化，因为单元可能在此期间已被另一个上下文使用。

更改 FS 的设置不会影响浮点寄存器状态的内容。具体而言，将 FS 设置为关闭（Off）不会破坏状态，将 FS 设置为初始（Initial）也不会清除内容。类似地，VS 的设置不会影响向量寄存器状态的内容。然而，其他扩展在设置为关闭（Off）时可能不会保留状态。

实现可以选择不精确地跟踪浮点寄存器状态的脏性，即使状态未被修改也报告为脏（Dirty）。在某些实现中，一些不改变浮点状态的指令可能会导致状态从初始（Initial）或干净（Clean）转换为脏（Dirty）。在其他实现中，脏性可能根本不跟踪，在这种情况下，有效的 FS 状态是关闭（Off）和脏（Dirty），尝试将 FS 设置为初始（Initial）或干净（Clean）会导致其设置为脏（Dirty）。

FS 的此定义不禁止由于错误推测而将 FS 设置为脏（Dirty）。某些平台可能会选择禁止推测性地写入 FS 以关闭潜在的侧信道。

如果指令显式或隐式写入浮点寄存器或 fcsr 但不改变其内容，并且 FS=初始（Initial）或 FS=

干净（Clean），则 FS 是否转换为脏（Dirty）是实现定义的。

实现可以选择以类似的不精确方式跟踪向量寄存器状态的脏性，包括在软件尝试将 VS 设置为初始（Initial）或干净（Clean）时可能将 VS 设置为脏（Dirty）。当 VS=初始（Initial）或 VS=干净（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 转换为脏（Dirty）是实现定义的。

表 3.4 FS、VS 或 XS 状态位的可能状态转换

当前状态 行动	Off	Initial	Clean	Dirty
保存状态? 下一个状态	No Off	No Initial	No Clean	Yes Clean
恢复状态? 下一个状态	No Off	Yes, to initial Initial	Yes, from memory Clean	N/A N/A
行动? 下一个状态	Exception Off	Execute Initial	Execute Clean	Execute Dirty
行动? 下一个状态	Exception Off	Execute Dirty	Execute Dirty	Execute Dirty
行动? 下一个状态	Exception Off	Execute Initial	Execute Initial	Execute Initial
行动? 下一个状态	Execute Off	Execute Off	Execute Off	Execute Off
行动? 下一个状态	Execute Initial	Execute Initial	Execute Initial	Execute Initial

标准的特权指令用于初始化、保存和恢复扩展状态，通过将状态视为不透明对象来隔离特权代码与附加扩展状态的细节。

许多协处理器扩展仅在有限的上下文中使用，这使得软件在完成后可以安全地取消配置甚至禁用单元。这减少了具有大状态协处理器的上下文切换开销。

将浮点状态与其他扩展状态分开，因为当存在浮点单元时，浮点寄存器是标准调用约定的一部分，因此U模式软件无法知道何时可以安全地禁用浮点单元。

XS 字段提供了所有附加扩展状态的摘要，但扩展中可能会维护额外的微架构位，以进一步减少上下文保存和恢复的开销。

SD 位是只读的，当 FS、VS 或 XS 位编码为脏（Dirty）状态时设置（即 SD=((FS==11) OR (XS==11) OR (VS==11))）。这允许特权代码快速确定何时不需要在整数寄存器集和 pc 之外进行额外的上下文保存。浮点单元状态始终使用标准指令（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。

机器模式和监管模式共享 FS、VS 和 XS 位的单一副本。监管级软件通常直接使用 FS、VS 和 XS 位来记录与监管级保存上下文相关的状态。机器级软件在保存和恢复扩展状态时必须更加保守，以对应其上下文的版本。

在任何合理的使用情况下，用户和监管级别之间的上下文切换次数应远远超过到其他特权级别的上下文切换次数。请注意，协处理器不应要求保存和恢复其上下文以服务异步中断，除非中断导致用户级上下文交换。

3.1.7 机器陷阱向量基址寄存器（mtvec）

mtvec 寄存器是一个 MXLEN 位的 WARL 读/写寄存器，用于保存陷阱向量配置，包括向量基地址（BASE）和向量模式（MODE）。

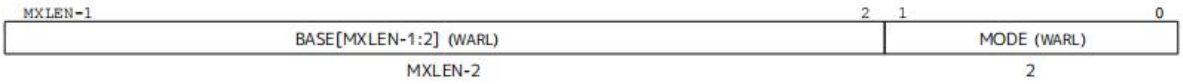


图 3.8 mtvec 字段的编码

mtvec 寄存器必须始终实现，但可以包含一个只读值。如果 mtvec 是可写的，则寄存器可能保存的值集可能因实现而异。BASE 字段中的值必须始终对齐到 4 字节边界，并且 MODE 设置可能会对 BASE 字段中的值施加额外的对齐约束。

在陷阱向量基地址的实现中允许相当大的灵活性。一方面，不希望给低端实现增加大量状态位的负担，但另一方面，希望为更大的系统提供灵活性。

表 3.5 mtvec MODE 字段的编码

值	名称	描述
0	直接	所有陷阱设置 pc 为 BASE
1	矢量	异步中断设置 pc 为 BASE+4×cause
≥2	---	保留

MODE 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都会将 pc 设置为 BASE 字段中的地址。当 MODE=Vectored 时，所有进入机器模式的同步异常都会将 pc 设置为 BASE 字段中的地址，而中断则会将 pc 设置为 BASE 字段中的地址加上中断原因编号的四倍。例如，机器模式的定时器中断会将 pc 设置为 BASE+0x1c。

实现可能对不同的模式有不同的对齐约束。具体而言，MODE=Vectored 可能比 MODE=Direct 有更严格的对齐约束。

在 Vectored 模式中允许更粗的对齐方式，可以在没有硬件加法器电路的情况下实现向量化。

复位和 NMI 向量的位置在平台规范中给出。

3.1.8 机器陷阱委托（medeleg 和 mideleg）寄存器

默认情况下，任何特权级别的所有陷阱都在机器模式中处理，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的级别。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由较低特权级别直接处理。机器异常委托寄存器（medeleg）是一个 64 位的读/写寄存器。机器中断委托（mideleg）寄存器是一个 MXLEN 位的读/写寄存器。

在具有 S 模式的 硬件线程 中，medeleg 和 mideleg 寄存器必须存在，并且在 medeleg 或 mideleg 中设置一个位将在 S 模式或 U 模式中发生相应陷阱时将其委托给 S 模式陷阱处理程序。在没有 S 模式的 硬件线程 中，medeleg 和 mideleg 寄存器不应存在。

在 1.9.1 及更早版本中，这些寄存器存在但在仅 M 模式或 M/U 没有 N 的 硬件线程 中被硬连线为 0。在这些情况下没有理由要求它们返回0，因为 misa 寄存器指示它们是否存在。

当陷阱被委托给 S 模式时，scause 寄存器被写入陷阱原因；sepc 寄存器被写入发生陷阱的指令的虚拟地址；stval 寄存器被写入异常特定的数据；mstatus 的 SPP 字段被写入陷阱发生时的活动特权模式；mstatus 的 SPIE 字段被写入陷阱发生时 SIE 字段的值；并且 mstatus 的 SIE 字段被清除。mcause、mepc 和 mtval 寄存器以及 mstatus 的 MPP 和 MPIE 字段不会被写入。

实现可以选择子集可委托的陷阱，通过向每个位位置写入一来找到支持的可委托位，然后读取 medeleg 或 mideleg 中的值以查看哪些位位置保持为一。

实现不得将 medeleg 的任何位设置为只读一，即任何可以委托的同步陷阱必须支持不被委托。同样，实现不得将 mideleg 的任何对应于机器级中断的位固定为只读一（但可以对较低级别的中断这样做）。

1.11 及更早版本禁止将 mideleg 的任何位设置为只读一。平台标准可能总是添加此类限制。

陷阱永远不会从较高特权模式转换到较低特权模式。例如，如果 M 模式已将非法指令异常委托给 S 模式，并且 M 模式软件稍后执行了非法指令，则陷阱将在 M 模式中处理，而不是委托给 S 模式。相比之下，陷阱可以水平处理。使用相同的例子，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，则陷阱将在 S 模式中处理。

委托的中断会导致中断在委托者特权级别被屏蔽。例如，如果通过设置 mideleg[5] 将监管定时器中断（STI）委托给 S 模式，则在 M 模式中执行时不会处理 STI。相比之下，如果 mideleg[5] 被清除，则 STI 可以在任何模式中处理，并且无论当前模式如何，都会将控制权转移到 M 模式。

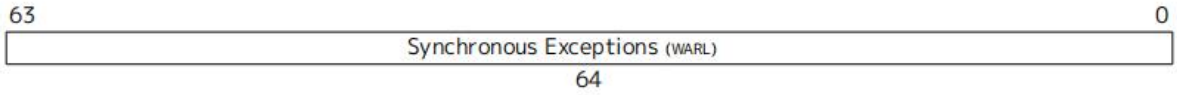


图 3.9 机器异常委托注册 medeleg

medeleg 为每个同步异常分配了一个位位置，位置的索引等于 mcause 寄存器中返回的值（即，设置位 8 允许将 U 模式环境调用委托给较低特权的陷阱处理程序）。

当 XLEN=32 时，medelegh 是一个 32 位的读/写寄存器，它别名为 medeleg 的位 63:32。

当 XLEN=64 时，medelegh 寄存器不存在。

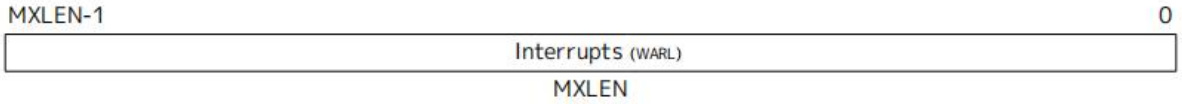


图 3.10 机器中断委托寄存器

mideleg 保存了各个中断的陷阱委托位，其位的布局与 mip 寄存器中的位相匹配（即，STIP 中断委托控制位于位 5）。

对于在较低特权模式中不可能发生的异常，相应的 medeleg 位应为只读 0。具体而言，medeleg[11]是只读 0。

3.1.9 机器中断（mip 和 mie）寄存器

mip 寄存器是一个 MXLEN 位的读/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 中报告）对应于 mip 和 mie 中的位 i。位 15:0 仅分配给标准中断原因，而位 16 及以上则指定供平台使用。

指定供平台使用的中断可以根据平台的判断指定为自定义用途。

图3.11 机器中断等待寄存器

图3.12 机器中断使能寄存器

如果满足以下所有条件，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 mstatus 寄存器中的 MIE 位被设置，或者当前特权模式的特权低于 M 模式；(b) 位 i 在 mip 和 mie 中均被设置；以及 (c) 如果寄存器 mideleg 存在，则位 i 未在 mideleg 中设置。

这些中断陷入发生的条件必须在中断在 mip 中变为挂起或停止挂起后的有限时间内进行评估，并且必须在执行 xRET 指令或显式写入这些中断陷入条件明确依赖的 CSR（包括 mip、mie、mstatus 和 mideleg）后立即进行评估。

对 M 模式的中断优先于对较低特权模式的任何中断。

寄存器 mip 中的每个单独位可以是可写的或只读的。当 mip 中的位 i 是可写的时，可以通过向该位写入 0 来清除挂起的中断 i。如果中断 i 可以变为挂起但 mip 中的位 i 是只读的，则实现必须提供其他机制来清除挂起的中断。

如果相应的中断可以变为挂起，则 mie 中的位必须是可写的。不可写的 mie 位必须是只读 0。

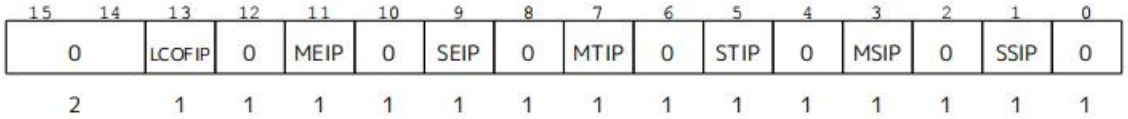


图3.13 mip的标准部分（位15:0）

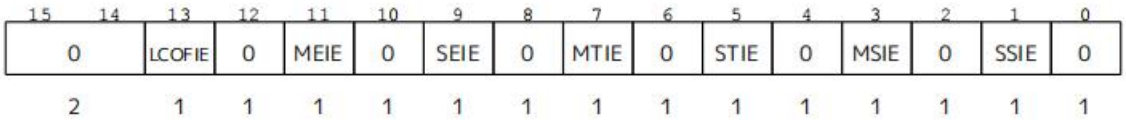


图3.14 mie的标准部分（位15:0）

机器级中断寄存器处理一些根中断源，这些中断源被分配了固定的服务优先级以简化操作，而单独的外部中断控制器可以在更大规模的中断集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中。

不可屏蔽中断不会通过 mip 寄存器显示，因为当执行 NMI 陷阱处理程序时，其存在是隐式已知的。

mip.MEIP 和 mie.MEIE 位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，由平台特定的中断控制器设置和清除。

mip.MTIP 和 mie.MTIE 位是机器定时器中断的中断挂起和中断使能位。MTIP 在 mip 寄存器中是只读的，通过写入内存映射的机器模式定时器比较寄存器来清除。

mip.MSIP 和 mie.MSIE 位是机器级软件中断的中断挂起和中断使能位。MSIP 在 mip 中是只读的，通过访问内存映射的控制寄存器来写入，这些寄存器由远程 硬件线程 用于提供机器级处理器间中断。硬件线程 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 硬件线程，或者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 都可以是只读 0。

如果未实现监管模式，则 mip 的 SEIP、STIP 和 SSIP 位以及 mie 的 SEIE、STIE 和 SSIE 位是只读 0。

如果实现了监管模式，mip.SEIP 和 mie.SEIE 位是监管级外部中断的中断挂起和中断使能位。SEIP 在 mip 中是可写的，可以由 M 模式软件写入以向 S 模式指示外部中断挂起。此外，平台级中断控制器可能会生成监管级外部中断。监管级外部中断的挂起基于软件可写的 SEIP 位和外部中断控制器的信号的逻辑或。当使用 CSR 指令读取 mip 时，返回到 rd 目标寄存器中的 SEIP 位的值是软件可写位和中断控制器信号逻辑或的结果，但中断控制器的信号不用于计算写入 SEIP 的值。只有软件可写的 SEIP 位参与 CSRRS 或 CSRRC 指令的读-修改-写序列。

例如，如果将软件可写的 SEIP 位命名为 B，将外部中断控制器的信号命名为 E，那么如果执行 csrrs t0, mip, t1，则 t0[9] 被写入 B || E，然后 B 被写入 B || t1[9]。如果执行 csrrw t0, mip, t1，则 t0[9] 被写入 B || E，而 B 仅被写入 t1[9]。在这两种情况下，B 都不依赖于 E。

SEIP 字段的行为旨在允许较高特权层干净地模拟外部中断，而不会丢失任何真实的外部中断。因此，CSR 指令的行为与常规 CSR 访问略有不同。

如果实现了监管模式，mip.STIP 和 mie.STIE 位是监管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，可以由 M 模式软件写入以向 S 模式传递定时器中断。

如果实现了监管模式,mip.SSIP 和 mie.SSIE 位是监管级软件中断的中断挂起和中断使能位。SSIP 在 mip 中是可写的，也可以由平台特定的中断控制器设置为 1。如果实现了 Sscofpmf 扩展，mip.LCOFIP 和 mie.LCOFIE 位是本地计数器溢出中断的中断挂起和中断使能位。LCOFIP 在 mip 中是读写的，并反映由于任何 mhpmeventn.OF 位被设置而导致的本地计数器溢出中断请求的发生。如果未实现 Sscofpmf 扩展，mip.LCOFIP 和 mie.LCOFIE 是只读 0。

多个同时发生的目标为 M 模式的中断按以下优先级递减顺序处理：MEI、MSI、MTI、SEI、SSI、STI、LCOFI。

机器级中断的固定优先级排序规则是基于以下理由开发的。

较高特权模式的中断必须在较低特权模式的中断之前处理，以支持抢占。

位 16 及以上的平台特定机器级中断源具有平台特定的优先级，但通常选择具有最高的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切片，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，因为这些位通常由软件写入，并且此位置允许使用带有五位立即数的单条 CSR 指令。

mip 和 mie 寄存器的受限视图作为监管级别的 sip 和 sie 寄存器出现。如果通过设置 mideleg 寄存器中的位将中断委托给 S 模式，则它在 sip 寄存器中可见，并且可以使用 sie 寄存器进行屏蔽。否则，sip 和 sie 中的相应位 w 为只读 0。

3.1.10 硬件性能监控器

M 模式包括一个基本的硬件性能监控设施。mcycle CSR 计算运行硬件线程的处理器核执行的时钟周期数。minstret CSR 计算硬件线程已退役的指令数。mcycle 和 minstret 寄存器在所有 RV32 和 RV64 硬件线程 上具有 64 位精度。

计数器寄存器在硬件线程复位后具有任意值，并且可以写入给定值。任何 CSR 写入在写入指令完成之后生效。mcycle CSR 可以在同一核上的 硬件线程 之间共享，在这种情况下，对 mcycle 的写入对这些 硬件线程 可见。平台应提供一种机制来指示哪些 硬件线程共享 mcycle CSR。

硬件性能监控器包括 29 个额外的 64 位事件计数器，mhpmcounter3-mhpmcounter31。事件选择器 CSR，mhpmevent3-mhpmevent31，是 64 位 WARL 寄存器，控制哪个事件导致相应的计数器递增。这些事件的含义由平台定义，但事件 0 定义为“无事件”。所有计数器都应实现，但合法的实现是使计数器及其相应的事件选择器均为只读 0。

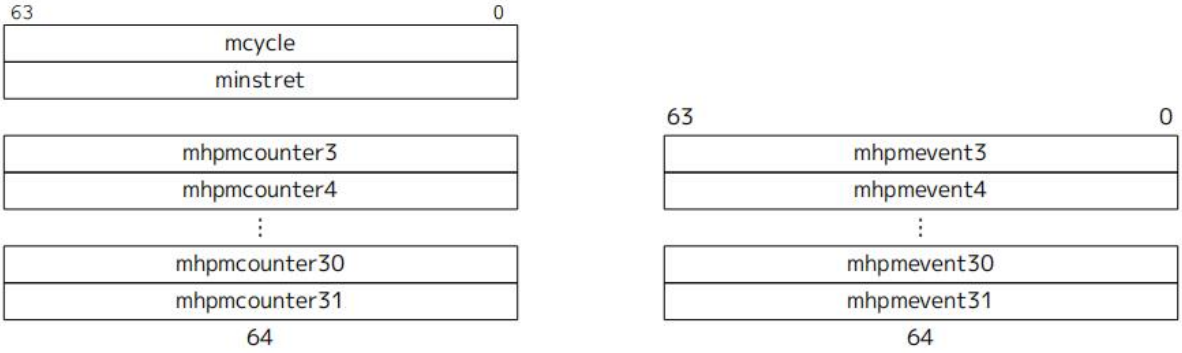


图3.15 硬件性能监控计数器

mhpmcounters 是 WARL 寄存器，支持在 RV32 和 RV64 上最多 64 位的精度。

本规范的未来修订版将定义一种机制，用于在硬件性能监控计数器溢出时生成中断。

当 XLEN=32 时，读取 mcycle、minstret、mhpmcountern 和 mhpmeventn CSR 返回相应寄存器的位 31-0，写入仅更改位 31-0；读取 mcycleh、minstreth、mhpmcounternh 和 mhpmeventnh CSR 返回相应寄存器的位 63-32,写入仅更改位 63-32。只有在实现了 Sscofpmf 扩展时,才会提供 mhpmeventnh CSR。

3.1.11 机器计数器使能（mcounteren）寄存器

计数器使能 mcounteren 寄存器是一个 32 位寄存器，用于控制硬件性能监控计数器对下一个较低特权模式的可用性。

图3.16 Counter-enable 注册

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操作不会影响底层计数器，即使不可访问时计数器仍会继续递增。

当 mcounteren 寄存器中的 CY、TM、IR 或 HPMn 位被清除时，在 S 模式或 U 模式下执行时尝试读取 cycle、time、instret 或 hpmcountern 寄存器将导致非法指令异常。当这些位中的某一位被设置时，允许在下一个实现的特权模式（如果实现了 S 模式则为 S 模式，否则为 U 模式）中访问相应的寄存器。

计数器使能位以最少的硬件支持两种常见用例。对于不需要高性能定时器和计数器的 硬件线程，机器模式软件可以捕获访问并在软件中实现所有功能。对于需要高性能定时器和计数器但不关心混淆底层硬件计数器的 硬件线程，计数器可以直接暴露给较低特权模式。

cycle、instret 和 hpmcountern CSR 分别是 mcycle、minstret 和 mhpmcounter n 的只读影子。

time CSR 是内存映射 mtime 寄存器的只读影子。类似地，当 XLEN=32 时，cycleh、instreth 和 hpmcounternh CSR 分别是 mcycleh、minstreth 和 mhpmcounternh 的只读影子。当 XLEN=32 时，timeh CSR 是内存映射 mtime 寄存器高 32 位的只读影子，而 time 仅影子 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，或者在M模式软件中代表较低特权模式模拟此功能。

在具有 U 模式的 硬件线程 中，必须实现 mcounteren，但所有字段都是 WARL 并且可以是只读 0，表示在较低特权模式下执行时读取相应计数器将导致非法指令异常。在没有 U 模式的 硬件线程 中，mcounteren 寄存器不应存在。

3.1.12 机器计数器禁止（mcountinhibit）寄存器

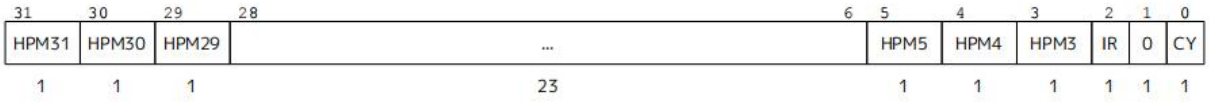


图3.17 反抑制寄存器不受限制

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。此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 位被清除时，mcycle、minstret 或 mhpmcountern 寄存器会照常递增。当 CY、IR 或 HPMn 位被设置时，相应的计数器不会递增。

mcycle CSR 可以在同一核上的 硬件线程 之间共享，在这种情况下，mcountinhibit.CY 字段也在这些 硬件线程 之间共享，因此对 mcountinhibit.CY 的写入对这些 硬件线程 可见。如果未实现 mcountinhibit 寄存器，则实现的行为类似于该寄存器设置为 0。

当不需要 mcycle 和 minstret 计数器时，建议有条件地禁止它们以减少能耗。提供一个 CSR 来禁止所有计数器，这还允许对计数器进行原子采样。

由于 mtime 计数器可以在多个核之间共享，因此不能使用mcountinhibit机制禁止它。

3.1.13 机器临时（mscratch）寄存器

mscratch 寄存器是一个 MXLEN 位的读/写寄存器，专供机器模式使用。通常，它用于保存指向机器模式 硬件线程 本地上下文空间的指针，并在进入 M 模式陷阱处理程序时与用户寄存器交换。

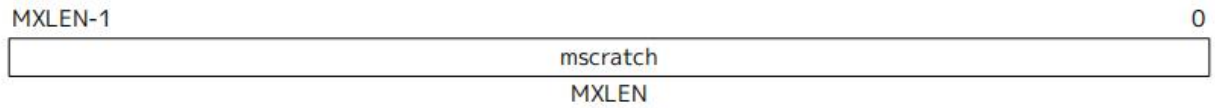


图3.18 机器模式划痕寄存器

MIPS ISA分配了两个用户寄存器（k0/k1）供操作系统使用。尽管 MIPS 的方案提供了快速且简单的实现，但它也减少了可用的用户寄存器，并且无法扩展到更多的特权级别或嵌套陷阱。它还可能要求在返回到用户级别之前清除这两个寄存器，以避免潜在的安全漏洞并提供确定的调试行为。

RISC-V 用户级 ISA 的设计旨在支持多种可能的特权系统环境，因此不希望在用户级 ISA 中引入任何依赖于操作系统的特性。RISC-V 的 CSR 交换指令可以快速将值保存/恢复到 mscratch 寄存器中。与 MIPS 设计不同，操作系统可以依赖在用户上下文运行时将值保存在 mscratch 寄存器中。

3.1.14 异常程序计数器（mepc）寄存器

mepc 是一个 MXLEN 位的读/写寄存器，其格式如 mepcreg 所示。mepc 的最低位（mepc[0]）始终为 0。在仅支持 IALIGN=32 的实现中，mepc 的最低两位（mepc[1:0]）始终为 0。

如果实现允许 IALIGN 为 16 或 32（例如通过更改 CSR misa），那么每当 IALIGN=32 时，mepc[1]位在读取时会被屏蔽，使其看起来为 0。这种屏蔽也发生在 MRET 指令的隐式读取中。尽管被屏蔽，mepc[1] 在 IALIGN=32 时仍然可写。

mepc 是一个 WARL 寄存器，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 能够保存的其他无效地址。

当地址转换未生效时，虚拟地址和物理地址是相等的。因此，mepc 必须能够表示的地址集包括可以用作有效pc 或有效地址的物理地址集。

当发生陷入 M 模式的陷阱时，mepc 会被写入被中断或遇到异常的指令的虚拟地址。否则，mepc 不会被实现自动写入，尽管软件可以显式地写入它。

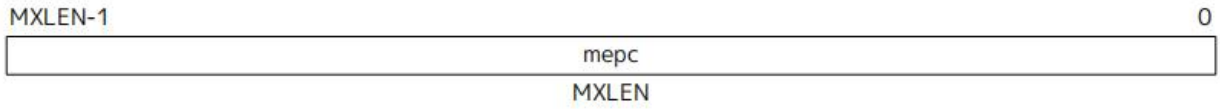
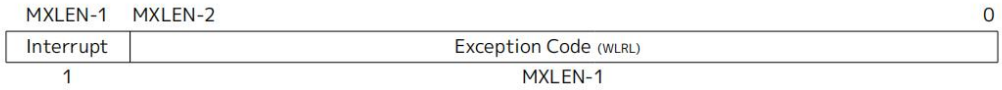


图3.19 机器异常程序计数器寄存器

3.1.15 机器原因（mcause）寄存器

mcause 寄存器是一个 MXLEN 位的读/写寄存器，其格式如下图所示。当发生陷入 M 模式的陷阱时，mcause 会被写入一个代码，指示导致陷阱的事件。否则，mcause 不会被实现自动写入，尽管软件可以显式地写入它。

如果陷阱是由中断引起的，则 mcause 寄存器中的中断位会被置位。异常代码字段包含一个代码，用于标识最近的异常或中断。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，因此仅保证支持已定义的异常代码。



请注意，加载（load）和加载保留（load-reserved）指令会生成加载异常，而存储（store）、条件存储（store-conditional）以及原子内存操作（AMO）指令会生成存储/AMO 异常。

可以通过对 mcause 寄存器值的符号进行一次分支操作，将中断与其他陷阱区分开来。通过左移操作可以移除中断位，并对异常代码进行缩放，以便索引到陷阱向量表中。

不会将特权指令异常与非法指令异常区分开来。这简化了架构，同时也隐藏了实现中支持哪些更高特

权指令的细节。处理陷阱的特权级别可以实现一种策略，决定是否需要区分这些异常，以及如果需要，某个给定的操作码应被视为非法指令还是特权指令。

如果一条指令可能引发多个同步异常，则按照 表 3.7 中优先级递减的顺序决定哪个异常被捕获并报告在 mcause 中。任何自定义同步异常的优先级由实现定义。

表 3.6 陷阱机器原因寄存器值

中断	异常代码	描述
1	0	保留
1	1	监控软件中断
1	2	保留
1	3	机器软件中断
1	4	保留
1	5	监控定时器中断
1	6	保留
1	7	机器定时器中断
1	8	保留
1	9	主管外部中断
1	10	保留
1	11	机器外部中断
1	12	保留
1	13	Counter-overflow 中断
1	14-15	保留
1	≥16	指定作平台用途
0	0	指令地址错位
0	1	指令存取故障
0	2	非法指令
0	3	断点
0	4	负载地址错位
0	5	负载接入故障
0	6	仓库/AMO 地址未对齐
0	7	存储/AMO 访问故障
0	8	u 模式的环境调用
0	9	s 模式的环境调用
0	10	保留
0	11	从 m 模式调用环境
0	12	说明页故障
0	13	加载页面故障
0	14	保留
0	15	Store/AMO 页面错误
0	16	双重陷阱
0	17	保留
0	18	软件检查
0	19	硬件错误

0	20-23	保留
0	24-31	指定自定义使用
0	32-47	保留
0	48-63	指定自定义使用
0	≥64	保留

表 3.7 同步异常优先级（降序排列）

优先级	异常代码	描述
最高	3	指令地址断点
	12，1	在指令地址转换期间：首次遇到的页错误或访问错误
	1	使用指令的物理地址：指令访问错误
	2	非法指令
	0	指令地址未对齐
	8，9，11	环境调用
	3	环境断点
	3	加载/存储/AMO 地址断点
	4，6	可选：加载/存储/AMO 地址未对齐
	13，15，5，7	在显式内存访问的地址转换期间：首次遇到的页错误或访问错误
	5，7	使用显式内存访问的物理地址：加载/存储/AMO 访问错误
最低	4，6	如果优先级不高：加载/存储/AMO 地址未对齐

当一个虚拟地址被转换为物理地址时，地址转换算法决定了可能会引发哪些特定的异常。加载/存储/原子内存操作（AMO）地址未对齐异常（address-misaligned）的优先级可能高于或低于加载/存储/AMO 页面错误（page-fault）和访问错误（access-fault）异常。

加载/存储/AMO 地址未对齐异常与页面错误异常的相对优先级由实现定义，以灵活适应两种设计需求。对于不支持未对齐访问的实现，可以在不执行地址转换或保护检查的情况下无条件触发地址未对齐异常。而对于仅支持部分物理地址未对齐访问的实现，则必须在确定未对齐访问是否可以继续之前进行地址转换和检查，此时触发页面错误异常或访问错误异常更为合适。

指令地址断点与数据地址断点（也称为监视点）和环境断点异常（由 EBREAK 指令触发）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目标地址未对齐的控制流指令触发的，而不是由取指令的操作触发的。因此，这些异常的优先级低于其他指令地址异常。

软件检查异常（Software Check Exception）是一种同步异常，当违反 ISA 扩展定义的检查和断言时触发，旨在保护软件资产的完整性，包括控制流和内存访问约束等。当触发此异常时，xtval 寄存器会被设置为 0 或由触发异常的扩展定义的信息值。此异常相对于其他同步异常的优先级取决于异

常的原因，并由触发异常的扩展定义。

硬件错误异常（Hardware Error Exception）是一种同步异常，当指令显式或隐式访问损坏或不可纠正的数据时触发。在此上下文中，“数据“包括 RISC-V 硬件线程中使用的所有类型的信息。当发生硬件错误异常时，xepc 寄存器会被设置为尝试访问损坏数据的指令地址，而 xtval 寄存器会被设置为 0 或尝试访问损坏数据的指令取指、加载或存储的虚拟地址。硬件错误异常的优先级由实现定义，但通常期望在发现硬件错误时，在整体优先级顺序的相应位置识别该异常。

3.1.16 机器陷阱值（mtval）寄存器

mtval 寄存器是一个 MXLEN 位的读/写寄存器，其格式如下图所示。当发生陷入 M 模式的陷阱时，mtval 会被设置为 0 或写入异常相关的信息，以协助软件处理陷阱。否则，mtval 不会被实现自动写入，尽管软件可以显式地写入它。硬件平台将指定哪些异常必须将 mtval 设置为有意义的值，哪些异常可以无条件将其设置为 0，以及哪些异常的行为可能取决于触发异常的根本事件。如果硬件平台指定所有异常都不会将 mtval 设置为非 0 值，则 mtval 为只读 0。

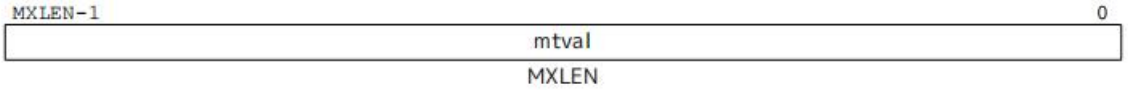


图3.20 机器陷阱值寄存器

如果在指令取指、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，并且 mtval 被写入非 0 值，则 mtval 将包含触发异常的虚拟地址。

当启用基于页面的虚拟内存时，mtval 会被写入触发异常的虚拟地址，即使是由物理内存访问错误异常触发的。这种设计降低了大多数实现的硬件成本，具体而言那些具有硬件页表遍历器的实现。

如果 mtval 在未对齐的加载或存储导致访问错误（access-fault）或页面错误（page-fault）异常时被写入非 0 值，则 mtval 将包含导致错误的访问部分的虚拟地址。如果 mtval 在具有可变长度指令的硬件线程上发生指令访问错误或页面错误异常时被写入非 0 值，则 mtval 将包含导致错误的指令部分的虚拟地址，而 mepc 将指向该指令的起始位置。

mtval 寄存器还可以选择性地用于在非法指令异常（illegal-instruction exception）时返回触发异常的指令位（mepc 指向内存中的触发异常指令）。如果 mtval 在非法指令异常发生时被写入非 0 值，则 mtval 将包含以下最短的内容：

实际的触发异常指令

触发异常指令的前 ILEN 位

触发异常指令的前 MXLEN 位

在非法指令异常时加载到 mtval 中的值是右对齐的，所有未使用的高位会被清 0。

将触发异常的指令捕获到 mtval 中可以减少指令仿真的开销，如果指令未对齐，则可能避免多次部分指令加载；如果使用加载操作将指令取到数据寄存器中，则可能避免数据缓存未命中或缓慢的非缓存访问。如果另一个代理正在操作指令内存（例如在动态翻译系统中），还会存在原子性问题。

一个要求是在触发陷阱之前将整个指令（或至少前 MXLEN 位）取到 mtval 中。这通常不会限制实现，因为实现通常会在尝试解码指令之前获取整个指令，从而避免使软件处理程序复杂化。

mtval 中的值为0表示该功能不受支持，或者获取到了非法的0指令。可以通过从 mepc 指向的指令内

存中加载来区分这两种情况（或者，可以在运行时之前查询系统配置信息以安装适当的陷阱处理程序）。

对于其他陷阱，mtval 被设置为 0，但未来的标准可能会重新定义 mtval 在其他陷阱中的设置。

如果 mtval 不是只读 0，则它是一个 WARL 寄存器，必须能够保存所有有效的虚拟地址和 0 值。它不需要能够保存所有可能的无效地址。在写入 mtval 之前，实现可能会将无效地址转换为 mtval 能够保存的其他无效地址。如果实现了返回触发异常指令位的功能，mtval 还必须能够保存所有小于 2^N 的值，其中 N 是 MXLEN 和 ILEN 中的较小者。

3.1.17 机器配置指针寄存器（mconfigptr）

mconfigptr 寄存器是一个 MXLEN 位的只读 CSR，其格式如下图所示，用于保存配置数据结构的物理地址。软件可以遍历此数据结构以发现有关硬件线程、平台及其配置的信息。mconfigptr 寄存器必须实现，但它可能为 0，以表示配置数据结构不存在或必须使用其他机制来定位它。

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 寄存器可能是硬连线的，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 可能呈现一个内存映射寄存器的值，该寄存器由平台或M模式软件在启动过程开始时进行编程。

3.1.18 机器环境配置（menvcfg）寄存器

menvcfg CSR 是一个 64 位的读/写寄存器，其格式如下图所示，用于控制比 M 模式权限更低的模式的执行环境的某些特性。

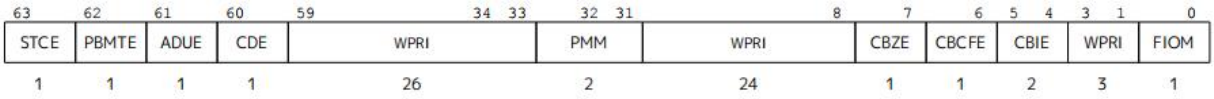


图3.21 机器环境配置寄存器

如果 menvcfg 中的 FIOM 位（Fence of I/O implies Memory，I/O 栅栏隐含内存栅栏）被设置为 1，则在比 M 模式权限更低的模式下执行的 FENCE 指令会被修改，使得对设备 I/O 访问的排序要求也隐含对主内存访问的排序要求。表 3.7 详细说明了当 FIOM=1 时，FENCE 指令的 PI、PO、SI 和 SO 位在比 M 模式权限更低的模式下的修改解释。

类似地，当 FIOM=1 时，在比 M 模式权限更低的模式下，如果访问被排序为设备 I/O 区域的原子指令的 aq 和/或 rl 位被设置，则该指令的排序行为将如同它同时访问了设备 I/O 和内存。

如果不支持 S 模式，或者 satp.MODE 为只读 0（始终为 Bare 模式），则实现可以将 FIOM 设置为只读 0。

表 3.8 FIOM=1 时对非 M 特权级 FENCE 指令前后序集合的修改解释

指令位	置位时的含义
PI	前驱设备输入与存储器读取（隐含 PR 标记）

PO	前驱设备输出与存储器写入（隐含 PW 标记）
SI	后继设备输入与存储器读取（隐含 SR 标记）
SO	后继设备输出与存储器写入（隐含 SW 标记）

menvcfg 中的 FIOM 位是必需的，以便M模式可以模拟第18章中的虚拟机监视器扩展，该扩展在虚拟机监视器 CSR henvcfg 中具有等效的 FIOM 位。

PBMTE 位控制 Svpbmt 扩展是否可用于 S 模式和 G 阶段地址转换（即由 satp 或 hgatp 指向的页表）。当 PBMTE=1 时，Svpbmt 可用于 S 模式和 G 阶段地址转换。当 PBMTE=0 时，实现的行为就像未实现 Svpbmt 一样。如果未实现 Svpbmt，则 PBMTE 为只读 0。此外，对于具有虚拟机监视器扩展的实现，如果 menvcfg.PBMTE 为 0，则 henvcfg.PBMTE 为只读 0。

在更改 menvcfg.PBMTE 后,执行一条 rs1=x0 和 rs2=x0 的 SFENCE.VMA 指令足以同步地址转换缓存，以反映页表项 PBMT 字段的修改解释。有关实现虚拟机监视器扩展时的额外同步要求，请参见 18.5.3。

如果实现了 Svadu 扩展，ADUE 位控制是否在 S 模式和 G 阶段地址转换期间启用硬件更新 PTE A/D 位。当 ADUE=1 时，在 S 模式地址转换期间启用硬件更新 PTE A/D 位，并且实现的行为就像未为 S 模式地址转换实现 Svade 扩展一样。当实现虚拟机监视器扩展时，如果 ADUE=1，则在 G 阶段地址转换期间启用硬件更新 PTE A/D 位，并且实现的行为就像未为 G 阶段地址转换实现 Svade 扩展一样。当 ADUE=0 时，实现的行为就像为 S 模式和 G 阶段地址转换实现了 Svade 扩展一样。如果未实现 Svadu，则 ADUE 为只读 0。此外，对于具有虚拟机监视器扩展的实现，如果 menvcfg.ADUE 为 0，则 henvcfg.ADUE 为只读 0。

Svade 扩展要求在需要设置 PTE A/D 位时触发页面错误异常，因此在 ADUE=0 时实现了 Svade。

如果实现了 Smcdeleg 扩展，CDE（Counter Delegation Enable，计数器委托启用）位控制 Zicntr 和 Zihpm 计数器是否可以委托给 S 模式。当 CDE=1 时，启用 Smcdeleg 扩展，参见第 9 章。当 CDE=0 时，Smcdeleg 和 Ssccfg 扩展看起来未实现。如果未实现 Smcdeleg，则 CDE 为只读 0。

STCE 字段的定义由 Sstc 扩展提供。

CBZE 字段的定义由 Zicboz 扩展提供。

CBCFE 和 CBIE 字段的定义由 Zicbom 扩展提供。

PMM 字段的定义由 Smnpm 扩展提供。

当 XLEN=32 时，menvcfgh 是一个 32 位读/写寄存器，它别名为 menvcfg 的 63:32 位。 当 XLEN=64 时，menvcfgh 寄存器不存在。

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

3.1.19 机器安全配置（mseccfg）寄存器

mseccfg 是一个可选的 64 位读/写寄存器，其格式如下图所示，用于控制安全特性。

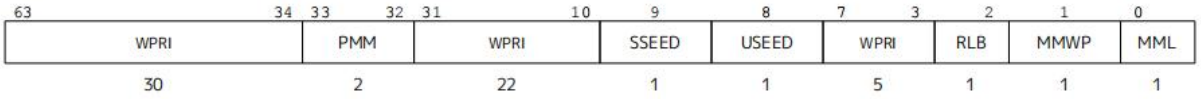


图3.22 机器安全配置寄存器

SSEED 和 USEED 字段的定义将由即将推出的熵源扩展 Zkr 提供。在 Zkr 扩展批准之前，它们在 mseccfg 中的分配可能会发生变化。

RLB、MMWP 和 MML 字段的定义将由即将推出的 PMP 增强扩展 Smepmp 提供。在 Smepmp 扩展批准之前，它们在 mseccfg 中的分配可能会发生变化。

PMM 字段的定义将由即将推出的 Smmpm 扩展提供。在 Smmpm 扩展批准之前，它在 mseccfg 中的分配可能会发生变化。

Zicfilp 扩展在 mseccfg 中添加了 MLPE 字段。当 MLPE 字段为 1 时，Zicfilp 扩展在 M 模式下启用。当 MLPE 字段为 0 时，Zicfilp 扩展在 M 模式下未启用，并且以下规则适用于 M 模式：

硬件线程不会更新 ELP 状态；它保持为 NOLPEXPECTED。

LPAD 指令作为空操作（no-op）执行。

当 XLEN=32 时，mseccfgh 是一个 32 位读/写寄存器，它别名为 mseccfg 的 63:32 位。 当 XLEN=64 时，mseccfgh 寄存器不存在。

3.2 机器级内存映射寄存器

3.2.1 机器定时器寄存器（mtime 与 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 暴露出来。mtime 必须以恒定频率递增，并且平台必须提供一种机制来确定 mtime 计数周期的时长。如果计数器溢出，mtime 寄存器将回绕。

在所有 RV32 和 RV64 系统中，mtime 寄存器具有 64 位精度。平台提供一个 64 位内存映射的机器模式定时器比较寄存器（mtimecmp）。当 mtime 的值大于或等于 mtimecmp 时（将值视为无符号整数），机器定时器中断将挂起。该中断将一直保持挂起状态，直到 mtimecmp 的值大于 mtime（通常是由于写入了 mtimecmp）。只有在中断启用且 mie 寄存器中的 MTIE 位被设置时，才会触发该中断。

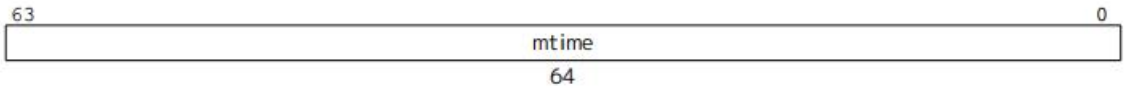


图3.23 机器时间寄存器

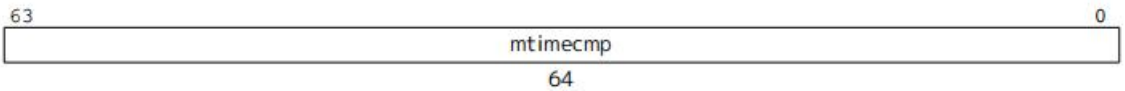


图3.24 机器时间比较寄存器

定时器设施被定义为使用挂钟时间（wall-clock time）而不是周期计数器，以支持现代处理器通过动态电压和频率缩放（DVFS）来节省能量，这些处理器的时钟频率可能高度可变。

精确的实时时钟（RTC）相对昂贵（需要晶体或 MEMS 振荡器），并且必须在系统其余部分断电时继续运行，因此通常系统中只有一个 RTC，且位于与处理器不同的频率/电压域中。因此，RTC 必须由

系统中的所有硬件线程共享，访问 RTC 可能会带来电压电平转换和时钟域跨越的代价。因此，将 mtime 作为内存映射寄存器暴露出来比作为 CSR 更为自然。

较低特权级别没有自己的 timecmp 寄存器。相反，机器模式（M模式）软件可以通过将下一个定时器中断多路复用到 mtimecmp 寄存器中，在 硬件线程 上实现任意数量的虚拟定时器。

简单的固定频率系统可以使用单个时钟来进行周期计数和挂钟计时。

如果 mtime 和 mtimecmp 之间的比较结果发生变化，最终会反映在 MTIP 中，但不一定会立即反映。

如果中断处理程序增加 mtimecmp 然后立即返回，可能会发生虚假的定时器中断，因为在此期间 MTIP 可能尚未下降。所有软件都应假设此事件可能发生，但大多数软件应假设此事件极不可能发生。偶尔发生虚假定时器中断几乎总是比轮询 MTIP 直到其下降更高效。

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

对于 RV64，自然对齐的 64 位内存访问 mtime 和 mtimecmp 寄存器也被支持，并且是原子的。

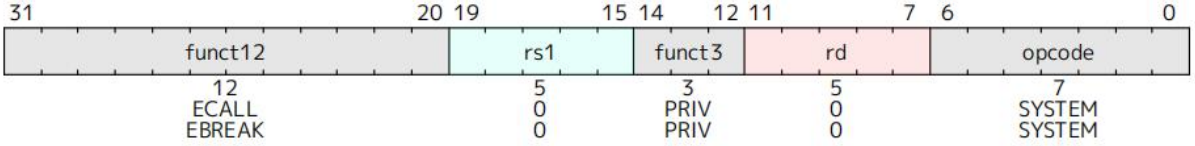
以下是在 RV32 中设置 64 位时间比较值的示例代码，假设内存系统为小端序，并且寄存器位于强有序的 I/O 区域。向 mtimecmp 的低位写入 -1 可以防止 mtimecmp 临时变得小于旧值和新值中的较小值。

```
# 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.
```

time CSR 是内存映射寄存器 mtime 的只读影子寄存器。当 XLEN=32 时，timeh CSR 是内存映射寄存器 mtime 高 32 位的只读影子寄存器，而 time 仅影子 mtime 的低 32 位。
当 mtime 发生变化时，最终会反映在 time 和 timeh 中，但不一定会立即反映。

3.3 机器模式特权指令

3.3.1 环境调用与断点



ECALL 指令用于向支持的执行环境发出请求。当在 U 模式、S 模式或 M 模式下执行时，它分别会生成来自 U 模式的环境调用异常、来自 S 模式的环境调用异常或来自 M 模式的环境调用异常，并且不执行其他操作。

ECALL 根据发起的特权模式生成不同的异常，以便选择性委托环境调用异常。例如，类 Unix 操作系统的典型用例是将来自U模式的环境调用异常委托给S模式，但不委托其他模式的异常。

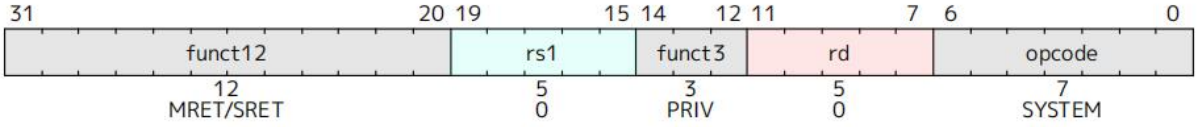
EBREAK 指令由调试器使用，以使控制权转移回调试环境。除非被外部调试环境覆盖，否则 EBREAK 会触发断点异常并且不执行其他操作。

如本手册第一卷中“C“压缩指令扩展所述，C.EBREAK 指令执行与 EBREAK 指令相同的操作。

ECALL 和 EBREAK 会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，而不是下一条指令的地址。由于 ECALL 和 EBREAK 会触发同步异常，因此它们不被视为已完成执行，并且不应增加 minstret CSR。

3.3.2 陷阱返回指令

用于从陷阱返回的指令编码在 PRIV 次要操作码下。



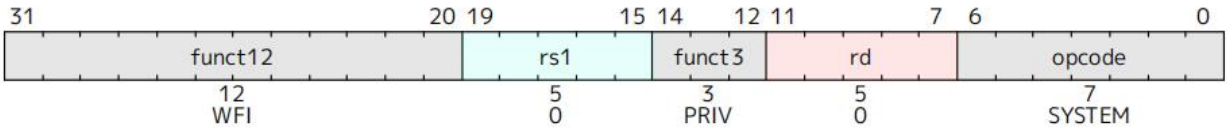
在处理陷阱后返回时，每个特权级别都有单独的陷阱返回指令：MRET 和 SRET。MRET 始终提供。如果支持监管模式（S 模式），则必须提供 SRET，否则应触发非法指令异常。当 mstatus 中的 TSR=1 时，SRET 也应触发非法指令异常，如 virt-control 中所述。xRET 指令可以在特权模式 x 或更高模式下执行，其中执行较低特权的 xRET 指令将弹出相关的较低特权中断使能和特权模式堆栈。除了如 privstack 中所述操作特权堆栈外，xRET 还将 pc 设置为存储在 xepc 寄存器中的值。

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

如果 xRET 指令总是清除 LR 保留，那么使用调试器单步执行 LR/SC 序列将变得不可能。

3.3.3 等待中断

等待中断指令（WFI）通知实现当前硬件线程可以暂停，直到可能需要处理中断为止。执行 WFI 指令还可以用于通知硬件平台，合适的中断应优先路由到此 硬件线程。WFI 在所有特权模式下都可用，并且可以选择性地在 U 模式下可用。当 mstatus 中的 TW=1 时，此指令可能会触发非法指令异常，如图。



如果在 硬件线程 暂停期间存在或随后出现一个已启用的中断，则将在下一条指令上触发中断陷阱，即执行在陷阱处理程序中恢复，并且 mepc = pc + 4。

下一条指令会触发中断陷阱，以便从陷阱处理程序简单返回后执行 WFI 指令之后的代码。

实现允许因任何原因恢复执行，即使没有挂起的已启用中断。因此，合法的实现可以简单地将 WFI 指令实现为 NOP。

如果实现在执行指令时没有暂停 硬件线程，则中断将在包含 WFI 的空闲循环中的某条指令上触发，并且在从处理程序简单返回后，空闲循环将恢复执行。

WFI 指令也可以在中断被禁用时执行。WFI 的操作必须不受 mstatus 中的全局中断位（MIE 和 SIE）以及委托寄存器 mideleg 的影响（即，如果本地启用的中断挂起，硬件线程 必须恢复，即使它已被委托给较低特权模式），但应尊重各个中断使能（例如 MTIE）（即，如果中断挂起但未单独启用，实现应避免恢复 硬件线程）。WFI 还需要为任何特权级别的本地启用挂起中断恢复执行，无论每个特权级别的全局中断使能如何。

如果导致 硬件线程 恢复执行的事件未导致中断触发，则执行将在 pc + 4 处恢复，软件必须确定要采取的操作，包括如果没有可操作的事件则循环回重复 WFI。

通过允许在中断禁用时唤醒，可以调用中断处理程序的替代入口点，而无需保存当前上下文，因为可以在执行 WFI 之前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，软件必须在 WFI 之后的代码中显式检查任何相关但禁用的挂起中断，如果未检测到合适的中断，则应循环回 WFI。可以查询 mip 或 sip 寄存器以分别确定机器模式或监管模式下是否存在任何中断。

WFI 的操作不受委托寄存器设置的影响。

WFI 的定义使得实现可以陷入更高特权模式，例如在遇到 WFI 时立即陷入，或在某些间隔后陷入以启动机器模式向低功耗状态的转换。

相同的“等待事件“模板可能会用于未来可能的扩展，例如等待内存位置更改或消息到达。

3.3.4 自定义 SYSTEM 指令

下图中显示的 SYSTEM 主要操作码的子空间被指定为自定义使用。建议这些指令使用位 29:28 来指定所需的最低特权模式，就像其他 SYSTEM 指令一样。

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

3.4 复位

复位时，硬件线程 的特权模式被设置为M模式。mstatus 字段中的 MIE 和 MPRV 被复位为 0。如果支持小端内存访问，mstatus/mstatush 字段中的 MBE 被复位为 0。misa 寄存器被复位为启用所有支持的扩展集，如 misa 中所述。对于支持“A“标准扩展的实现，不存在有效的加载保留。pc 被设置为实现定义的复位向量。mcause 寄存器被设置为指示复位原因的值。可写的 PMP 寄存器的 A 和 L 字段被复位为 0，除非平台为某些 PMP 寄存器的 A 和 L 字段规定了不同的复位值。如果实现了虚拟机监视器扩展，hgatp.MODE 和 vsatp.MODE 字段被复位为 0。如果实现了 Smrnmi 扩展，mnstatus.NMIE 字段被复位为 0。所有 WARL 字段不包含非法值。如果实现了 Zicfilp 扩展，mseccfg.MLPE 字段被复位为 0。所有其他 硬件线程 状态是未指定的。

复位后的 mcause 值具有实现特定的解释，但对于不区分不同复位条件的实现，应返回值 0。区分不同复位条件的实现应仅使用 0 来表示最完整的复位。

某些设计可能有多种复位原因（例如，上电复位、外部硬复位、检测到欠压、看门狗定时器超时、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 复位值可能与同步异常后的 mcause 值重叠。这种重叠不应产生歧义，因为在复位时，pc 通常被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）仅用于硬件错误条件，并且无论 硬件线程 的中断使能位状态如何，都会立即跳转到实现定义的 NMI 向量，在M模式下运行。mepc 寄存器被写入被中断指令的虚拟地址，mcause 被设置为指示 NMI 来源的值。因此，NMI 可以覆盖活动机器模式中断处理程序中的状态。

NMI 时写入 mcause 的值是实现定义的。mcause 的高位中断位应设置为指示这是一个中断。异常代码 0 保留为“未知原因“，并且不通过 mcause 寄存器区分 NMI 来源的实现应在异常代码中返回 0。与复位不同，NMI 不会复位处理器状态，从而可以诊断、报告并可能控制硬件错误。

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，其中一些对应于内存区域，另一些对应于内存映射的控制寄存器，部分可能不可访问。某些内存区域可能不支持读取、写入或执行；某些可能不支持子字或子块访问；某些可能不支持原子操作；某些可能不支持缓存一致性或可能具有不同的内存模型。同样，内存映射的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副作用方面有所不同。在 RISC-V 系统中，机器物理地址空间每个区域的这些属性和能力称为 物理内存属性（PMA）。本节描述了 RISC-V PMA 术语以及 RISC-V 系统如何实现和检查 PMA。

PMA 是底层硬件的固有属性，在系统运行期间很少更改。与 pmp 中描述的物理内存保护值不同，PMA 不随执行上下文而变化。某些内存区域的 PMA 在芯片设计时是固定的——例如，片上 ROM。其他区域的 PMA 在板级设计时是固定的，例如，取决于连接到片外总线的其他芯片。片外总线也可能支持在每次电源周期（冷插拔）或系统运行时动态（热插拔）更改的设备。某些设备可能在运行时可配置以

支持不同的用途，这意味着不同的 PMA——例如，片上暂存器 RAM 在一个最终应用程序中可能被一个核私有缓存，而在另一个最终应用程序中可能作为共享的非缓存内存访问。

物理地址已知后，大多数系统将动态检查某些 PMA，以确保其符合可配置属性。某些操作在所有物理内存地址上不被支持，而另一些操作则依赖于当前的 PMA 属性设置，因此需在执行流水线的后期进行检查。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 向流水线通知这些属性，但这种方法将平台特定信息注入虚拟化层，并且除非在每个物理内存区域的每个页表条目中正确初始化属性，否则可能导致系统错误。此外，可用的页面大小可能不是指定物理内存空间中属性的最佳选择，从而导致地址空间碎片化和昂贵的 TLB 条目的低效使用。

对于 RISC-V，将 PMA 的规范和检查分离到一个单独的硬件结构中，即 PMA 检查器。在许多情况下，属性在系统设计时对于每个物理地址区域是已知的，并且可以硬连线到 PMA 检查器中。如果属性在运行时是可配置的，那么可以提供平台特定的内存映射控制寄存器，以适配平台上每个区域的粒度来指定这些属性，例如，对于可以在缓存和非缓存用途之间灵活划分的片上 SRAM。PMA 检查适用于对物理内存的任何访问，包括经过虚拟到物理内存转换的访问。为了帮助系统调试，我们强烈建议尽可能在 RISC-V 处理器中精确捕获失败的 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，与虚拟内存页面错误异常不同。精确的 PMA 捕获可能并不总是可行的，例如，在探测使用访问失败作为发现机制的一部分的传统总线架构时。在这种情况下，外围设备的错误响应将报告为不精确的总线错误中断。

PMA 还必须可由软件读取，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，例如 DMA 引擎。由于 PMA 与给定物理平台的组织紧密相关，许多细节本质上是平台特定的，软件了解平台 PMA 值的方式也是如此。某些设备，特别是传统总线，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，平台特定的机器模式代码将提取 PMA，并最终使用某种标准表示将此信息呈现给更高级别的低特权软件。

如果平台支持 PMA 的动态重新配置，则将提供一个接口，通过将请求传递给可以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缓存属性可能涉及平台特定的操作，例如缓存刷新，这些操作仅对机器模式可用。

3.6.1 主内存与 I/O 区域

给定内存地址范围的最重要特征是它是否包含常规主内存或 I/O 设备。常规主内存必须具有以下属性，而 I/O 设备可以具有更广泛的属性。不符合常规主内存的内存区域，例如设备暂存器 RAM，被归类为 I/O 区域。

本规范先前版本中称为 空闲 区域的区域不再是一个独立的类别；它们现在被描述为不可访问的 I/O 区域（即缺乏读取、写入和执行权限）。不可访问的主内存区域也是允许的。

3.6.2 支持的访问类型 PMA

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访问。

尽管在 RISC-V 硬件线程 上运行的软件不能直接生成对内存的突发访问，但软件可能需要编程 DMA 引擎以访问 I/O 设备，因此可能需要知道支持的访问大小。

主内存区域始终支持所有连接设备所需的访问宽度的读取和写入，并且可以指定是否支持指令取指。

某些平台可能要求所有主内存支持指令取指。其他平台可能禁止从某些主内存区域取指。

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，但必须能够使用主内存支持的类型运行。

I/O 区域可以指定支持哪些数据宽度的读取、写入或执行访问组合。

对于具有基于页面的虚拟内存的系统，I/O 和内存区域可以指定支持哪些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支持页表遍历。

原子性 PMA 描述了该地址区域支持哪些原子指令。对原子指令的支持分为两类：LR/SC 和 AMO。

某些平台可能要求所有可缓存主内存支持连接处理器所需的所有原子操作。

3.6.3 原子性物理内存属性

3.6.3.1 AMO PMA

在 AMO 中，有四个支持级别：AMONone、AMOSwap、AMOLogical 和 AMOArithmetic。AMONone 表示不支持任何 AMO 操作。AMOSwap 表示仅在此地址范围内支持 amoswap 指令。AMOLogical 表示支持交换指令以及所有逻辑 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该宽度的读取和写入，则支持自然对齐的 AMO。主内存和 I/O 区域可能仅支持处理器支持的原子操作的子集或不支持任何原子操作如表 3.8。

表 3.9 I/O 区域支持的原子内存操作(AMO)类别

AMO Class	支持的操作
AMONone	None
AMOSwap	amoswap
AMOLogical	above + amoand, amoor, amoxor
AMOArithmetic	above + amoadd, amomin, amomax, amominu, amomaxu

建议尽可能为 I/O 区域提供至少 AMOLogical 支持。

3.6.3.2 可保留性 PMA

对于 LR/SC，有三个支持级别，表示可保留性和最终性属性的组合：RsrvNone、RsrvNonEventual 和 RsrvEventual。RsrvNone 表示不支持任何 LR/SC 操作（该位置不可保留）。RsrvNonEventual 表示支持操作（该位置可保留），但不提供非特权 ISA 规范中描述的最终成功保证。RsrvEventual 表示支持操作并提供最终成功保证。

建议尽可能为主内存区域提供 RsrvEventual 支持。大多数 I/O 区域不支持 LR/SC 访问，因为这些访问最方便地建立在缓存一致性方案之上，但某些区域可能支持 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缺乏进展时使用的备用回退机制。

3.6.4 未对齐原子性粒度 PMA

未对齐原子性粒度 PMA 为未对齐的 AMO 提供有限支持。如果存在此 PMA，则它指定 未对齐原子性粒度 的大小，即自然对齐的 2 的幂字节数。此 PMA 的具体支持值由 MAGNN 表示，例如，MAG16 表示未对齐原子性粒度至少为 16 字节。

未对齐原子性粒度 PMA 仅适用于基本 ISA 中定义的 AMO、加载和存储，以及 F、D 和 Q 扩展中定义的不超过 MXLEN 位的加载和存储。

对于该集合中的指令，如果所有访问的字节位于同一未对齐原子性粒度内，则指令不会因地址对齐原因引发异常，并且出于 RVWMO 的目的，指令只会引发一次内存操作——即它将原子执行。

如果未对齐的 AMO 访问未指定未对齐原子性粒度 PMA 的区域，或者并非所有访问的字节位于同一未对齐原子性粒度内，则会引发异常。

对于访问此类区域或并非所有访问字节位于同一原子性粒度内的常规加载和存储，要么引发异常，要么访问继续进行但不保证是原子的。

实现可能会为某些未对齐访问引发访问错误异常，而不是地址未对齐异常，表明不应由陷阱处理程序模拟该指令。

LR/SC 指令不受此 PMA 影响，因此在未对齐时始终引发异常。向量内存访问也不受影响，因此即使包含在未对齐原子性粒度内，也可能非原子执行。隐式访问同样不受此 PMA 影响。

3.6.5 内存排序 PMA

地址空间的区域根据 FENCE 指令和原子指令排序位的目的被分类为 主内存 或 I/O。

一个 硬件线程 对主内存区域的访问不仅可被其他 硬件线程 观察到，还可被具有在主内存系统中发起请求能力的其他设备（例如 DMA 引擎）观察到。一致的主内存区域始终具有 RVWMO 或 RVTSO 内存模型。非一致的主内存区域具有实现定义的内存模型。

一个 硬件线程 对 I/O 区域的访问不仅可被其他 硬件线程 和总线主控设备观察到，还可被目标 I/O 设备观察到，并且 I/O 区域可以以 宽松 或 严格 排序访问。对具有宽松排序的 I/O 区域的访问

通常以类似于 RVWMO 内存区域访问排序的方式被其他 硬件线程 和总线主控设备观察到，如本规范第一卷 A.4.2 节中讨论的那样。相比之下，对具有严格排序的 I/O 区域的访问通常以程序顺序被其他 硬件线程 和总线主控设备观察到。

每个严格排序的 I/O 区域指定一个编号的排序通道，这是一种可以在不同 I/O 区域之间提供排序保证的机制。通道 0 用于仅指示点对点严格排序，其中仅 硬件线程 对单个关联 I/O 区域的访问是严格排序的。

通道 1 用于提供跨所有 I/O 区域的全局严格排序。硬件线程 对与通道 1 关联的任何 I/O 区域的任何访问只能以程序顺序被所有其他 硬件线程 和 I/O 设备观察到，包括相对于该 硬件线程 对宽松 I/O 区域或具有不同通道号的严格排序 I/O 区域的访问。换句话说，对通道 1 中区域的任何访问等效于在执行指令之前和之后执行 fence io,io 指令。

其他较大的通道号为该 硬件线程 对具有相同通道号的任何区域的访问提供程序排序。系统可能支持每个内存区域上排序属性的动态配置。

严格排序可用于提高与遗留设备驱动程序代码的兼容性，或在已知实现不会重新排序访问时，通过插入显式排序指令来提高性能。

本地严格排序（通道 0）是严格排序的默认形式，因为如果 硬件线程 和 I/O 设备之间只有一条按序通信路径，通常很容易提供。

通常，如果不同的严格排序 I/O 区域共享相同的互连路径并且路径不会重新排序请求，则它们可以共享相同的排序通道而无需额外的排序硬件。

3.6.6 一致性和可缓存性 PMA

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一致代理可见。一致性不应与系统的内存一致性模型混淆，后者定义了在给定整个内存系统的读写历史的情况下，内存读取可以返回哪些值。在 RISC-V 平台中，由于软件复杂性、性能和能耗影响，不鼓励使用硬件非一致区域。

内存区域的可缓存性不应影响软件对该区域的视图，除非反映在其他 PMA 中的差异，例如主内存与 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，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可缓存性设置，则平台特定的机器模式例程将更改设置并在必要时刷新缓存，因此系统仅在可缓存性设置之间的转换期间不一致。这种过渡状态不应对较低特权级别可见。

对于未被任何代理缓存的共享内存区域，提供一致性是直接的。此类区域的 PMA 只需指示它不应缓存在私有或共享缓存中。

对于只读区域，一致性是自然的，这些区域可以被多个代理安全地缓存，无需复杂的缓存一致性方案。此区域的 PMA 将表明它支持缓存，但不支持写入。

某些读写区域可能仅由单个代理访问，在这种情况下，它们可以被该代理私有缓存，而无需一致性方案。此类区域的 PMA 将指示它们可以被缓存。数据也可以缓存在共享缓存中，因为其他代理不应访问该区域。

如果代理可以缓存可由其他代理（无论是缓存还是非缓存）访问的读写区域，则需要缓存一致性方

案以避免使用过时的值。在缺乏硬件缓存一致性的区域（硬件非一致区域）中，缓存一致性可以完全在软件中实现，但软件一致性方案 notoriously 难以正确实现，并且由于需要保守的软件导向缓存刷新，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，但对软件来说是透明的。

对于每个硬件缓存一致区域，PMA 将指示该区域是一致的，并且如果系统有多个一致性控制器，则指示使用哪个硬件一致性控制器。对于某些系统，一致性控制器可能是外部共享缓存，它本身可能会分层访问更外部的缓存一致性控制器。

平台内的大多数内存区域对软件来说都是一致的，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，则对该区域的访问必须由内存本身满足，而不是由任何缓存满足。

对于具有可缓存性控制机制的实现，可能会出现程序以不可缓存方式访问当前缓存驻留的内存位置的情况。在这种情况下，必须忽略缓存副本。此约束是必要的，以防止更高特权模式的推测性缓存重新填充影响较低特权模式的不可缓存访问的行为。

3.6.7 幂等性 PMA

幂等性 PMA 描述了对地址区域的读取和写入是否是幂等的。主内存区域被假定为幂等的。对于 I/O 区域，读取和写入的幂等性可以分别指定（例如，读取是幂等的，但写入不是）。如果访问是非幂等的，即任何读取或写入访问都可能产生副作用，则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排序变化不被视为副作用。

虽然硬件应始终设计为避免对标记为非幂等的内存区域进行推测性或冗余访问，但还必须确保软件或编译器优化不会生成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访问。对此类区域的未对齐访问应引发访问错误异常，而不是地址未对齐异常，表明软件不应使用多个较小的访问来模拟未对齐访问，这可能会导致意外的副作用。

对于非幂等区域，隐式读取和写入不得提前或推测性执行，但以下情况除外。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式读取地址的自然对齐的 2 的幂区域内的任何字节。此外，当执行非推测性指令取指时，允许实现额外读取 下一个 相同大小的自然对齐的 2 的幂区域内的任何字节（区域地址取模 2^{XLEN}）。这些额外读取的结果可用于满足后续的提前或推测性隐式读取。这些自然对齐的 2 的幂区域的大小是实现定义的，但对于具有基于页面的虚拟内存的系统，不得超过支持的最小页面大小。

3.7 物理内存保护

为了支持安全处理和限制故障，希望限制在 硬件线程 上运行的软件可访问的物理地址。一个可选的物理内存保护（PMP）单元提供了每个 硬件线程 的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（读取、写入、执行）。PMP 值与 pma 中描述的 PMA 检查并行进行。

PMP 访问控制设置的粒度取决于平台，但标准的 PMP 编码支持小至四个字节的区域。某些区域的权限可以是硬连线的，例如，某些区域可能仅在机器模式下可见，而在较低特权层中不可见。

平台对物理内存保护的需求差异很大，某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 检查适用于所有有效特权模式为 S 或 U 的访问，包括 S 和 U 模式下的指令取指和数据访问，以及当 mstatus 中的 MPRV 位被设置且 mstatus 中的 MPP 字段包含 S 或 U 时的 M 模式数据访问。PMP 检查还适用于虚拟地址转换的页表访问，其有效特权模式为 S。。PMP 检查可选用于 M 模式访问。此时，PMP 寄存器被锁定，只有在 硬件线程 复位后才能更改。PMP 能为 S 和 U 模式赋予权限，默认二者无权限；也能撤销 M 模式的权限，默认该模式有完全权限。PMP 违规总是在处理器上精确捕获。

3.7.1 物理内存保护 CSR

PMP 条目由一个 8 位配置寄存器和一个 MXLEN 位地址寄存器描述。某些 PMP 设置还使用与前一个 PMP 条目关联的地址寄存器。最多支持 64 个 PMP 条目。实现可以实现 0、16 或 64 个 PMP 条目；必须首先实现编号最低的 PMP 条目。所有 PMP CSR 字段都是 WARL，并且可能是只读 0。PMP CSR 仅对 M 模式可访问。

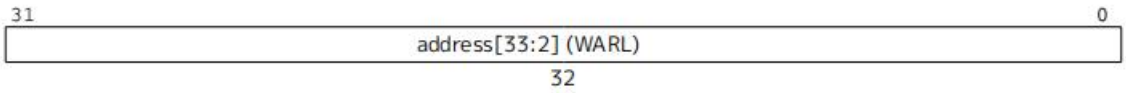


图3.25 PMP地址寄存器格式，RV32

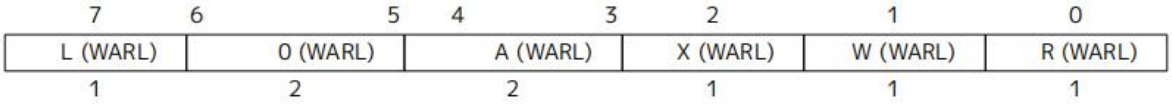
图3.26 PMP地址寄存器格式，RV64

PMP 配置寄存器密集打包到 CSR 中，以最小化上下文切换时间。对于 RV32，十六个 CSR，pmpcfg0 - pmpcfg15，保存了 64 个 PMP 条目的配置 pmp0cfg - pmp63cfg，如图 3.25 所示。对于 RV64，八个偶数编号的 CSR，pmpcfg0，pmpcfg2，…，pmpcfg14，保存了 64 个 PMP 条目的配置，如 图 3.26 所示。对于 RV64，奇数编号的配置寄存器，pmpcfg1，pmpcfg3，…，pmpcfg15，是非法的。

RV64 硬件线程 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 PMP 条目 8-11 的配置在 RV32 和 RV64 中都出现在 pmpcfg2[31:0] 中。

PMP 地址寄存器是名为 pmpaddr0-pmpaddr63 的 CSR。每个 PMP 地址寄存器编码 RV32 的 34 位物理地址的第 33-2 位，如图 3.25 所示。对于 RV64，每个 PMP 地址寄存器编码 56 位物理地址的第 55-2 位，如图 3.26 所示。并非所有物理地址位都必须实现，因此 pmpaddr 寄存器是 WARL。

sv32 中描述的 Sv32 基于页面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 XLEN 更宽的 RV32 地址。sv39 和 sv48 中描述的 Sv39 和 Sv48 基于页面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。



3.29 PMP配置寄存器格式

R、W 和 X 位在被设置时，分别表示 PMP 条目允许读取、写入和指令执行。当这些位中的某一位被清除时，相应的访问类型将被拒绝。R、W 和 X 字段共同构成一个 WARL 字段，其中 R=0 且 W=1 的组合被保留。其余两个字段 A 和 L 将在以下部分中描述。

尝试从没有执行权限的 PMP 区域取指会引发指令访问错误异常。尝试执行加载或加载保留指令，访问没有读取权限的 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、条件存储或 AMO 指令，访问没有写入权限的 PMP 区域内的物理地址会引发存储访问错误异常。

3.7.1.1 地址匹配

PMP 条目的配置寄存器中的 A 字段编码了相关 PMP 地址寄存器的地址匹配模式。该字段的编码如表 3.9 所示。

当 A=0 时，此 PMP 条目被禁用，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 的幂区域（NAPOT），包括自然对齐的四字节区域（NA4）的特殊情况；以及任意范围的顶部边界（TOR）。这些模式支持四字节粒度,如表 3.9。

表 3.10 PMP 配置寄存器中字段的编码

A	名称	描述
0	OFF	空区域（已禁用）
1	TOR	范围顶部
2	NA4	自然对齐的四字节区域
3	NAPOT	自然对齐的 2 次幂区域，≥8 字节

NAPOT 范围利用相关地址寄存器的低位来编码范围的大小，如表 3.10 所示。

表 3.11 PMP 地址与配置寄存器中的 NAPOT 区域编码方案

pmpaddr	pmpcfg.A	匹配类型和尺寸
yyyy⋯ yyyy	NA4	4-byte NAPOT 范围
yyyy… yyy0	NAPOT	8-byte NAPOT 范围
yyyy… yy01	NAPOT	16-byte NAPOT 范围
yyyy… y011	NAPOT	32-byte NAPOT 范围
…	…	…
yy01… 1111	NAPOT	2XLEN-byte NAPOT 范围
y011… 1111	NAPOT	2XLEN+1-byte NAPOT 范围
0111… 1111	NAPOT	2XLEN+2-byte NAPOT 范围
1111… 1111	NAPOT	2XLEN+3-byte NAPOT 范围

如果选择了 TOR，相关的地址寄存器构成地址范围的顶部，而前一个 PMP 地址寄存器构成地址范围的底部。如果 PMP 条目 i 的 A 字段设置为 TOR,则该条目匹配任何满足 $pmpaddr_{i-1} \leqslant y < pmpaddr_i$ 的

地址 y（无论 $pmpcfg_{i-1}$ 的值如何）。如果 PMP 条目 0 的 A 字段设置为 TOR，则使用 0 作为下限，因此它匹配任何地址 y<pmpaddr0。

If $pmpaddr_{i-1} \geq pmpaddr_i$ and $pmpcfg_i$.A=TOR, then PMP entry i matches no addresses.

如果 $pmpaddr_{i-1} \geq ;pmpaddr_i$ 且 $pmpcfg_i$.A=TOR，则 PMP 条目 i 不匹配任何地址。

软件可以通过将0写入 pmp0cfg，然后将全 1 写入 pmpaddr0，再读取 pmpaddr0 来确定 PMP 粒度。如果 G 是最低有效位设置的索引，则 PMP 粒度为 2^{G+2} 字节。

如果当前 XLEN 大于 MXLEN，则为了地址匹配的目的，PMP 地址寄存器从 MXLEN 位 0 扩展到 XLEN 位。

3.7.1.2 锁定与优先模式

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 条目保持锁定状态，直到 硬件线程 复位。如果 PMP 条目 i 被锁定，则对 pmpicfg 和 pmpaddri 的写入被忽略。此外，如果 PMP 条目 i 被锁定且 pmpicfg.A 设置为 TOR，则对 pmpaddri-1 的写入被忽略。

设置 L 位会锁定 PMP 条目，即使 A 字段设置为 OFF。

除了锁定 PMP 条目外，L 位还指示是否对 M 模式访问强制执行 R/W/X 权限。当 L 位被设置时，这些权限对所有特权模式强制执行。当 L 位被清除时，任何与 PMP 条目匹配的 M 模式访问都会成功；R/W/X 权限仅适用于 S 和 U 模式。

3.7.1.3 优先级与匹配逻辑

PMP 条目是静态优先的。匹配访问的任何字节的最低编号 PMP 条目决定该访问是否成功或失败。匹配的 PMP 条目必须匹配访问的所有字节，否则访问失败，无论 L、R、W 和 X 位的设置如何。例如，如果 PMP 条目配置为匹配四字节范围 0xC－0xF，则对范围 0x8－0xF 的 8 字节访问将失败，假设该 PMP 条目是匹配这些地址的最高优先级条目。

如果 PMP 条目匹配访问的所有字节，则 L、R、W 和 X 位决定访问是否成功或失败。如果 L 位被清除且访问的特权模式为 M，则访问成功。否则，如果 L 位被设置或访问的特权模式为 S 或 U，则仅当与访问类型对应的 R、W 或 X 位被设置时，访问才会成功。

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。

如果至少实现了一个 PMP 条目，但所有 PMP 条目的 A 字段都设置为 OFF，则所有S模式和U模式的内存访问都将失败。

失败的访问会生成指令、加载或存储访问错误异常。请注意，单个指令可能会生成多个访问，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并产生可见的副作用。值得注意的是，引用虚拟内存的指令会被分解为多个访

问。

在某些实现中，未对齐的加载、存储和指令取指也可能被分解为多个访问，其中一些访问可能会在访问错误异常发生之前成功。具体而言，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 检查。对于宽度超过 XLEN 位的存储（例如 RV32D 中的 FSD 指令），即使存储地址是自然对齐的，也可能表现出相同的行为。

3.7.2 物理内存保护与分页

物理内存保护机制旨在与 supervisor 中描述的基于页面的虚拟内存系统结合使用。当启用分页时，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为 S。

具有虚拟内存的实现允许在显式内存访问之前推测性地执行地址转换，并允许将它们缓存在地址转换缓存结构中——包括可能缓存 Bare 转换模式和 M 模式中使用的从有效地址到物理地址的恒等映射。生成的物理地址的 PMP 设置可以在地址转换和显式内存访问之间的任何时间点进行检查（并可能缓存）。因此，当修改 PMP 设置时，M 模式软件必须将 PMP 设置与虚拟内存系统以及任何 PMP 或地址转换缓存同步。这是在写入 PMP CSR 后通过执行带有 rs1=x0 和 rs2=x0 的 SFENCE.VMA 指令来实现的。有关实现虚拟机监视器扩展时的额外同步要求，请参见 hyp-mm-fences。

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

4 “Smstateen/Ssstateen“扩展，V1.0.0

可选 RISC-V 扩展的实现有可能在独立的用户线程之间，或在虚拟机监视器下运行的独立客户操作系统之间打开隐蔽通道。当扩展添加了处理器状态（通常是显式寄存器，但也可能是其他形式的状态）时，就会出现这个问题，而主操作系统或虚拟机监视器并不知道这些状态（因此不会在上下文切换时保存和恢复这些状态），但这些状态可以被一个用户线程或客户操作系统修改/写入，并被另一个用户线程或客户操作系统感知/检查/读取。

例如，RISC-V 的高级中断架构（AIA）为每个硬件线程添加了多达十个监管者级别的 CSR（siselect、sireg、stopi、sseteipnum、sclreipnum、sseteienum、sclreienum、sclaimei、sieh 和 siph），并且还提供了硬件向后兼容旧版、非 AIA 软件的选项。由于一个不了解 AIA 的旧版虚拟机监视器在上下文切换时不会交换 AIA 的新 CSR，这些寄存器可能会被用作运行在该虚拟机监视器上的多个客户操作系统之间的隐蔽通道。尽管传统实践可能认为这种通信通道是无害的，但当今对安全的高度关注要求提供一种方法来堵塞此类通道。

RISC-V 浮点扩展的 f 寄存器和向量扩展的 v 寄存器同样可能成为用户线程之间的隐蔽通道，但 f 寄存器和 v 寄存器的访问由 sstatus 寄存器中的 FS 和 VS 字段控制。即使操作系统不知道向量扩展及其 v 寄存器的存在，当 VS 字段初始化为 0 时（无论是在机器级别还是由操作系统本身初始化 sstatus），对这些寄存器的访问将被阻止。

显然，防止新的用户级 CSR 被用作隐蔽通道的一种方法是为每个相关扩展在 mstatus 或 sstatus 中添加一个“XS“字段，类似于向量扩展的 VS 字段。然而，这并不被认为是解决该问题的通用方案，因为未来可能会有许多扩展添加少量状态。即使使用 64 位的 sstatus（对于 RV32 需要添加 sstatush），也不确定 sstatus 中是否有足够的剩余位来容纳所有未来的用户级扩展。无论如何，没有必要为此扩展 sstatus（并添加 sstatush）。用于堵塞隐蔽通道的“启用“标志通常不需要在用户线程的上下文切换时交换，这使得它们不太适合包含在 sstatus 中。因此，为它们提供了一个新的位置。

4.1 状态启用扩展

Smstateen 和 Ssstateen 扩展共同指定了机器模式和监管者模式的功能。Smstateen 扩展规范包括

mstateen*、sstateen 和 hstateen CSR 及其功能。Ssstateen 扩展规范仅包括 sstateen 和 hstateen CSR 及其功能。

对于 RV64 硬件线程，此扩展在机器级别添加了四个新的 64 位 CSR：mstateen0（机器状态启用 0）、mstateen1、mstateen2 和 mstateen3。

如果实现了监管者模式，则在监管者级别定义了另外四个 CSR：sstateen0、sstateen1、sstateen2 和 sstateen3。

如果实现了虚拟机监视器扩展，则添加了另一组 CSR：hstateen0、hstateen1、hstateen2 和 hstateen3。

对于 RV32，上述寄存器为 32 位，对于机器级别和虚拟机监视器的 CSR，每个寄存器的高 32 位对应一组高半部分 CSR：mstateen0h、mstateen1h、mstateen2h、mstateen3h、hstateen0h、hstateen1h、hstateen2h 和 hstateen3h。对于监管者级别的 sstateen 寄存器，目前没有添加高半部分 CSR，因为预计这些寄存器的上 32 位将始终为 0，如下文所述。

每个 stateen CSR 的位控制对扩展状态的较低特权访问，适用于那些未被认为值得在 sstatus 中拥有完整 XS 字段（如 F 和 V 扩展的 FS 和 VS 字段）的扩展。每个级别提供的寄存器数量为四个，因为相信 4 * 64 = 256 位（用于机器和虚拟机监视器级别）和 4 * 32 = 128 位（用于监管者级别）将在未来许多年内足够使用，甚至可能在 RISC-V ISA 的整个使用期间都足够。具体的数字四是一个折衷方案，一方面避免提供过少的位，另一方面避免过度提供永远不会使用的 CSR。未来可能会将 stateen CSR 的数量翻倍，这将在后面讨论。

每个级别的 stateen 寄存器控制对所有较低特权级别的状态访问，但不控制其自身级别的访问。这与现有的 counteren CSR 控制对性能计数器寄存器的访问方式类似。与 counteren CSR 一样，当 stateen CSR 阻止较低特权级别访问状态时，在这些特权模式之一中尝试执行读取或写入受保护状态的指令将引发非法指令异常，或者如果在 VS 或 VU 模式下执行并且满足虚拟指令异常的条件，则引发虚拟指令异常而不是非法指令异常。

当未实现此扩展时，扩展添加的所有状态均可按该扩展的定义访问。

当 stateen CSR 阻止某个特权模式访问状态时，尝试在该特权模式下执行隐式更新状态而不读取状态的指令可能会或可能不会引发非法指令或虚拟指令异常。此类情况必须通过明确指定来消除歧义。

在某些情况下，stateen CSR 的位将具有双重用途，作为引入受控状态的 ISA 扩展的启用标志。

每个监管者级别 sstateen CSR 的位控制用户级别（从 U 模式或 VU 模式）对扩展状态的访问。意图是从最低有效位(位 0)开始分配 sstateen CSR 的位,直到位 31,然后继续到下一个更高编号的 sstateen CSR。

对于 sstateen CSR 中每个具有定义用途的位，匹配的 mstateen CSR 中的相同位被定义为控制对相同状态的机器级别以下的访问。mstateen CSR 的高 32 位（或对于 RV32，对应的高半部分 CSR）控制对用户级别本质上无法访问的状态的访问，因此在监管者级别的 sstateen CSR 中没有相应的启用位。意图是从最高有效位（位 63）开始分配用于此目的的位，直到位 32，然后继续到下一个更高的 mstateen CSR。如果从 sstateen CSR 的最低有效位分配位的速度足够低，则可以从 mstateen CSR 的最高有效位开始分配位，并在跳转到下一个更高的 mstateen CSR 之前允许其侵占低 32 位。在这种情况下，“侵占“位的位位置将在匹配的 sstateen CSR 中永远为只读 0。

对于虚拟机监视器扩展，hstateen CSR 的编码与 mstateen CSR 相同，只是控制虚拟机（从 VS 和 VU 模式）的访问。

每个 stateen CSR 的标准定义位是 WARL（写任意读合法），并且可能是只读 0 或一，但需满足以下条件。

任何 stateen CSR 中定义为控制状态的位，如果硬件线程未实现该状态，则对该硬件线程为只读 0。同样，所有尚未定义含义的保留位也是只读 0。对于 mstateen CSR 中每个为 0 的位（无论是只读 0 还是设置为 0），匹配的 hstateen 和 sstateen CSR 中的相同位显示为只读 0。对于 hstateen CSR 中每个

为 0 的位（无论是只读 0 还是设置为 0），当在 VS 模式下访问时，sstateen 中的相同位显示为只读 0。

监管者级别 sstateen CSR 中的位不能是只读一，除非匹配的 mstateen CSR 中的相同位是只读一，并且如果存在，匹配的 hstateen CSR 中的相同位也是只读一。hstateen CSR 中的位不能是只读一，除非匹配的 mstateen CSR 中的相同位是只读一。

在复位时，所有可写的 mstateen 位由硬件初始化为 0。如果机器级软件更改了这些值，它负责将 hstateen 和 sstateen CSR 中的相应可写位也初始化为 0。每个特权级别的软件应设置其各自的 stateen CSR，以指示它准备允许较低特权软件访问的状态。对于操作系统和虚拟机监视器，这通常意味着操作系统或虚拟机监视器准备在上下文切换时交换的状态，或以其他方式管理的状态。

对于每个 mstateen CSR，位 63 被定义为控制对匹配的 sstateen 和 hstateen CSR 的访问。即，mstateen0 的位 63 控制对 sstateen0 和 hstateen0 的访问；mstateen1 的位 63 控制对 sstateen1 和 hstateen1 的访问；依此类推。同样，每个 hstateen CSR 的位 63 相应地控制对匹配的 sstateen CSR 的访问。

虚拟机监视器可能需要这种控制来访问 sstateen CSR，如果它必须为虚拟机模拟一个受 sstateen CSR 中位影响的扩展。即使这种模拟不常见，也不应排除。

机器级软件需要相同的控制来能够模拟虚拟机监视器扩展。即，机器级需要控制对监管者级别 sstateen CSR 的访问，以便模拟 hstateen CSR，后者具有这种控制。

每个 mstateen CSR 的位 63 只有在未实现虚拟机监视器扩展且匹配的监管者级别 sstateen CSR 全部为只读 0 时才能为只读 0。在这种情况下，机器级软件应模拟从 S 模式访问受影响的 sstateen CSR 的尝试，忽略写入并返回 0 进行读取。每个 hstateen CSR 的位 63 始终是可写的（不是只读的）。

4.2 机器状态使能 0 寄存器

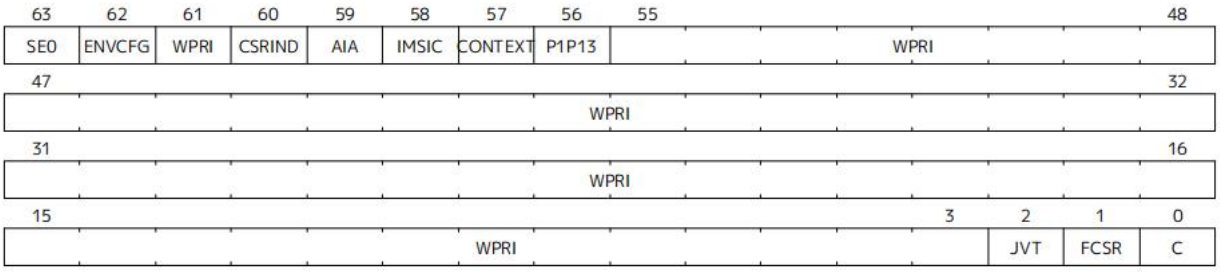


图4.1 机器状态使能0寄存器

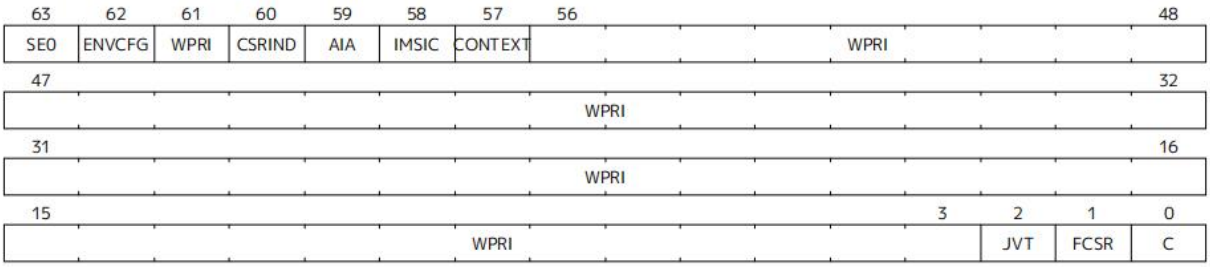


图4.2 超级监督状态启用0寄存器

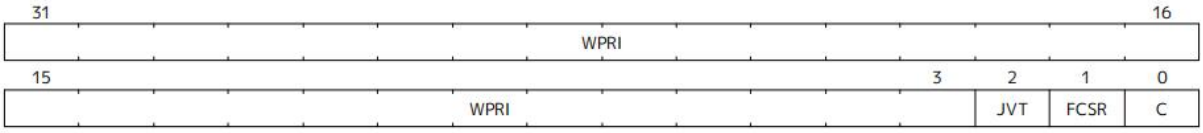


图4.3 监督状态启用0寄存器

C 位控制对任何和所有自定义状态的访问。该位本身不是自定义状态。这些寄存器的 C 位本身不是自定义状态；它是标准 CSR（mstateen0、hstateen0 或 sstateen0）的标准字段。

某些平台可能要求所有可缓存主内存支持连接处理器所需的所有原子操作。

FCSR 位控制在浮点指令操作 x 寄存器而不是 f 寄存器时（如 Zfinx 和相关扩展（Zdinx 等）所规定的那样)对 fcsr 的访问。每当 misa.F = 1 时，mstateen0 的 FCSR 位为只读 0(因此在 hstateen0 和 sstateen0 中也为只读 0）。为了方便起见，当实现了 stateen CSR 且 misa.当 F = 0 时，若控制 stateen0 CSR 的 FCSR 位为零，则所有浮点指令都会触发非法指令陷阱（或虚拟指令陷阱，如果适用），就像它们都访问了 fcsr 一样，无论它们是否真的访问了 fcsr。

JVT 位控制对 Zcmt 扩展提供的 jvt CSR 的访问。

mstateen0 中的 SE0 位控制对 hstateen0、hstateen0h 和 sstateen0 CSR 的访问。hstateen0 中的 SE0 位控制对 sstateen0 CSR 的访问。

mstateen0 中的 ENVCFG 位控制对 henvcfg、henvcfgh 和 senvcfg CSR 的访问。hstateen0 中的 ENVCFG 位控制对 senvcfg CSR 的访问。

mstateen0 中的 CSRIND 位控制对 Sscsrind 扩展提供的 siselect、sireg*、vsiselect 和 vsireg* CSR 的访问。hstateen0 中的 CSRIND 位控制对 Sscsrind 扩展提供的 siselect 和 sireg*（实际上是 vsiselect 和 vsireg*）CSR 的访问。

mstateen0 中的 IMSIC 位控制对 IMSIC 状态的访问，包括 Ssaia 扩展提供的 CSR stopei 和 vstopei。hstateen0 中的 IMSIC 位控制对客户 IMSIC 状态的访问，包括 Ssaia 扩展提供的 CSR stopei（实际上是 vstopei）。

将 hstateen0 中的 IMSIC 位设置为0会阻止虚拟机访问 硬件线程 的 IMSIC，就像将 hstatus.VGEIN = 0 一样。

mstateen0 中的 AIA 位控制对 Ssaia 扩展引入的所有状态的访问，这些状态不受 CSRIND 或 IMSIC 位的控制。hstateen0 中的 AIA 位控制对 Ssaia 扩展引入的所有状态的访问，这些状态不受 hstateen0 的 CSRIND 或 IMSIC 位的控制。

mstateen0 中的 CONTEXT 位控制对 Sdtrig 扩展提供的 scontext 和 hcontext CSR 的访问。hstateen0 中的 CONTEXT 位控制对 Sdtrig 扩展提供的 scontext CSR 的访问。mstateen0 中的 P1P13 位控制对特权规范版本 1.13 引入的 hedelegh 的访问。

4.3 用法说明

在复位时，机器级别的 mstateen CSR 的可写位被初始化为 0 后，机器级软件可以设置这些寄存器中的位，以启用较低特权级别对受控状态的访问。这可能是因为机器级软件知道如何交换状态，或者更可能是因为机器级软件不交换监管者级别的环境。（回想一下，mstateen CSR 必须存在的主要原因是机器级别可以模拟虚拟机监视器扩展。当机器级别不模拟虚拟机监视器扩展时，可能不需要将任何实现的 mstateen 位保持为 0。）

如果机器级别将任何可写的 mstateen 位设置为非 0 值，则必须通过将 0 写入匹配的 hstateen CSR

（如果存在）来初始化它们。如果任何设置为 1 的 mstateen 位在 sstateen CSR 中有匹配的位，机器级软件还必须通过将 0 写入这些 sstateen CSR 来初始化它们。通常，机器级软件会希望设置所有 mstateen CSR 的第 63 位，这要求它将 0 写入所有 hstateen CSR。

软件应确保在首次进入监管者级别的操作系统时，所有 sstateen CSR 的可写位都初始化为 0。然后，操作系统可以设置这些寄存器中的位，以启用用户级别对受控状态的访问，大概是因为它知道如何上下文交换状态。

对于由相应 hstateen CSR 的第 63 位允许客户操作系统访问的 sstateen CSR，虚拟机监视器必须将这些 sstateen CSR 包含在为客户操作系统交换的上下文中。当它启动一个新的客户操作系统时，它必须确保这些 sstateen CSR 的可写位初始化为 0，并且必须模拟对任何其他 sstateen CSR 的访问。

如果任何特权级别的软件不支持较低特权级别的多个上下文，则它可以选择通过将全 1 的值写入其级别的 stateen CSR（机器级别的 mstateen CSR，操作系统的 sstateen CSR 和虚拟机监视器的 hstateen CSR）来最大化较低特权级别对所有状态的访问，而无需知道它授予访问权限的所有状态。这是合理的，因为当较低特权级别仅存在一个上下文时，执行上下文之间不存在隐蔽通道的风险。这种情况预计在机器级别很常见，也可能出现在例如仅托管单个客户虚拟机的类型 1 虚拟机监视器中。

如果预计有需求，未来可以通过添加以下内容将 stateen CSR 的数量翻倍：

* 0x38C mstateen4, 0x39C mstateen4h

* 0x38D mstateen5, 0x39D mstateen5h

* 0x38E mstateen6, 0x39E mstateen6h

* 0x38F mstateen7, 0x39F mstateen7h

* 0x18C sstateen4

* 0x18D sstateen5

* 0x18E sstateen6

* 0x18F sstateen7

* 0x68C hstateen4, 0x69C hstateen4h

* 0x68D hstateen5, 0x69D hstateen5h

* 0x68E hstateen6, 0x69E hstateen6h

* 0x68F hstateen7, 0x69F hstateen7h

这些额外的 CSR 并不是原始提案的确定部分，因为尚不清楚它们是否会被需要，并且人们认为第一组寄存器（编号为 0-3）中位的消耗速度足够慢，以至于任何即将到来的短缺都会提前多年被察觉。目前，甚至还不清楚需要多少年才能耗尽仅 mstateen0、sstateen0 和 hstateen0。

5 “Smcsrind/Sscsrind“ 间接 CSR 访问，V1.0.0

5.1 介绍

Smcsrind/Sscsrind 是一个 ISA 扩展,它扩展了最初定义为 Smaia/Ssaia 扩展一部分的间接 CSR 访问机制，以便其他扩展可以使用该机制，而无需对 Smaia/Ssaia 产生不必要的依赖。该扩展带来了两个好处：

提供了一种通过 CSR 访问寄存器数组的方法，而无需在有限的 CSR 地址空间中分配大块地址。

使软件能够通过索引访问寄存器数组中的每个寄存器,而无需为每个寄存器编写 switch 语句的分支。

通过此扩展，CSR 是使用选择值间接访问的，而不是使用标准的 CSR 编号直接访问的。通过一种方法访问的 CSR 可能无法通过另一种方法访问。选择值与 CSR 编号以及 Sdtrig 扩展中的 tselect 值是独立的地址空间。如果某个 CSR 既可以直接访问又可以间接访问，则该 CSR 的选择值与其 CSR 编号无关。

此外，机器级（Machine-level）和监管级（Supervisor-level）的选择值是相互独立的地址空间；然而，扩展可以定义具有相同选择值的机器级和监管级 CSR 为部分或完全别名。这通常适用于可以从机器级委托到监管级的 CSR。

机器级扩展 Smcsrind 涵盖了所有新增的 CSR 以及 硬件线程 在所有特权级别上的所有行为修改。对于监管级环境，扩展 Sscsrind 与 Smcsrind 基本相同，只是排除了机器级 CSR 以及不直接对监管级可见的行为。

5.2 机器级 CSR

序列	特权	宽度	名称	描述
0x350	MRW	XLEN	miselect	机械间接寄存器选择
0x351	MRW	XLEN	mireg	机械间接寄存器别名
0x352	MRW	XLEN	mireg2	机械间接寄存器别名 2
0x353	MRW	XLEN	mireg3	机械间接寄存器别名 3
0x355	MRW	XLEN	mireg4	机械间接寄存器别名 4
0x356	MRW	XLEN	mireg5	机械间接寄存器别名 5
0x357	MRW	XLEN	mireg6	机械间接寄存器别名 6

“mireg*“CSR编号不是连续的，因为miph是CSR编号0x354。

上表中列出的 CSR 提供了一个间接访问寄存器状态的窗口。miselect 的值决定了在读取或写入每个机器间接别名 CSR（mireg*）时访问哪个寄存器。miselect 的值范围被分配给依赖的扩展，这些扩展指定了通过每个 miregi 寄存器访问的寄存器状态，针对每个 miselect 值。miselect 是一个 WARL（Write-Any-Read-Legal）寄存器。

miselect 寄存器至少实现了足够的位数以支持所有实现的 miselect 值（对应于利用

miselect/mireg* 间接访问寄存器状态的已实现扩展）。如果没有实现使用 miselect 的扩展，则 miselect 寄存器可能为只读零。

miselect 的最高有效位（bit XLEN－1）设置为 1 的值仅用于自定义用途，通常用于通过别名 CSR 访问自定义寄存器。miselect 的最高有效位清零的值仅用于标准用途，并保留给标准架构扩展分配。如果 XLEN 发生变化，miselect 的最高有效位将移动到新位置，并保留其之前的值。

实现不需要支持 miselect 的任何自定义值。

在 M 模式下访问 mireg* 时，如果 miselect 持有一个未实现的值，其行为是未指定的（UNSPECIFIED）。

预计实现通常会对此类访问引发非法指令异常，以便将其识别为软件错误。平台规范、配置文件规范和/或特权 ISA 规范可能会对此类访问的行为施加更多限制。

当 miselect 持有一个已分配且已实现范围内的值时，尝试访问 mireg* 会导致特定行为，该行为由分配 miselect 值的扩展定义，针对每个 miselect 和 miregi 的组合。通常情况下，每个 miregi 会访问寄存器状态、访问只读的 0 状态，或者引发非法指令异常。

对于 RV32，如果某个扩展将一个间接访问的寄存器定义为 64 位宽，建议通过 mireg、mireg2 或 mireg3 之一访问寄存器的低 32 位，而通过 mireg4、mireg5 或 mireg6 分别访问寄存器的高 32 位。

定义了六个 *ireg* 寄存器，以确保覆盖开发中扩展的需求，并留有一定的增长空间。例如，对于与计数器 X 关联的 siselect 值，sireg/sireg2 可用于访问 mhpmcounterX/mhpmeventX，而 sireg4/sireg5 则可访问 mhpmcounterXh/mhpmeventXh。六个 *ireg* 寄存器允许每个索引值（*iselect）访问最多 3 个仅适用于 RV32 的 CSR 数组，或者在没有仅适用于 RV32 的 CSR 时，每个索引值最多可访问 6 个 CSR 数组。

5.3 主管级 CSR

序列	特权	宽度	名称	描述
0x150	SRW	XLEN	siselect	主管间接寄存器选择
0x151	SRW	XLEN	sireg	主管间接寄存器别名
0x152	SRW	XLEN	sireg2	主管间接寄存器别名 2
0x153	SRW	XLEN	sireg3	主管间接寄存器别名 3
0x155	SRW	XLEN	sireg4	主管间接寄存器别名 4
0x156	SRW	XLEN	sireg5	主管间接寄存器别名 5
0x157	SRW	XLEN	sireg6	主管间接寄存器别名 6

如果实现了 S 模式，则上表中的 CSR 是必需的。siselect 寄存器至少需要支持 0..0xFFF 的值范围。未来的扩展可能会定义超出此最小范围的值范围。只有在实现了此类扩展的情况下，siselect 才需要支持更大的值。

要求 siselect 的范围为 0－0xFFF，即使大部分或全部空间可能被保留或不可访问，也允许 M 模式在此实现范围内模拟间接访问的寄存器，包括未来可能标准化的寄存器。

当 siselect 的最高有效位被设置时（即第 XLEN－1 位为 1），这些值仅用于自定义用途，通常用于通过别名 CSR 访问自定义寄存器。而当 siselect 的最高有效位未被设置时，这些值仅用于标准用途，并且在分配给标准架构扩展之前是保留的。如果 XLEN 发生变化，siselect 的最高有效位将移动到新的位置，并保留其之前的值。

当从 M 模式或 S 模式访问 sireg* 时，如果 siselect 持有的值在监管者级别未实现，其行为是未指定的。

建议实现在这种情况下引发非法指令异常，以便于 M 模式可能对这些访问进行模拟。

如果在机器级别通过某些设置（例如 CSR menvcfg 中的某些字段）禁用了 S 模式的扩展，则该扩展被视为在监管者级别未实现。

否则，当 siselect 持有一个在标准定义且已实现范围内的数值时，从 M 模式或 S 模式尝试访问 sireg* 会导致特定行为,这些行为由分配 siselect 值的扩展为每对 siselect 和 siregi 组合定义。

通常，每个sireg*i*都会访问寄存器状态、访问只读0状态，或者，除非在虚拟机中执行（下一节将介绍），否则会引发非法指令异常。

需要注意的是，siselect 和 sireg* 的宽度始终是当前的 XLEN，而不是 SXLEN。因此，例如，如果 MXLEN = 64 且 SXLEN = 32，则当当前特权模式为 M（运行 RV64 代码）时，这些寄存器为 64 位；而当特权模式为 S（运行 RV32 代码）时，这些寄存器为 32 位。

5.4 虚拟监督者级 CSR

序列	特权	宽度	名称	描述
0x250	HRW	XLEN	vsiselect	虚拟管理器间接寄存器选择
0x251	HRW	XLEN	vsireg	虚拟管理器间接寄存器别名
0x252	HRW	XLEN	Vsireg2	虚拟管理器间接寄存器别名 2
0x253	HRW	XLEN	Vsireg3	虚拟管理器间接寄存器别名 3
0x255	HRW	XLEN	Vsireg4	虚拟管理器间接寄存器别名 4
0x256	HRW	XLEN	Vsireg5	虚拟管理器间接寄存器别名 5
0x257	HRW	XLEN	Vsireg6	虚拟管理器间接寄存器别名 6

如果实现了管理程序扩展（hypervisor extension），则上表中的 CSR 是必需的。这些 VS CSR 与监督者 CSR（supervisor CSRs）一一对应，并在虚拟机中执行时（在 VS 模式或 VU 模式下）替代这些监督者 CSR。

vsiselect 寄存器至少需要支持值范围 0..0xFFF。未来的扩展可能会定义超出此最小范围的值范围。只有在实现了此类扩展的情况下，vsiselect 才需要支持更大的值。

要求 vsiselect 支持 0－0xFFF 的范围，即使大部分或全部空间可能是保留的或不可访问的，也允许管理程序在此实现范围内模拟间接访问的寄存器，包括未来可能标准化的寄存器。

更一般地，建议 vsiselect 和 siselect 以相同的位数实现。这也可以避免由于 vsiselect 和 siselect 宽度之间的可观察差异而导致的虚拟化漏洞。

vsiselect 的最高有效位（bit XLEN－1＝1）被设置为 1 的值仅用于自定义用途，通常用于通过别名 CSR 访问自定义寄存器。vsiselect 的最高有效位为 0 的值仅用于标准用途，并且在分配给标准架构扩展之前是保留的。如果 XLEN 发生变化，vsiselect 的最高有效位将移动到新的位置，并保留其之前的值。对于别名 CSR sireg* 和 vsireg*，管理程序扩展的常规规则（基于指令是否具有 HS 资格）不适用于何时引发虚拟指令异常。本节中关于 sireg 和 vsireg 的规则适用，除非被下面部分中指定的要求所覆盖，当扩展 Smstateen 也被实现时，下面部分的规则优先于本节。

从 VS 模式或 VU 模式尝试直接访问 vsiselect 或 vsireg*，或从 VU 模式尝试访问 siselect 或 sireg* 时，将引发虚拟指令异常。 在 M 模式或 HS 模式下访问 vsireg*，或在 VS 模式下访问 sireg*（实际上是 vsireg*），而 vsiselect 持有的值在 HS 级别未实现时，行为是未指定的。

建议实现在这种情况下引发非法指令异常，以便于 M 模式可能对这些访问进行模拟。

否则，当 vsiselect 持有一个在标准定义且已实现的范围内的数值时，从具有足够权限的模式尝试访问 vsireg*，或从 VS 模式访问 sireg*（实际上是 vsireg*），将导致特定行为。对于每个 vsiselect 和 vsiregi 的组合，该行为由 vsiselect 值所属的扩展定义。

通常情况下，每个 vsiregi 会访问寄存器状态、访问只读的 0 状态，或引发异常（可能是非法指令异常，或者对于从 VS 模式的某些访问，可能是虚拟指令异常）。当 vsiselect 持有的值在 HS 级别实现但未在 VS 级别实现时，从 VS 模式尝试访问 sireg*（实际上是 vsireg*）通常会引发虚拟指令异常。但在某些特定于扩展的情况下，可能会有更适合的不同行为。

与 siselect 和 sireg* 类似，vsiselect 和 vsireg* 的宽度始终是当前的 XLEN，而不是 VSXLEN。因此，例如，如果 HSXLEN＝64 且 VSXLEN＝32，那么当虚拟机监视器在 HS 模式下（运行 RV64 代码）访问这些寄存器时，它们的宽度为 64 位；而当客户操作系统在 VS 模式下（运行 RV32 代码）访问时，它们的宽度为 32 位。

5.5 通过授权的证书签名请求实现访问控制

如果扩展 Smstateen 与 Smcsrind 一起实现，状态启用寄存器 mstateen0 的第 60 位控制对 siselect、sireg*、vsiselect 和 vsireg* 的访问。当 mstateen0[60]=0 时，从低于 M 模式的权限模式尝试访问这些 CSR 中的任何一个都会导致非法指令异常。与往常一样，状态启用 CSR 不会影响 M 模式下任何状态的可访问性，仅影响较低权限模式。有关更多解释，请参阅扩展 Smstateen 的文档。

其他扩展可能会指定某些 mstateen 位控制通过 siselect＋sireg* 和/或 vsiselect＋vsireg* 间接访问的寄存器的访问权限。然而，无论其他 mstateen 位的值如何，如果 mstateen0[60]＝1，则对于所有从 VS 模式或 VU 模式直接访问 vsiselect 或 vsireg* 的尝试，以及所有从 VU 模式访问 siselect 或 sireg* 的尝试，都会如前一节所述引发虚拟指令异常。

如果实现了虚拟机监视器扩展（hypervisor extension），相同的位也定义在虚拟机监视器 CSR

hstateen0 中，但仅控制对 siselect 和 sireg*（实际上是 vsiselect 和 vsireg*）的访问，这些状态可能对在 VS 或 VU 模式下执行的虚拟机可访问。当 hstateen0[60]=0 且 mstateen0[60]=1 时，所有从 VS 或 VU 模式访问 siselect 或 sireg* 的尝试都会引发虚拟指令异常，而不是非法指令异常，无论 vsiselect 或其他 mstateen 位的值如何。

扩展 Ssstateen 被定义为 Smstateen 的监管者级别视图。因此，Sscsrind 和 Ssstateen 的组合包含了上述为 hstateen0 定义的位，但不包含为 mstateen0 定义的位，因为机器级别的 CSR 对监管者级别不可见。

CSR地址空间是为未来可能的“Sucsrind“扩展保留的，该扩展将间接CSR访问扩展到用户模式。

6 “Smepmp“扩展：用于机器模式下内存访问和执行保护的 PMP 增强功能，V1.0.0

6.1 介绍

能够从较低特权模式（如 U 模式）访问运行在较高特权执行模式（如监管者模式或机器模式）下的进程的内存，引入了一个明显的攻击向量，因为它允许攻击者执行权限提升，并篡改该进程的代码和/或数据。当相反的情况发生时，存在一个不太明显的攻击向量，即攻击者可以篡改无特权/较低特权进程的内存，并诱使高特权进程使用或执行它。

为了防止这种攻击向量，最近的系统中引入了两种机制，称为监管者内存访问保护（SMAP）和监管者内存执行保护（SMEP）。前者防止操作系统访问无特权进程的内存，除非遵循特定的代码路径；后者则始终防止操作系统执行无特权进程的内存。RISC-V 已经通过 sstatus.SUM 位支持 SMAP，并通过始终拒绝执行标记有 U 位的虚拟内存页（在监管者模式（OS）权限下）来支持 SMEP，这是特权规范所要求的。

PMP 条目：一对 pmpcfg[i] / pmpaddr[i]寄存器。

PMP 规则：pmpcfg 寄存器及其关联的 pmpaddr 寄存器的内容，编码了一个有效的受保护物理内存区域，其中 pmpcfg[i].A != OFF，并且如果 pmpcfg[i].A == TOR，则 pmpaddr[i-1] < pmpaddr[i]。

忽略：任何由匹配的 PMP 规则设置的权限都被忽略，并且允许对请求地址范围的所有访问。

强制执行：仅允许在匹配请求地址范围的 PMP 规则中配置的访问类型；失败将导致访问故障异常。

拒绝：任何由匹配的 PMP 规则设置的权限都被忽略，并且不允许对请求地址范围的任何访问；失败将导致访问故障异常

锁定：pmpcfg.L 位设置的 PMP 规则/条目。

PMP 重置：一种重置过程，其中硬件线程的所有 PMP 设置（包括锁定的规则/设置）在将硬件线程（重新）释放给固件/OS/应用程序之前重新初始化为一组安全默认值。

6.1.1 威胁模型

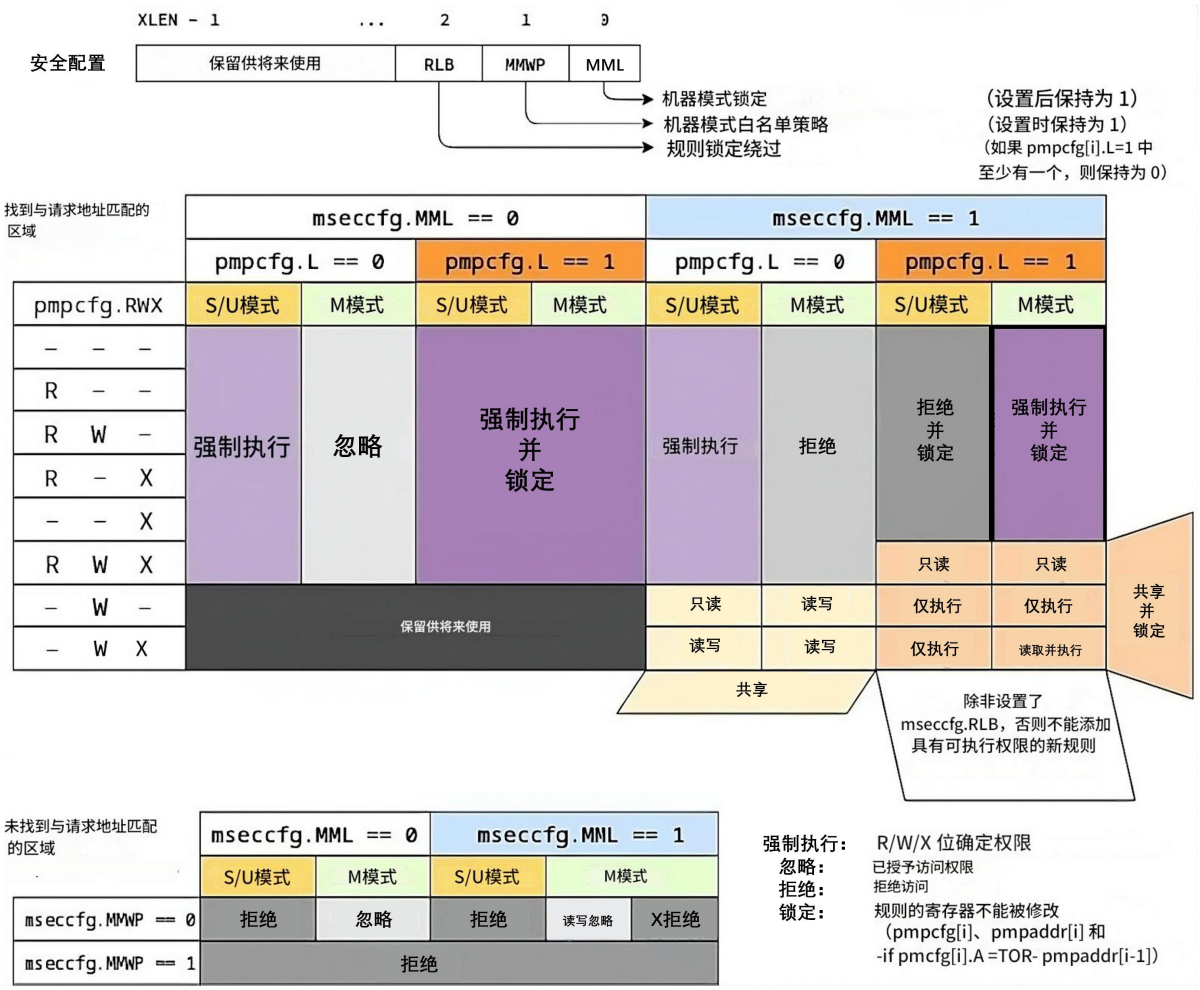
然而，在当前（v1.11）特权规范中，机器模式下没有此类机制可用。无法使 PMP 规则仅在非机器模式下强制执行，而在机器模式下拒绝，以仅允许较低特权模式访问内存区域。只能有一个在所有模式下强制执行的锁定规则，或者一个在非机器模式下强制执行而在机器模式下忽略的规则。因此，对于任何未使用锁定规则保护的物理内存区域，机器模式具有无限制的访问权限，包括执行它的能力。

如果不能保护较低特权模式免受机器模式的影响，就无法防止上述攻击向量。这对于 RISC-V 来说比其他架构更为重要，因为允许实现仅具有机器模式和 U 模式的硬件线程，因此整个操作系统将在机器

模式下运行，而不是不存在的监管者模式。在这种实现中，攻击面大大增加，并且可以在机器模式下执行在监管者模式下执行的相同类型的攻击，而没有任何可用的缓解措施。即使在具有监管者模式的实现中，仍然可能对运行在机器模式下的固件和/或安全监视器进行攻击。

6.2 提案

机器安全配置（mseccfg）是一个新的 RW 机器模式 CSR，用于配置硬件线程上存在的各种安全机制，仅机器模式可访问。它在 RV64 上为 64 位宽，地址为 0x747，在 RV32 上为 0x747（低 32 位），0x757（高 32 位）。本提案中定义的所有 mseccfg 字段都是 WARL，其余位保留供未来标准使用，应始终读取为 0。mseccfg 的复位值是实现特定的，否则如果需要向后兼容，应在硬复位时复位为 0。



在 mseccfg 中，在第 2 位引入了一个称为**规则锁定绕过（mseccfg.RLB）**的字段，其功能如下：

当 mseccfg.RLB 为 1 时，可以移除/修改锁定的 PMP 规则，并且可以编辑锁定的 PMP 条目。

当 mseccfg.RLB 为 0 且任何规则或条目（包括禁用的条目）中的 pmpcfg.L 为 1 时，mseccfg.RLB 保持为 0，并且在 PMP 重置之前，对 mseccfg.RLB 的任何进一步修改都将被忽略。

请注意，此功能旨在用作调试机制，或在启动过程中作为临时解决方案，以简化软件并优化内存和 PMP 规则的分配。在启动过程完成后，应避免在正常操作中使用此功能，因为它会削弱仅 M 模式规则的保护。不需要此功能的供应商可以将此字段硬连线为 0。

在 mseccfg 中，在第 1 位引入了一个称为机器模式白名单策略（mseccfg.MMWP）的字段。这是一个粘性位，意味着一旦设置，直到 PMP 重置之前无法取消设置。当设置时，它将 M 模式访问没有匹配 PMP

规则的内存区域的默认 PMP 策略更改为拒绝，而不是忽略。

在 mseccfg 中，在第 0 位引入了一个称为机器模式锁定（mseccfg.MML）的字段。这是一个粘性位，意味着一旦设置，直到 PMP 重置之前无法取消设置。当 mseccfg.MML 设置时，系统的行为会发生以下变化：

pmpcfg.L 的含义发生变化：它不再标记一个规则为在所有模式下锁定和强制执行，而是当设置时标记一个规则为仅 M 模式，未设置时标记为仅 S/U 模式。先前保留的 pmpcfg.RW=01 编码和 pmpcfg.LRWX=1111 编码现在表示共享区域。

仅在 M 模式下强制执行 M 模式规则，在 S 模式或 U 模式下则拒绝执行。规则一旦设置将保持锁定状态，除非 mseccfg.RLB 被设置，否则在 PMP 重置之前，对其关联的配置寄存器或地址寄存器的任何进一步修改都将被忽略。

仅 S/U 模式规则在监管者和 U 模式下强制执行，在机器模式下拒绝。共享区域规则在所有模式下强制执行，但根据 pmpcfg.L 和 pmpcfg.X 位的设置有不同的限制：

未设置 pmpcfg.L 的共享区域规则可用于在 M 模式和 S/U 模式之间共享数据，因此不可执行。M 模式对该区域具有读/写访问权限，S/U 模式在未设置 pmpcfg.X 时具有读访问权限，或在设置 pmpcfg.X 时具有读/写访问权限。

设置了 pmpcfg.L 的共享区域规则可用于在 M 模式和 S/U 模式之间共享代码，因此不可写。M 模式和 S/U 模式对该区域都具有执行访问权限，如果设置了 pmpcfg.X，M 模式还具有读访问权限。该规则保持锁定，因此除非设置了 mseccfg.RLB，否则在 PMP 重置之前，对其关联的配置或地址寄存器的任何进一步修改都将被忽略。

pmpcfg.LRWX=1111 编码可用于在 M 模式和 S/U 模式之间共享数据，其中两种模式对该区域仅具有只读访问权限。该规则保持锁定，因此除非设置了 mseccfg.RLB，否则在 PMP 重置之前，对其关联的配置或地址寄存器的任何进一步修改都将被忽略。

添加具有可执行权限的规则，无论是仅 M 模式还是锁定的共享区域，都是不可能的，此类 pmpcfg 写入将被忽略，pmpcfg 保持不变。可以通过设置 mseccfg.RLB 暂时解除此限制，例如在启动过程中。

只有在具有匹配的仅 M 模式规则或具有可执行权限的锁定的共享区域规则的内存区域中，才能以机器模式权限执行代码。在没有匹配规则或具有匹配的仅 S/U 模式规则的内存区域中执行代码将被拒绝。

如果未设置 mseccfg.MML，pmpcfg.RW=01 的组合仍保留供未来标准使用。

表 6.1 当 mseccfg.MML 设置时的真值表

PMP 配置寄存器(pmpcfg)位域说明				结果	
L	R	W	X	M Mode	S/U Mode
0	0	0	0	不可访问区域	
0	0	0	1	访问异常	Execute-only region
0	0	1	0	共享数据区域：M 模式可读写，S/U 模式只读	
0	0	1	1	共享数据区域：M 模式与 S/U 模式均可读写	
0	1	0	0	访问异常	只读
0	1	0	1	访问异常	读/执行
0	1	1	0	访问异常	读/写
0	1	1	1	访问异常	读/写/执行
1	0	0	0	锁定不可访问区域	

1	0	0	1		访问异常
1	0	1	0	锁定式共享代码区域：M 模式与 S/U 模式均仅可执行	
1	0	1	1	锁定式共享代码区域：S/U 模式仅执行，M 模式可读/执行	
1	1	0	0	锁定读	访问异常
1	1	0	1	锁定读/执行	访问异常
1	1	1	0	锁定读/写	访问异常
1	1	1	1	锁定式共享数据区域：M 模式与 S/U 模式均为只读	

6.3 SMEPMP 软件探测机制

由于本提案中定义的 mseccfg 字段在设置（如 MMWP/MML）或清除（如 RLB）时会被锁定，因此软件无法通过轮询这些字段来检测 Smepmp 的存在。预期 BootROM 会在早期启动阶段设置 mseccfg.MMWP 和/或 mseccfg.MML，然后跳转到固件。此时，固件可以通过读取 mseccfg 并检查 mseccfg.MMWP 和 mseccfg.MML 的状态，来判断 Smepmp 是否存在。

6.4 设计理由

由于当前规范中没有用于安全和/或全局 PMP 行为设置的 CSR，需要定义一个新的 CSR。这个新的 CSR 将允许在未来添加更多的安全配置选项，并允许开发人员验证本提案中定义的新机制的存在。

在某些用例中，开发人员希望在启动过程中在 M 模式下强制执行 PMP 规则，并且能够在之后修改、合并和/或删除这些规则。在 M 模式下强制执行的规则需要被锁定，这是为了防止编写不当或恶意的 M 模式软件随时删除这些规则。因此，开发人员唯一能做的就是不断向链中添加 PMP 规则，并依赖规则优先级机制来确保系统的稳定性和安全性。这是一种对 PMP 规则的浪费，而且由于它仅在启动过程中需要，mseccfg.RLB 是一个简单的临时解决方案，可以在启动过程中使用，然后禁用并锁定。

请注意，如果在启动过程结束后 RLB 仍然保持设置状态，则会引入安全漏洞，因此即使临时使用也应谨慎使用。 在 M 模式下具有可编辑的 PMP 规则会给人一种虚假的安全感，因为只需几条恶意指令就可以通过这种方式解除任何 PMP 限制。拥有安全控制措施却不加以保护是没有意义的。规则锁定绕过（RLB）仅用于优化 PMP 规则的分配、在调试期间捕获错误，并允许 BootROM/固件注册可执行的共享区域规则。如果开发人员/供应商不需要此类功能，他们应永远不要设置 mseccfg.RLB，如果可能的话，应将其硬连线为 0。无论如何，RLB 应尽快禁用并锁定。

请注意，如果在启动过程结束后 RLB 仍然保持设置状态，则会引入安全漏洞，因此即使临时使用也应谨慎使用。 在 M 模式下具有可编辑的 PMP 规则会给人一种虚假的安全感，因为只需几条恶意指令就可以通过这种方式解除任何 PMP 限制。拥有安全控制措施却不加以保护是没有意义的。规则锁定绕过（RLB）仅用于优化 PMP 规则的分配、在调试期间捕获错误，并允许 BootROM/固件注册可执行的共享区域规则。如果开发人员/供应商不需要此类功能，他们应永远不要设置 mseccfg.RLB，如果可能的话，应将其硬连线为 0。无论如何，RLB 应尽快禁用并锁定。

如果未使用 mseccfg.RLB 并保持未设置状态，则在配置了带有 pmpcfg.L 位设置的 PMP 规则/条目时，它将被立即锁定。

由于优先级较高的 PMP 规则会覆盖优先级较低的规则，锁定的规则必须位于非锁定规则之前。

在当前规范中，除非受到带有 pmpcfg.L 位设置的 PMP 规则的限制，否则 M 模式可以访问任何内存区域。在某些情况下，这种方法过于宽松，尽管可以在启动过程中通过添加 PMP 规则来限制 M 模式，但这也可以被视为对 PMP 规则的浪费。默认情况下阻止任何访问，并将 PMP 用作 M 模式的白名单被认为是一种更安全的方法。此功能可以在启动过程中或 PMP 重置时使用初始寄存器设置来使用。

当前 pmpcfg.L 位的双重含义（标记规则为锁定并在所有模式下强制执行）既不灵活也不清晰。随着机器模式锁定的引入，pmpcfg.L 位区分了仅在 M 模式下强制执行的规则（仅 M 模式）或仅在 S/U 模式下强制执行的规则（仅 S/U 模式）。规则锁定成为仅 M 模式规则定义的一部分，因为在 M 模式下添加规则时，如果未锁定，则可以通过几条指令修改或删除该规则。另一方面，S/U 模式无论如何都无法修改 PMP 规则，因此锁定它们没有意义。

仅 M 模式和仅 S/U 模式规则之间的分离还允许区分哪些区域将由机器模式进程使用（pmpcfg.L == 1），哪些区域将由监管者或 U 模式进程使用（pmpcfg.L == 0），就像虚拟内存的 PTE 中的 U 位标记哪些虚拟内存页面将由 U 模式应用程序使用（U=1），哪些将由监管者/操作系统使用（U=0）一样。有了这种区分，我们能够为任何非仅 M 模式的物理内存区域实现 M 模式下的内存访问和执行保护。

成功篡改 S/U 模式使用的内存区域的攻击者，即使成功诱使在 M 模式下运行的进程使用或执行该区域，也无法成功发起攻击，因为该区域将是仅 S/U 模式，因此在 M 模式下的任何访问都会触发访问异常。

为了支持M模式和 S/U 模式之间的0拷贝传输，需要允许共享内存区域，或者引入类似于 sstatus.SUM 位的机制，以临时允许高特权模式（在本例中为M模式）能够对较低特权进程（在本例中为 S/U 模式）的区域执行加载和存储操作。在案例中，经过小组讨论后，似乎更好的方法是采用第一种方法，并在每个规则的基础上编码此功能，以避免在退出M模式时留下临时的全局绕过激活状态，从而使内存访问保护失效。

尽管可以在 M 模式下使用 mstatus.MPRV 通过通用寄存器复制来读取/写入仅 S/U 模式区域中的数据，但这将以 S/U 模式权限进行，并遵守 S 模式设置的任何 MMU 限制。当然，M 模式仍然可以篡改页表和/或添加仅 S/U 模式规则，从而绕过 S 模式设置的保护措施，但如果攻击者已经成功以这种方式破坏了 M 模式，则无法提供任何安全保证。另请注意，在此提出的威胁模型假设 M 模式中的软件存在漏洞，而不是被攻破的软件。我们曾考虑禁用 mstatus.MPRV，但这似乎过于激进且超出了范围。

共享区域规则既可用于 0 拷贝数据传输，也可用于共享代码段。后者可以用于例如允许 S/U 模式执行供应商提供的代码，这些代码利用了某些供应商特定的 ISA 扩展，而无需通过固件进行 ecall。这类似于 Linux 上采用的 vDSO 方法，允许用户空间代码执行内核代码而无需进行系统调用。

为了确保共享数据区域不可执行，共享代码区域不可修改，编码更改了 pmpcfg.X 位的含义。对于共享数据区域，除了 pmpcfg.LRWX=1111 编码外，pmpcfg.X 位标记了 S/U 模式对该区域的写入能力，因此无法编码可执行的共享数据区域。对于共享代码区域，pmpcfg.X 位标记了 M 模式对该区域的读取能力，并且由于 pmpcfg.RW=01 用于编码共享区域，因此无法编码可写的共享代码区域。

为了添加具有可执行权限的共享区域规则以在M模式和 S/U 模式之间共享代码段，需要实现 mseccfg.RLB，否则此类规则只能在PMP 重置时与 mseccfg.MML 一起设置。这是因为用于共享区域规则的保留编码 pmpcfg.RW=01 仅在设置了 mseccfg.MML 时定义，并且 4b 阻止在设置了

mseccfg.MML 后在M模式下添加具有可执行权限的规则，除非同时设置了 mseccfg.RLB。

使用 pmpcfg.LRWX=1111 编码来锁定共享只读数据区域是后来决定的，其最初的含义是仅M模式的读/写/执行区域。做出这一更改的原因是，已经定义的共享数据区域未被锁定，因此无法限制M模式的读/写访问。同样，我们为两种模式提供了仅执行的共享代码区域，因此决定也能够为两种模式提供最低特权的共享数据区域。这种方法允许例如使用共享代码区域共享 ELF 的 .text 部分，并使用锁定的共享数据区域共享 .rodata 部分，而不允许M模式修改 .rodata。我们还认为，在M模式下拥有锁定的读/写/执行区域没有多大意义，而且可能很危险，因为M模式将无法在那里添加进一步的限制（就像 S/U 模式中S模式可以通过 MMU 进一步限制对 pmpcfg.LWRX=0111 区域的访问一样），从而为在M模式下修改可执行区域留下了可能性。

最初，我们使用 pmpcfg 上的两个保留位之一（第 5 位）来编码共享区域规则，但为了避免分配额外的位（因为这些位是非常有限的资源），决定使用保留的 R=0,W=1 组合。

此限制的目的是，在固件或运行在 M 模式下的操作系统初始化并设置了 mseccfg.MML 后，预计不会添加新的代码区域,因为预计不会有其他内容在 M 模式下运行(其他所有内容将在 S/U 模式下运行)。由于我们希望尽可能限制系统的攻击面，因此禁止在 M 模式下添加/执行可能包含恶意代码的任何新代码区域是有意义的。

如果未设置 mseccfg.MMWP，M 模式仍然可以访问和执行任何未被 PMP 规则覆盖的区域。由于我们试图防止 M 模式执行恶意代码，并且攻击者可能成功将代码放置在未被 PMP 覆盖的某些区域（例如可直接寻址的闪存），我们需要确保 M 模式只能执行在固件/操作系统初始化期间初始化的代码段。

我们仅在设置了 mseccfg.MML 时使用 pmpcfg.RW=01 编码，如果未设置 mseccfg.MML，该编码仍可供未来使用。

7 “Smcntrpmf“ 周期与指令计数特权模式过滤，V1.0.0

7.1 介绍

循环计数器（cycle）和指令计数器（instret）用于支持用户模式下的自剖析（self-profiling）用途，用户可通过两次读取计数器并计算差值来评估用户态软件的性能与行为。当前，这些计数器未按特权模式进行过滤，因此在处理更高特权级代码的异常（如缺页中断或外部中断）时，计数器仍会持续递增。这将导致两个问题：

1.为用户观察到的计数器数值引入不可预测的噪声

2.向用户模式泄露特权级软件的运行信息

本提案通过为循环计数器与指令计数器增加特权模式过滤机制来解决上述问题。

7.2 CSRs

mcyclecfg 与 minstretcfg 是两个 64 位寄存器，分别用于配置循环计数器（cycle）和指令计数器（instret）的特权模式过滤功能。

当所有 xINH 位均为零时，所有模式下的事件计数功能均被启用。

对于 61:58 位的每一位，若关联的特权模式未实现，则该位为只读零值。57:56 位保留供未来可能的模式使用。

在 RV32 架构中:mcyclecfg 的 63:32 位可通过 mcyclecfgh CSR 访问 minstretcfg 的 63:32 位可通过 minstretcfgh CSR 访问

CSR 地址分配如下：

mcyclecfg：0x321

minstretcfg：0x322

mcyclecfgh：0x721

minstretcfgh：0x722

若实现了 Smcdeleg/Ssccfg 扩展，这些寄存器的内容可能从监督者（Supervisor）级别访问。

更自然的 CSR（控制和状态寄存器）地址分配方案本应将 0x320 分配给 mcyclecfg，但该地址已被 mcountinhibit 寄存器占用。

该寄存器格式与 Sscofpmf 扩展定义的可编程计数器规范一致。其中溢出标志位（OF，位63）为只读0值，因为此类计数器在溢出时不会产生本地计数器溢出中断。

7.3 计数器行为

循环计数器（cycle）和指令计数器（instret）的基础行为已被修改为：当 CPU 在执行被抑制特权模式时，将暂停计数。此外，以下规则明确定义了非抑制特权模式与被抑制特权模式之间转换时的计数行为：

当 CPU 处于非抑制特权模式时，循环计数器将正常计数 CPU 时钟周期。

模式转换操作（包括陷入和陷出）可能需要多个时钟周期完成，且特权模式变更的实际生效时刻可能被记录在这些周期中的任意一个（每次陷入或陷出操作的具体生效周期可能不同）。

RISC-V指令集架构并未要求所有陷入(trap)或陷出(trap return)操作所需的时钟周期数必须相同。具体实现可以自主决定：该周期数是否（以及在何种程度上）保持一致性，该周期数是否具备可预测性特权模式切换的具体生效周期同样遵循此自由实现原则。

对于指令计数器(instret)而言，大多数指令不会影响模式转换，因此其行为是明确的：在非抑制模式下完成的指令会使 instret 递增，在抑制模式下完成的指令则不会。但有两类特殊指令会影响特权模式切换：引发同步异常（跳转到更高特权模式）的指令，不被视为完成执行(retire)，因此不会增加 instret 计数。xRET 类指令（返回到较低特权模式）：被视为完成执行，仅当原始特权模式未被抑制时才应增加 instret 计数。

上述关于 instret（指令计数器）的定义旨在确保计数器以可预测的方式递增。举例说明：假设 minstretcfg 寄存器配置为仅用户模式(U-mode)未被抑制，其他所有模式均被抑制。用户模式加载指令：

即使触发缺页异常或其他异常，最终也只会使 instret 递增一次异常处理流程：

a) 引发异常的加载指令执行不会递增（未完成执行）

b) 异常处理程序指令不会递增（在抑制模式执行）

c) 包括 xRET 指令（虽在非抑制模式完成，但源自抑制模式）

8 “Smrnmi“可恢复不可屏蔽中断扩展，V0.5

基础机器级架构仅支持不可恢复的不可屏蔽中断（UNMI），其中 NMI 跳转到机器模式的处理程序，覆盖当前的 mepc 和 mcause 寄存器值。如果硬件线程在陷阱处理程序中执行机器模式代码，则 mepc 和 mcause 中的先前值将无法恢复，因此通常无法恢复执行。

Smrnmi 扩展为 RISC-V 添加了对可恢复不可屏蔽中断（RNMI）的支持。该扩展添加了四个新的 CSR

（mnepc、mncause、mnstatus 和 mnscratch）来保存中断状态，并添加了一条新指令 MNRET，用于从 RNMI 处理程序恢复。

8.1 RNMI 中断信号

rnmi 中断信号是硬件线程的输入。这些中断的优先级高于硬件线程上的任何其他中断或异常，并且不能被软件禁用。具体来说，它们不会通过清除 mstatus.MIE 寄存器来禁用。

8.2 RNMI 处理程序地址

RNMI 中断陷阱处理程序地址由实现定义。RNMI 还具有相关的异常陷阱处理程序地址，该地址由实现定义。

例如，某些实现可能使用mtvec中指定的地址作为RNMI异常陷阱处理程序。

8.3 RNMI CSR

该提案添加了额外的 M 模式 CSR，以启用可恢复的不可屏蔽中断（RNMI）。

图8.1 可恢复非屏蔽中断临时寄存器 mnscratch

mnscratch CSR 保存一个 MXLEN 位的读写寄存器，使 NMI 陷阱处理程序能够保存和恢复被中断的上下文。mnepc CSR 是一个 MXLEN 位的读写寄存器，在进入 NMI 陷阱处理程序时，它保存了发生中断的指令的 PC。

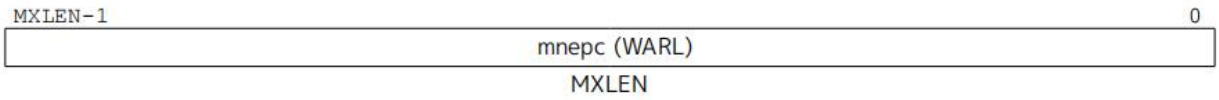


图8.2 可恢复NMI程序计数器mnepc

mnepc 的低位（mnepc[0]）始终为 0。在仅支持 IALIGN=32 的实现中，两个低位（mnepc[1:0]）始终为 0。

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 mnepc[1] 在读取时被屏蔽，使其显示为 0。这种屏蔽也发生在 MRET 指令的隐式读取中。尽管被屏蔽，当 IALIGN=32 时，mnepc[1]仍然可写。

mnepc 是一个 WARL 寄存器，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 mnepc 之前，实现可能会将无效地址转换为 mnepc 能够保存的其他无效地址。

mncause CSR 保存了 NMI 的原因。

如果原因是中断，则 MXLEN-1 位设置为 1，且 NMI 原因编码在最低有效位中。

如果原因是中断且不支持 NMI 原因，则 MXLEN-1 位设置为 1，且最低有效位写入 0。

如果原因是 M 模式中的异常，并且根据 Smdbltrp 扩展导致双重陷阱，则 MXLEN-1 位设置为 0，且最低有效位设置为与引发双重陷阱的异常对应的原因代码。

mnstatus CSR 包含一个两位的 MNPP 字段，该字段在进入 RNMI 陷阱处理程序时，用于保存被中断上下文的特权模式，其编码方式与 mstatus.MPP 相同。此外，mnstatus CSR 还包含一个一位的 MNPV 字段，该字段在进入 RNMI 陷阱处理程序时，用于保存被中断上下文的虚拟化模式，其编码方式与 mstatus.MPV 相同。

如果实现了 Zicfilp 扩展，那么 mnstatus 中还会包含 MNPELP 字段。在进入 RNMI 陷阱处理程序时，MNPELP 用于保存之前的 ELP 状态，此时 MNPELP 会被设置为 ELP 的值，同时 ELP 被设置为 0。此外，mnstatus 还包含 NMIE 位。当 NMIE=1 时，不可屏蔽中断被启用；而当 NMIE=0 时，则会禁用所有中断。

当 NMIE=0 时，硬件线程 的行为就像 mstatus.MPRV 被清除一样，无论 mstatus.MPRV 的当前设置如何。复位时，NMIE 的值为 0。

RNMI（可恢复不可屏蔽中断）在复位时被屏蔽，以便软件有机会初始化数据结构和设备，为后续的RNMI处理做好准备。

软件可以将 NMIE 设置为 1，但尝试清除 NMIE 的操作无效。

通常，只有复位序列会显式设置NMIE位。

NMIE 位可设置并不足以支持 RNMI 的嵌套。为了直接支持此功能，需要允许软件清除 NMIE 位——这一设计选择将违背不可屏蔽性的概念。

希望最小化下一个 RNMI 被捕获的延迟的软件可以采用“上半部/下半部“模型，其中 RNMI 处理程序本身仅将任务加入任务队列然后返回。大部分中断服务工作稍后执行，此时 RNMI 已启用。

对于 WFI 指令，NMIE 是一个全局中断使能，意味着 NMIE 的设置不会影响 WFI 指令的操作。

mnstatus 中的其他位是保留的；软件应写入 0，硬件实现应返回 0。

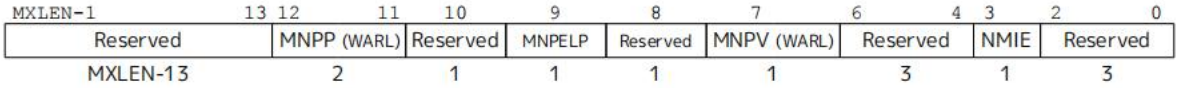


图8.3 可恢复NMI状态寄存器mnstatus

8.4 MNRET 指令

MNRET 是一条仅限 M 模式的指令，它使用 mnepc 和 mnstatus 中的值返回到被中断上下文的程序计数器、特权模式和虚拟化模式。该指令还会设置 mnstatus.NMIE。如果 MNRET 将特权模式更改为比 M 模式更低的特权模式，它还会将 mstatus.MPRV 设置为 0。

如果实现了 Zicfilp 扩展，则如果新的特权模式为 y，MNRET 将 ELP 设置为 yLPE（参见 FCFIACT）和 mnstatus.MNPELP 的逻辑与。

8.5 RNMI 操作

当检测到 RNMI 中断时，被中断的 PC 被写入 mnepc CSR，RNMI 类型被写入 mncause CSR，被中断上下文的特权模式被写入 mnstatus CSR。mnstatus.NMIE 位被清除，屏蔽所有中断。然后硬件线程进入机器模式并跳转到 RNMI 陷阱处理程序地址。RNMI 处理程序可以使用新的 MNRET 指令恢复原始执行，该指

令从 mnepc 恢复 PC，从 mnstatus 恢复特权模式，并设置 mnstatus.NMIE，重新启用中断。如果硬件线程在 mnstatus.NMIE 位清除的情况下在 M 模式中执行时遇到异常，则采取的操作与 mnstatus.NMIE 位设置时发生异常的情况相同，只是程序计数器被设置为 RNMI 异常陷阱处理程序地址。

Smrnmi扩展不会改变MRET和SRET指令的行为。具体而言，MRET和SRET不受mnstatus.NMIE位的影响，它们的执行也不会改变mnstatus.NMIE位。

9 “Smcdeleg“计数器委托扩展，V1.0.0

在现代“富操作系统“环境中，硬件性能监控资源由内核、内核驱动程序和/或虚拟机监视器管理。计数器可以配置为不同的范围，在某些情况下统计系统范围内的事件，而在其他情况下则代表单个虚拟机或应用程序统计事件。在此类环境中，由于以下频繁的计数器写入操作，计数器写入的延迟会直接影响整体性能分析的开销：

样本收集：清除溢出指示并重新加载溢出的计数器。

上下文切换：在进程、线程、容器或虚拟机之间切换。

该扩展提供了一种方法，使 M 模式允许从 S/HS 模式写入选定的计数器和事件选择器。其目的是避免在这些性能关键的管理程序/虚拟机监视器代码段中增加进出 M 模式的转换延迟。该扩展还定义了一个新的 CSR，即 scountinhibit。

对于机器级环境，扩展 Smcdeleg（“Sm“表示特权架构和机器级扩展，“cdeleg“表示计数器委托）涵盖了所有添加的 CSR 以及硬件线程在所有特权级别上的所有行为修改。对于监管者级环境，扩展 Ssccfg（“Ss“表示特权架构和监管者级扩展，“ccfg“表示计数器配置）提供了对委托计数器的访问权限以及对新的监管者级状态的访问权限。

9.1 计数器委托

mcounteren 寄存器允许 M 模式为下一个较低特权模式提供对选定计数器的读取访问权限。当启用 Smcdeleg/Ssccfg 扩展（menvcfg.CDE=1）时，它还允许 M 模式将选定的计数器委托给 S 模式。

siselect（和 vsiselect）索引范围 0x40-0x5F 保留用于委托计数器访问。当计数器 i 被委托（mcounteren[i]=1 且 menvcfg.CDE=1）时，与计数器 i 相关的寄存器状态可以通过 sireg*读取或写入，而 siselect 则保存 0x40+i。通过别名 CSR 可访问的计数器状态如下表所示。

表 9.1 间接 HPM 状态映射

siselect value	sireg	sireg4	sireg2	sireg5
0x40	cycle	cycleh	cyclecfg	cyclecfgh
0x41	见下文			
0x42	instret1	instreth	instretcfg	instretcfgh
0x43	hpmcounter3	hpmcounter3h	hpmevent3	hpmevent3h
0x5F	hpmcounter31	hpmcounter31h	hpmevent31	hpmevent31h

hpmeventi 表示由 mhpmeventi 寄存器访问的状态的子集。同样，cyclecfg 和 instretcfg 分别表示由 mcyclecfg 和 minstretcfg 寄存器访问的状态的子集。有关子集的详细信息，请参见下文。

如果实现了 Smstateen 扩展，请参阅扩展 Smcsrind/Sscsrind（间接 CSR）了解如何通过将 CSR mstateen0 的第 60 位设置为 0 来阻止从低于 M 模式的特权模式访问寄存器 siselect、sireg*、vsiselect 和 vsireg*，以及如何通过将 hstateen0 的第 60 位设置为 0 来阻止从 VS 模式访问 siselect 和 sireg*（实际上是 vsiselect 和 vsireg*）。

本节中的其余规则仅适用于当 CSR 的访问未被 mstateen0[60] = 0 或 hstateen0[60] = 0 阻止时。

当特权模式为 M 或 S 且 siselect 保存的值在 0x40-0x5F 范围内时，以下情况会引发非法指令异常：

* attempts to access any sireg* when menvcfg.CDE = 0;

* attempts to access sireg3 or sireg6;

* attempts to access sireg4 or sireg5 when XLEN = 64;

* attempts to access sireg* when siselect = 0x41, or when the counter

当 menvcfg.CDE = 0 时，尝试访问任何 sireg*；

尝试访问 sireg3 或 sireg6；

当 XLEN = 64 时，尝试访问 sireg4 或 sireg5；

当 siselect = 0x41 或 siselect 选择的计数器未委托给 S 模式（mcounteren 中的相应位 = 0）时，尝试访问 sireg*。

内存映射的 mtime 寄存器不是由监管者软件管理的性能监控计数器，因此上述对 siselect 值 0x41 的特殊处理。

对于 间接 HPM 状态映射 中定义的每个 siselect 和 sireg* 组合，该表进一步指示了底层计数器状态所依赖的扩展。如果底层状态所依赖的任何扩展未实现，则从 M 或 S 模式尝试通过 sireg* 访问给定状态会引发非法指令异常。

如果还实现了虚拟机监视器（H）扩展，则根据扩展 Smcsrind/Sscsrind 的规定，从 VS 模式或 VU 模式尝试直接访问 vsiselect 或 vsireg*，或从 VU 模式尝试访问 siselect 或 sireg*，会引发虚拟指令异常。此外，当 vsiselect 保存的值在 0x40-0x5F 范围内时：

从 M 或 S 模式尝试访问任何 vsireg* 会引发非法指令异常。

从 VS 模式尝试访问任何 sireg*（实际上是 vsireg*）时，如果 menvcfg.CDE = 0，会引发非法指令异常；如果 menvcfg.CDE = 1，会引发虚拟指令异常。

如果实现了 Sscofpmf,sireg2 和 sireg5 仅提供对事件选择器寄存器子集的访问权限。具体来说，事件选择器位 62（MINH）在通过 sireg* 访问时为只读 0。同样，如果实现了 Smcntrpmf，sireg2 和 sireg5 仅提供对计数器配置寄存器子集的访问权限。计数器配置寄存器位 62(MINH)在通过 sireg* 访问时为只读 0。

9.2 监管者计数器禁止（scountinhibit）寄存器

Smcdeleg/Ssccfg 定义了一个新的 scountinhibit 寄存器，它是 mcountinhibit 的掩码别名。对

于委托给 S 模式的计数器，可以通过 scountinhibit 访问相关的 mcountinhibit 位。对于未委托给 S 模式的计数器，scountinhibit 中的相关位为只读 0。

当 menvcfg.CDE=0 时，尝试访问 scountinhibit 会引发非法指令异常。当启用监管者计数器委托扩展时，从 VS 模式或 VU 模式尝试访问 scountinhibit 会引发虚拟指令异常。

9.3 虚拟化 scountovf

对于支持 Smcdeleg/Ssccfg、Sscofpmf 和 H 扩展的实现，当 menvcfg.CDE=1 时，从 VS 模式或 VU 模式尝试读取 scountovf 会引发虚拟指令异常。

9.4 虚拟化本地计数器溢出中断

对于支持 Smcdeleg、Sscofpmf 和 Smaia 的实现，CSR mvip 和 mvien 中的本地计数器溢出中断（LCOFI）位（第 13 位）已实现并可写。

对于支持 Smcdeleg/Ssccfg、Sscofpmf、Smaia/Ssaia 和 H 扩展的实现，hvip 和 hvien 中的 LCOFI 位（第 13 位）已实现并可写。

hvip 寄存器由虚拟机监视器（H）扩展定义，而 mvien 和 hvien 寄存器由 Smaia/Ssaia 扩展定义。

由于实现了 hvip.LCOFI，隐含地，vsie 和 vsip 中的 LCOFI 位（第 13 位）也已实现。要求支持上述列出的 LCOFI 位，确保虚拟 LCOFI 可以传递给运行在 S 模式的操作系统以及运行在 VS 模式的客户操作系统。是否实现并使 mideleg 和 hideleg 中的 LCOFI 位（第 13 位）可写是可选的，这些位允许将所有 LCOFI 分别委托给 S 模式和 VS 模式。

10 监管级 ISA，V1.13

本章介绍 RISC-V 监管级架构，它包含一个通用内核，可与各种监管级地址转换和保护方案配合使用。

S模式在与底层物理硬件交互方面受到有意限制，目的是支持简洁的虚拟化。例如，某些监管级设施（如定时器和处理器间中断的请求）由特定实现机制提供。在一些系统中，监管程序执行环境（SEE）通过监管程序二进制接口（SBI）以指定方式提供这些设施；而在其他系统中，则通过其他实现机制直接提供这些设施。

10.1 CSR 监管

为监管提供了许多 CSR。

监管者只能查看监管级操作系统应能看到的 CSR 状态。具体而言，在监管可访问的 CSR 中，不存在（或不存在）更高权限级别（机器级别或其他级别）的信息。

许多监管员 CSR 是等效的机器模式 CSR 的子集，因此应首先阅读机器模式章节，以帮助理解监管员级 CSR 的描述。

10.1.1 监管者状态寄存器（sstatus）

sstatus 寄存器是一个 SXLEN 位读/写寄存器，格式如图所示：当 SXLEN=32 时为监控模式状态（sstatus）寄存器，当 SXLEN=64 时为监控模式状态（sstatus）寄存器。sstatus 寄存器记录处理器当前的运行状态。

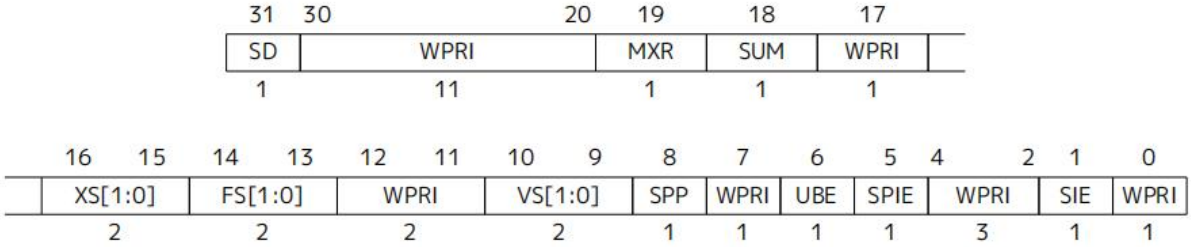


图10.1 当 SXLEN=32 时的监管者模式状态寄存器

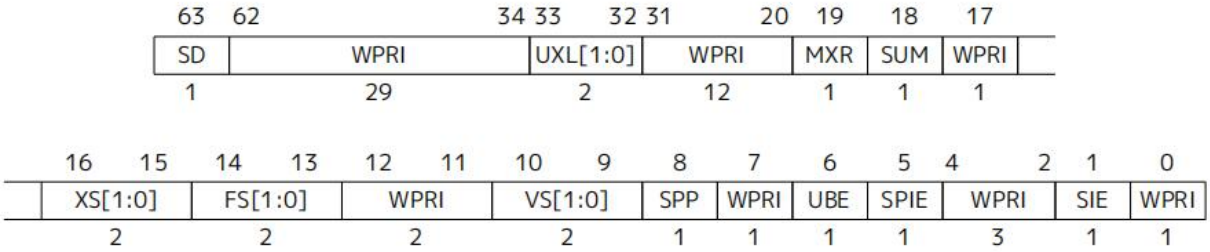


图10.2 当 SXLEN=64 时，监管者模式状态寄存器

SPP 位指示进入监控模式前执行陷阱的权限级别。捕获陷阱时，如果陷阱来自 U 模式，则 SPP 设为 0，否则设为 1。当执行 SRET 指令（参见 otherpriv）从陷阱处理程序返回时，如果 SPP 位为 0，则特权级别被设置为 U 模式；如果 SPP 位为 1，则特权级别被设置为监管模式；然后 SPP 被设置为 0。

SIE 位可启用或禁用监管模式下的所有中断。当 SIE 位清 0 时，中断不会在监管模式下发生。当 硬件线程 以 U 模式运行时，SIE 中的值将被忽略，并启用监管级中断。监管员可使用 sie CSR 禁用单个中断源。

SPIE 位指示在陷阱进入监管模式之前是否启用了监管中断。当陷阱进入监管模式时，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。

sstatus 寄存器是 mstatus 寄存器的子集。

在简单的实现中，读取或写入 sstatus 中的任何字段都等同于读取或写入 mstatus 中的同名字段。

10.1.1.1 sstatus 寄存器中的 ISA 基本控制

UXL 字段控制 U 模式的 XLEN 值（称为 UXLEN），它可能不同于 S 模式的 XLEN 值（称为 SXLEN）。UXL 的编码与 [misabase] 中 misa 的 MXL 字段相同。

当 SXLEN=32 时，UXL 字段不存在，UXLEN=32。当 SXLEN=64 时，它是一个 WARL 字段，对 UXLEN 的当前值进行编码。具体而言，实现时可以将 UXL 设置为只读字段，其值始终确保 UXLEN=SXLEN。

如果 UXLEN≠SXLEN，那么在窄模式下执行的指令必须忽略配置 XLEN 以上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 XLEN。

如果 UXLEN <SXLEN、U 模式指令读取地址以及加载和存储有效地址取模数 2UXLEN .例如，当 UXLEN=32 和 SXLEN=64 时，U 模式内存访问会引用地址空间最低的 4 GiB。

某些 HINT 指令被编码为整数计算指令，用其当前值覆盖目标寄存器，例如 c.addi x8, 0。 当执

行此类 HINT 指令时，XLEN < SXLEN，且目标寄存器的 SXLEN...XLEN 位不全部等于 XLEN-1 位，目标寄存器的 SXLEN...XLEN 位是保持不变，还是用 XLEN-1 位的副本覆盖，由执行情况决定。

该定义允许执行程序省略某些 HINT 的寄存器回写，同时允许它们以与其他整数计算指令相同的方式执行其他 HINT。只有S模式（SXLEN > UXLEN）才能观察到这种执行选择；U 模式则无法观察到。

10.1.1.2 sstatus 寄存器中的内存权限

MXR（Make eXecutable Readable）位可修改加载访问虚拟内存的权限。当 MXR=0 时，只有从标记为可读（Sv32 页表项中的 R=1）的页面加载才会成功。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 无效。

SUM（允许超级用户内存访问）位可修改 S 模式加载和存储访问虚拟内存的权限。当 SUM=0 时，S 模式内存访问 U 模式可访问的页面（Sv32 页面表项中的 U=1）将出现故障。当 SUM=1 时，允许这些访问。当不使用基于页面的虚拟内存时，或在 U 模式下执行时，SUM 不会产生任何影响。请注意，无论 SUM 的状态如何，S 模式都不能执行来自用户页的指令。如果 satp.MODE 为只读 0，则 SUM 为只读 0。

SUM 机制可防止监控软件无意中访问用户内存。操作系统可以在清除 SUM 的情况下执行大部分代码；少数应该访问用户内存的代码段可以临时设置 SUM。

SUM 机制不允许S模式软件执行用户代码页中的指令。在监管程序上下文中从用户内存执行指令的合法用例一般很少，在 POSIX 环境中更是不存在。但是，如果监管程序的漏洞代码可以存储在用户缓冲区中，并位于攻击者选择的虚拟地址上，那么监管程序中导致任意代码执行的漏洞就更容易被利用了。

一些非 POSIX 单地址空间操作系统确实允许某些特权软件在监管模式下部分执行，而大部分程序则在U模式下运行，所有程序都在共享地址空间中运行。这种情况可以通过在多个虚拟地址上映射具有不同权限的物理代码页来实现，也可以在指令页故障处理程序的协助下，指导监管软件使用备用映射。

10.1.1.3 sstatus 寄存器中的尾数控制

UBE 位是一个 WARL 字段，用于控制 U 模式下显式内存访问的字节序，可能与 S 模式下内存访问的字节序不同。实现过程中，可以将 UBE 设置为只读字段，并始终指定与 S 模式相同的结束符。

UBE 控制从 U 模式进行的显式加载和存储是小字节（UBE=0）还是大字节（UBE=1）。它不影响指令取回，因为指令取回是隐式内存访问，始终小字节方向。

对于监管员级内存管理数据结构（如页表）的隐式访问，S 模式字节序始终适用，UBE 被忽略。

标准的 RISC-V ABI 预计是纯粹的小前端（little-endian-only）或纯粹的大前端（big-endian-only），不允许混合前端（endianness）。尽管如此，还是对端位控制进行了定义，以允许一种端位的操作系统执行相反端位的U模式程序。

10.1.2 监管员陷阱向量基地址（stvec）寄存器

stvec 寄存器是一个 SXLEN 位读/写寄存器，用于保存陷阱矢量配置，包括矢量基地址 (BASE) 和

矢量模式 (MODE)。监视器陷阱向量基地址（stvec）寄存器。stvec 中的 BASE 字段可以容纳任何有效的虚拟地址或物理地址，但必须遵守以下对齐限制：地址必须是 4 字节对齐，除直接模式外的其他模式设置可能会对 BASE 字段中的值施加额外的对齐限制。

MODE 字段的编码请参阅 Encoding of stvec MODE field（stvec MODE 字段编码）。当 MODE=Direct 时，所有进入上位机模式的陷阱都会导致 pc 被设置为 BASE 字段中的地址。当 MODE=Vectored 时，进入上位机模式的所有同步异常都会导致 pc 被设置为 BASE 字段中的地址，而中断则会导致 pc 被设置为 BASE 字段中的地址加上中断原因编号的四倍。例如，监管模式下的定时器中断（请参阅陷阱后的监管原因（scause）寄存器值）的 pc 设置为 BASE 字段中的地址加上中断原因编号的四倍。同步异常优先级由 exception-priority 给出）会导致 pc 被设置为 BASE+0x14。设置 MODE=Vectored 可能会对 BASE 施加更严格的对齐限制。

10.1.3 监管员中断（sip 和 sie）寄存器

sip 寄存器是一个 SXLEN 位读/写寄存器，包含待处理中断信息，而 sie 寄存器是一个相应的 SXLEN 位读/写寄存器，包含中断使能位。中断原因编号 i（如 CSR scause，上位机原因（scause）寄存器中的报告）与 sip 和 sie 中的位 i 相对应。第 15:0 位仅分配给标准中断原因，而第 16 位及以上指定用于平台。

如果以下两种情况均为真：(a) 当前权限模式为 S 且 sstatus 寄存器中的 SIE 位被置位，或当前权限模式的权限小于 S 模式；(b) sip 和 sie 中的 i 位均被置位，则中断 i 将陷波到 S 模式。

中断陷阱发生的这些条件必须在 sip 中中断成为或停止挂起的一定时间内进行评估，还必须在执行 SRET 指令或明确写入这些中断陷阱条件明确依赖的 CSR（包括 sip、sie 和 sstatus）后立即进行评估。S 模式的中断优先于低权限模式的中断。

寄存器 sip 中的每个位均可写入或只读。当寄存器 sip 中的第 i 位可写时，可通过向该位写 0 来清除待处理中断 i。如果中断 i 可以成为待处理中断，但寄存器 sip 中的位 i 是只读的，则必须提供其他机制来清除待处理中断（可能需要调用执行环境）。如果相应的中断挂起，sie 中的位必须是可写的。sie 中不可写的位只读为 0。

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

位 sip.SEIP 和 sie.SEIE 是中断等待位和中断启用位，用于监管级外部中断。如果执行，SEIP 在 sip 中为只读，由执行环境设置和清除，通常通过特定平台的中断控制器进行。

位 sip.STIP 和 sie.STIE 是中断等待位和中断启用位，用于监管级定时器中断。如果执行，STIP 在 sip 中为只读，由执行环境设置和清除。位 sip.SSIP 和 sie.SSIE 是中断等待位和中断启用位，用于监控级软件中断。如果执行，SSIP 可在 sip 中写入，也可由特定平台的中断控制器设置为 1。

如果实施了 Sscofpmf 扩展，则 sip.LCOFIP 和 sie.LCOFIE 位是本地计数溢出中断的中断等待位和中断启用位。LCOFIP 在 sip 中为读写器，反映 mhpmeventn.OF 的任何位被设置后发生的本地计数器溢出中断请求。如果未实施 Sscofpmf 扩展，则 sip.LCOFIP 和 sie.LCOFIE 为只读 0。

处理器间中断通过特定的实现方式发送给其他 硬件线程，最终会导致接收方 硬件线程 的 sip 寄存器中的 SSIP 位被设置。

每种标准中断类型（SEI、STI、SSI 或 LCOFI）都可能未执行，在这种情况下，相应的中断等待位和中断启用位都是只读 0。sip 和 sie 中的所有位都是 WARL 字段。向 sie 中的每个位写入 1，然后回读哪些位为 1，即可找到已执行的中断。

sip 和 sie 寄存器是 mip 和 mie 寄存器的子集。读取 sip/sie 寄存器的任何已执行字段或写入任何可写入字段，都会影响对 mip/mie 寄存器同名字段的读取或写入。

sip 和 sie 的第 3、7 和 11 位分别对应机器模式软件、定时器和外部中断。由于大多数平台不会将这些中断从M模式委托给S模式，因此它们在 sip 的标准部分（位 15:0）和 sie 的标准部分（位 15:0）中显示为 0。

以监控模式为目标的多个同时中断按以下优先级递减顺序处理：SEI、SSI、STI、LCOFI。

10.1.4 监控计时器和性能计数器

监控软件使用与 U 模式软件相同的硬件性能监控设施，包括时间、周期和 instret CSR。实施工作应提供修改计数器值的机制。

执行程序必须提供按实时计数器时间安排定时器中断的功能。

10.1.5 计数器使能（scounteren）寄存器

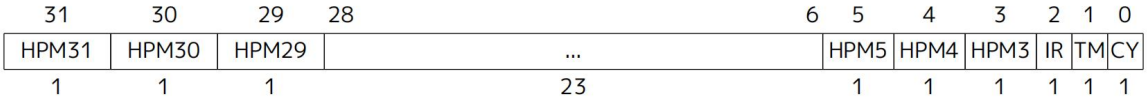


图10.3 计数器使能寄存器（scountere ）

计数器启用（scounterenen）CSR 是一个 32 位寄存器，用于控制硬件性能监控计数器对 U 模式的可用性。

当 scounteren 寄存器中的 CY、TM、IR 或 HPMn 位清 0 时，在 U 模式下执行时试图读取周期、时间、instret 或 hpmcountern 寄存器将导致非法指令异常。当其中一个位被置位时，允许访问相应的寄存器。

必须执行 scounteren。不过，其中任何一位都可能是只读 0 位，表示在 U 模式下执行时，对相应计数器的读取将导致异常。因此，它们实际上是 WARL 字段。

mcounteren 中某位的设置不会影响 scounteren 中相应位是否可写。不过，U 模式只能在 scounteren 和 mcounteren 中的相应位都被设置的情况下才能访问计数器。

10.1.6 刮擦寄存器（sscratch）

图10.4 监管暂存寄存器

sscratch CSR 是一个 SXLEN 位读/写寄存器，专门供监管程序使用。通常情况下，当 硬件线程 执行用户代码时，sscratch 用于保存指向 硬件线程 本地监管程序上下文的指针。在陷阱处理程序开始时，sscratch 与用户寄存器交换，以提供一个初始工作寄存器。

10.1.7 异常程序计数器（sepc）

sepc 是一个 SXLEN 位读/写 CSR,格式如上位机异常程序计数寄存器所示。sepc 的低位(sepc[0])始终为 0。在仅支持 IALIGN=32 的实现中，两个低位（sepc[1:0]）始终为 0。

如果实现允许 IALIGN 为 16 或 32(例如通过改变 CSR misa),那么当 IALIGN=32 时,位 sepc[1]在读取时会被屏蔽，使其显示为 0。 这种屏蔽也发生在 SRET 指令的隐式读取中。在 IALIGN=32 时，sepc[1] 位虽然被屏蔽，但仍然可以写入。

sepc 是一个 WARL 寄存器，必须能够保存所有有效的虚拟地址。它不一定能够保存所有可能的无效地址。在写入 sepc 之前，实现者可以将无效地址转换为 sepc 能够保存的其他无效地址。

当陷阱进入 S 模式时，sepc 会被写入被中断或遇到异常的指令的虚拟地址。否则，虽然软件可能会明确写入 sepc，但执行程序永远不会写入。

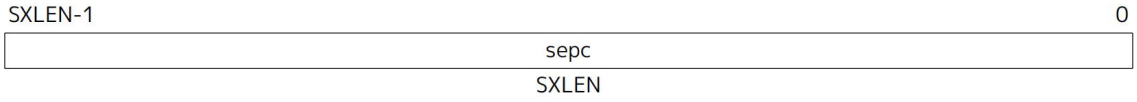


图10.5 监管异常程序计数器寄存器

10.1.8 监管员原因（scause）登记册

scause CSR 是一个 SXLEN 位读写寄存器，格式如监视器原因（scause）寄存器所示。当陷阱进入 S 模式时，scause 会被写入一个代码，表明引起陷阱的事件。否则，尽管软件可能会明确写入，但执行程序永远不会写入 scause。

如果陷阱是由中断引起的，则设置 scause 寄存器中的中断位。异常代码字段包含识别上次异常或中断的代码。陷阱后的上位机原因（scause）寄存器值。同步异常优先级由[exception-priority]给出。异常代码是一个 WLRL 字段。它必须包含 0-31 值（即必须执行第 4-0 位），否则只能保证包含受支持的异常代码。

图10.6 监管原因寄存器（scause）

表 10.1 监管者模式异常原因寄存器 (scause) 在陷入后的值

中断	异常代码	介绍
1	0	保留
1	1	监控软件中断
1	2-4	保留
1	5	监控定时器中断
1	6-8	保留
1	9	主管外部中断
1	10-12	保留
1	13	Counter-overflow 中断
1	14-15	保留
1	≥16	指定作平台用途
0	0	指令地址错位
0	1	指令存取故障

0	2	非法指令
0	3	断点
0	4	负载地址错位
0	5	负载接入故障
0	6	仓库/AMO 地址未对齐
0	7	存储/AMO 访问故障
0	8	u 模式的环境调用
0	9	s 模式的环境调用
0	10-11	保留
0	12	说明页故障
0	13	加载页面故障
0	14	保留
0	15	Store/AMO 页面错误
0	16-17	保留
0	18	软件检查
0	19	硬件错误
0	20-23	保留
0	24-31	指定自定义使用
	32-47	保留
	48-63	指定自定义使用
	≥64	保留

10.1.9 监管员陷阱值（stval）寄存器

stval CSR 是一个 SXLEN 位读写寄存器，格式如监管员陷阱值寄存器所示。当异常进入 S 模式时，特定于异常的信息会被写入 stval 寄存器，从而帮助软件对陷阱进行处理。否则，尽管软件可能会明确写入，但执行程序永远不会写入 stval。硬件平台将指定哪些异常必须设置 stval 信息，哪些可以无条件将其设置为 0，哪些可以表现出两种行为，具体取决于导致异常的基本事件。

如果在取指令、加载或存储过程中出现断点、地址错位、访问故障或页面故障异常时写入非 0 值的 stval，那么 stval 将包含发生故障的虚拟地址。

当错位加载或存储导致访问故障或页面故障异常时，如果写入非 0 值的 stval，那么 stval 将包含导致故障的访问部分的虚拟地址。如果在具有可变长度指令的系统中，当发生指令访问故障或页面故障异常时，stval 被写入一个非 0 值，那么 stval 将包含导致故障的指令部分的虚拟地址，而 sepc 将指向指令的起始部分。

stval 寄存器也可用于返回非法指令异常时的故障指令位（sepc 指向内存中的故障指令）。如果在发生非法指令异常时写入非 0 值的 stval，那么 stval 将包含以下指令中最短的一个：

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

出现非法指令异常时，加载到 stval 中的值将右对齐，所有未使用的高位清 0。

在软件检查异常引起的陷阱中，stval 寄存器保存异常原因。定义了以下编码：

0 - 未提供信息。

2 - 着陆台故障。由 Zicfilp 扩展名（[priv-forward]）定义。

3 - 影子堆栈故障。由 Zicfiss 扩展（[priv-backward]）定义。

对于其他陷阱，stval 设置为 0，但未来的标准可能会重新定义其他陷阱的 stval 设置。

stval 是一个 WARL 寄存器，必须能够保存所有有效的虚拟地址和值 0。它无需保存所有可能的无效地址。在写入 stval 之前，实现者可以将无效地址转换为 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功能，stval 还必须能够保存所有小于 2^N 的值。

其中，N 是 SXLEN 和 ILEN 中较小的一个。

10.1.10 上位机环境配置（senvcfg）寄存器

senvcfg CSR 是一个 SXLEN 位读/写寄存器，格式如 RV64 的监控器环境配置寄存器 (senvcfg) 所示，用于控制 U 模式执行环境的某些特性。

图10.7 监督陷阱值寄存器

如果在 Senvcfg 中将位 FIOM（I/O 意味着内存）设为 1，在 U 模式下执行的 FENCE 指令将被修改，因此设备 I/O 访问顺序的要求也意味着主存访问顺序的要求。详细说明了 FIOM=1 时 U 模式下 FENCE 指令位 PI、PO、SI 和 SO 的修改解释。同样，对于 FIOM=1 时的 U 模式，如果访问设备 I/O 区域的原子指令的 aq 和/或 rl 位被设置，则该指令的排序将被视为同时访问设备 I/O 和内存。如果 satp.MODE 为只读 0（始终为 “裸”），则执行时可将 FIOM 设为只读 0。

位 FIOM 存在于一种特殊情况下，即 I/O 设备被仿真为 U 模式，且以下两种情况均属实：(a) 被仿真设备有一个内存缓冲区，该缓冲区本应是 I/O 空间，但实际上通过地址转换映射到了主内存；(b) 从 U 模式访问该被仿真设备涉及多个物理 硬件线程。

在 S 模式下运行的管理程序，如果不能使用准虚拟化技术，则可能需要为 U 模式仿真一个设备，而不使用[hypervisor]的管理程序扩展。如果同一个管理程序为虚拟机（VM）提供了多个虚拟 硬件线程，并一对一地映射到真实 硬件线程，那么多个 硬件线程 可能会同时访问仿真设备，原因可能是：(a) 虚拟机内的客户操作系统将设备中断处理分配给一个 硬件线程，而该设备也由中断处理程序之外的另一个 硬件线程 访问，或 (b) 设备的控制权（或部分控制权）从一个 硬件线程 转移到另一个 硬件线程，例如用于虚拟机内的中断负载平衡。在这种情况下，虚拟机中的客户软件需要像往常一样，使用互斥锁和/或处理器间中断，在多个核间正确协调对（仿真）设备的访问，这部分需要执行 I/O 栅栏。但如果某些设备的 “I/O “实际上是主内存，而客户机并不知晓，那么这些 I/O 栅栏可能就不够用了。设置 FIOM=1 将修改这些规则（以及在 U 模式下执行的所有其他 I/O 规则），使其也包括主内存。

软件总是可以避免设置 FIOM，方法是不使用主内存来模拟应属于 I/O 空间的设备内存缓冲区。不过，这种选择通常需要捕获对模拟缓冲区的所有 U 模式访问，这可能会对性能产生明显影响。FIOM 提供的替代方案实施成本很低，因此我们认为即使很少启用，也值得支持。

CBZE 字段的定义将由即将推出的 Zicboz 扩展提供。在 Zicboz 扩展获批前，其在 Senvcfg 中的分配可能会有所变化。CBCFE 和 CBIE 字段的定义将由即将推出的 Zicbom 扩展提供，在 Zicbom 扩展获批前，它们在 Senvcfg 中的分配也可能发生变化。

PMM 字段的定义由 Ssnpm 扩展提供。Zicfilp 扩展在 Senvcfg 中增加了 LPE 字段。当 LPE 字段设置为 1 时，Zicfilp 扩展将在 VU/U 模式下启用。当 LPE 字段为 0 时，在 VU/U 模式下不启用 Zicfilp 扩展，以下规则适用于 VU/U 模式：

硬件线程 不会更新 ELP 状态，而是保持 NOLPEXPECTED。

LPAD 指令作为无操作符运行。

Zicfiss 扩展在 Senvcfg 中添加了 SSE 字段。当 SSE 字段设置为 1 时，Zicfiss 扩展将在 VU/U

模式下激活。当 SSE 字段为 0 时，Zicfiss 扩展在 VU/U 模式下保持非激活状态，并适用以下规则：

32 位 Zicfiss 指令将恢复 Zimop 所定义的行为。

16 位 Zicfiss 指令将恢复 Zcmop 所定义的行为。

当 menvcfg.SSE 为 1 时，SSAMOSWAP.W/D 在 U 模式下会引发非法指令异常，在 VU 模式下会引发虚拟指令异常。

10.1.11 上位机地址转换和保护（satp）寄存器

satp CSR 是一个 SXLEN 位读/写寄存器，格式如图所示：当 SXLEN=32 时为监管者地址转换和保护（satp）寄存器，当 SXLEN=32 时为监管者地址转换和保护（satp）寄存器，当 SXLEN=64 时为监管者地址转换和保护（satp）寄存器，当 SXLEN=64 时为模式值 Bare、Sv39、Sv48 和 Sv57。该寄存器保存根页表的物理页码 (PPN)，即监管器物理地址除以 4 KiB；地址空间标识符 (ASID)，用于在每个地址空间基础上进行地址转换；MODE 字段，用于选择当前的地址转换方案。有关访问该寄存器的更多详细信息，请参阅 [virt-control]。

当 SXLEN=32 时，上位机地址转换和保护 (satp) 寄存器。

在 satp 中存储 PPN，而不是物理地址，可支持 RV32 超过 4 GiB 的物理地址空间。

satp.PPN 字段可能无法容纳所有物理页码。某些平台标准可能会对 satp.PPN 的假设值进行限制，例如，要求与主内存相对应的所有物理页码都是可表示的。

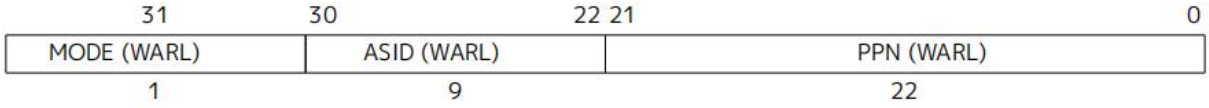


图10.8 监督模式地址转换与保护寄存器 satp（当 SXLEN=32 时）

我们将 ASID 和页表基地址存储在同一个 CSR 中，以便在上下文切换时原子地更改这两个地址对。非原子交换可能会用新的翻译污染旧的虚拟地址空间，反之亦然。这种方法还能略微降低上下文切换的成本。

satp MODE 字段的编码显示了 SXLEN=32 和 SXLEN=64 时 MODE 字段的编码。当 MODE=Bare 时，上位机虚拟地址等于上位机物理地址，除了 [pmp] 中描述的物理内存保护方案外，没有额外的内存保护。要选择 MODE=Bare，软件必须将 satp 的其余字段（当 SXLEN=32 时为第 30-0 位，当 SXLEN=64 时为第 59-0 位）写入 0。如果尝试在其余字段中以非 0 模式选择 MODE=Bare，则会对其余字段的取值产生不明影响，并对地址转换和保护行为产生不明影响。

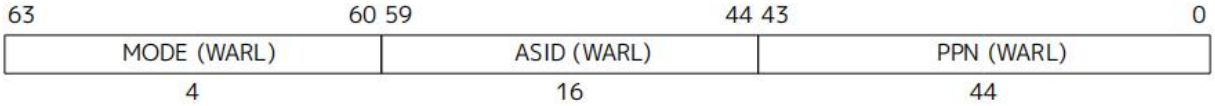


图10.9 监督模式地址转换与保护寄存器 satp（当 SXLEN=64 时，针对 Bare、Sv39、Sv48 和 Sv57 模式）

当 SXLEN=32 时，与 MODE=Bare 和 ASID[8:7]=3 相对应的 satp 编码指定为自定义使用，而与 MODE=Bare 和 ASID[8:7]≠3 相对应的编码则保留为未来标准使用。当 SXLEN=64 时，与 MODE=Bare 对应的所有 satp 编码将保留供未来标准使用。

本标准 1.11 版规定，当 MODE=Bare 时，satp 中的其余字段不起作用。保留这些字段有利于将来定义更多的转换和保护模式，特别是在 RV32 中，现有 MODE 字段的所有模式都已分配。

当 SXLEN=32 时，MODE 的唯一其他有效设置是 Sv32，即 Sv32：基于页面的 32 位虚拟内存系统中描述的分页虚拟内存方案。

当 SXLEN=64 时，定义了三种分页虚拟内存方案：Sv39、Sv48 和 Sv57 分别在 Sv39：基于分页的 39 位虚拟内存系统、Sv48：基于分页的 48 位虚拟内存系统和 Sv57：基于分页的 57 位虚拟内存系统中描述。本规范的后续版本还将定义另外一个方案，即 Sv64。其余的 MODE 设置保留给将来使用，并可能对 satp 中的其他字段定义不同的解释。不要求实现支持所有 MODE 设置，如果以不支持的 MODE 写入 satp，则整个写入操作无效；satp 中的任何字段都不会被修改。

ASID 位数不详，可能为 0。实现的 ASID 位数称为 ASIDLEN，可通过向 ASID 字段中的每个位位置写 1，然后读回 satp 中的值来确定 ASID 字段中哪些位位置为 1。首先执行 ASID 的最小有效位：也就是说，如果 ASIDLEN >0 时，ASID[ASIDLEN-1:0] 可写。ASIDLEN 的最大值称为 ASIDMAX，Sv32 为 9，Sv39、Sv48 和 Sv57 为 16。

对于许多应用而言，页面大小的选择会对性能产生重大影响。较大的页面大小会增加 TLB 的覆盖范围，并放松对虚拟索引、物理标记高速缓存的关联性限制。同时，大页面会加剧内部碎片，浪费物理内存，甚至可能浪费缓存容量

经过反复斟酌，我们最终确定 RV32 和 RV64 的常规页面大小均为 4 KiB。我们希望这一决定能简化底层运行时软件和设备驱动程序的移植。现代操作系统对透明超级页的支持改善了 TLB 的覆盖范围问题。此外，多级 TLB 层次结构相对于其地址空间映射的多级高速缓存层次结构而言成本很低。

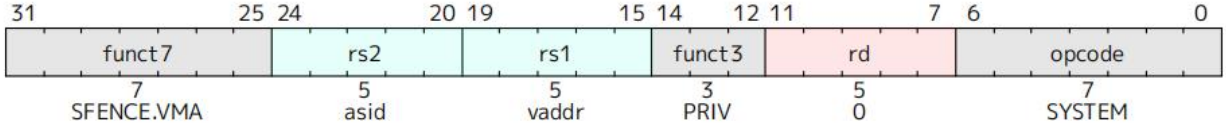
当有效特权模式为S模式或U模式时，satp CSR 被视为激活状态。只有当 satp 处于激活状态时，地址转换算法的执行才能使用给定的 satp 值。当 satp 处于活动状态时开始的地址翻译，除非执行了匹配地址和 ASID 的 SFENCE.VMA 指令，否则无需在 satp 不再活动时完成或终止。必须通过 SFENCE.VMA 指令来确保地址翻译数据结构的更新能够被 硬件线程 在后续隐式读取这些结构时所观察到。

请注意，写入 satp 并不意味着页表更新和后续地址转换之间有任何排序约束，也不意味着地址转换缓存无效。如果新地址空间的页表已被修改，或者 ASID 被重复使用，可能需要在写入 satp 之后或某些情况下在写入 satp 之前执行 SFENCE.VMA 指令（参见 “监管器内存管理栅栏指令”）。

只要有足够大的 ASID 空间，在 satp 写入时不强制实施刷新地址转换缓存，就能降低上下文切换的成本。

10.2 监管指令

除了 otherpriv 中定义的 SRET 指令外，还提供了一条新的监管级指令。



上位机内存管理栅栏指令 SFENCE.VMA 用于使内存管理数据结构的更新与当前执行同步。指令的执行会导致对这些数据结构的隐式读取和写入;不过,这些隐式引用通常不按显式加载和存储的顺序排列。执行 SFENCE.VMA 指令可确保当前 RISC-V 硬件线程 已可见的任何先前存储均在该 硬件线程 中后续指令对内存管理数据结构的某些隐式引用之前排序。如下所述，SFENCE.VMA 排序的特定操作集由 rs1 和 rs2 决定。SFENCE.VMA 还可用于使与高速缓存相关的地址转换缓存中的条目失效（请参阅虚拟地址转换过程）。有关该指令行为的更多详情，请参阅 virt-control 和 pmp-vmem。

SFENCE.VMA 用于刷新与地址转换相关的任何本地硬件缓存。它被指定为栅栏而非 TLB 清除，以便在哪些指令受清除操作影响方面提供更简洁的语义，并支持更广泛的动态缓存结构和内存管理方案。SFENCE.VMA 还被更高权限级别用于同步页表写入和地址转换硬件。

SFENCE.VMA 只命令本地 硬件线程 对内存管理数据结构的隐式引用。

因此，当内存管理数据结构被修改时，必须单独通知其他线程。一种方法是：1）使用本地数据栅栏，确保本地写入在全局可见；2）向其他线程发送处理器间中断；3）在远程线程的中断处理程序中发送本地 SFENCE.VMA；最后 4）向初始线程发送操作完成的信号。当然，这就是 RISC-V 类似于 TLB 崩溃的过程。

对于只针对单个地址映射（即一个页面或超级页面）修改了转换数据结构的常见情况，rs1 可以指定该映射中的一个虚拟地址，以便只针对该映射执行转换栅栏。此外，对于翻译数据结构只针对单个地址空间标识符进行修改的常见情况，rs2 可以指定地址空间。SFENCE.VMA 的行为取决于 rs1 和 rs2，具体如下：

如果 rs1=x0 和 rs2=x0，栅栏会命令对所有地址空间的任何级别页表进行的所有读写操作。栅栏还会使所有地址空间的所有地址转换缓存项失效。

如果 rs1=x0 且 rs2≠x0，则栅栏会对页表任何级别的所有读写进行排序，但仅限于整数寄存器 rs2 标识的地址空间。对全局映射（参见寻址和内存保护）的访问不受限制。除包含全局映射的条目外，栅栏还会使所有与整数寄存器 rs2 标识的地址空间相匹配的地址转换缓存条目失效。

如果 rs1≠x0 且 rs2=x0，则对于所有地址空间，栅栏只命令对与 rs1 中虚拟地址对应的叶子页表项的读写。对于所有地址空间，栅栏还会使包含与 rs1 中虚拟地址对应的叶子页表项的所有地址转换缓存项失效。

如果 rs1≠x0 和 rs2≠x0，则栅栏只对与 rs1 中虚拟地址相对应的叶子页表项的读写进行排序，而对整数寄存器 rs2 所标识的地址空间进行排序。对全局映射的访问不排序。除了包含全局映射的条目外,栅栏还会使所有包含与 rs1 中虚拟地址对应的叶子页表条目且与整数寄存器 rs2 标识的地址空间相匹配的地址转换缓存条目失效。

如果 rs1 中的值不是有效的虚拟地址，则 SFENCE.VMA 指令不起作用。在这种情况下不会出现异常。

过度栅栏始终是合法的，例如，仅根据 rs1 和/或 rs2 中的位子集进行栅栏，和/或简单地将所有 SFENCE.VMA 指令视为 rs1=x0 和/或 rs2=x0。例如，较简单的实现可以忽略 rs1 中的虚拟地址和 rs2 中的 ASID 值，并始终执行全局栅栏。当 rs1 中的虚拟地址无效时，选择不引发异常，有利于这种简化。

当 rs2≠x0 时，rs2 所持值的 SXLEN-1:ASIDMAX 位将保留给未来标准使用。在标准扩展定义其用途之前，软件应将其置 0，当前的实现应将其忽略。此外，如果 ASIDLEN<ASIDMAX，则实现时应忽略 rs2 中所持值的 ASIDMAX-1:ASIDLEN 位。

内存管理数据结构的隐式读取可能会返回地址的任何翻译,而该地址在最近一次 SFENCE.VMA 之后

的任何时间都是有效的。SFENCE.VMA 所隐含的排序方式无法将对内存管理数据结构的隐式读取和写入与标准 RVWMO 排序规则清晰地交互，从而将其置于全局内存顺序中。特别是，即使 SFENCE.VMA 在随后的隐式访问之前对之前的显式访问进行排序，并且这些隐式访问排序在相关的显式访问之前，SFENCE.VMA 也不一定会在全局内存排序中将之前的显式访问排在随后的显式访问之前。这些隐式加载也无需遵守正常的程序顺序语义，与之前对同一地址的加载或存储无关。

本规范的一个结果是，实施过程中可以使用自最近一次包含该地址的 SFENCE.VMA 以来任何时候有效的地址翻译。特别是，如果修改了叶子 PTE，但未执行从属 SFENCE.VMA，则将使用旧的翻译或新的翻译，但选择是不可预测的。除此之外，该行为是明确定义的。

在传统的 TLB 设计中，多个条目有可能与单个地址相匹配，例如，如果一个页面升级为超级页，而没有首先清除原来的非叶子 PTE 有效位并执行 rs1=x0 的 SFENCE.VMA，则有可能出现这种情况。在这种情况下，类似的注释也适用：使用旧的非叶子 PTE 还是新的叶子 PTE 是不可预测的，但在其他方面的行为是明确定义的。

该规范的另一个结果是，使用一组宽度小于 PTE 宽度的存储空间来更新 PTE 通常是不安全的，因为实现可以在任何时候读取 PTE，包括只有部分存储空间生效的时候。

本规范允许缓存 V（有效）位清0的 PTE。必须编写操作系统来应对这种可能性，但要提醒实施者，急于缓存无效的 PTE 会导致额外的页面故障，从而降低性能。

实现必须只对 satp 寄存器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，并且必须只对因执行指令而产生的隐式访问引发异常，而不是那些推测性执行的访问。

对 sstatus 字段 SUM 和 MXR 的更改立即生效，无需执行 SFENCE.VMA 指令。将 satp.MODE 从赤字模式改为其他模式，反之亦然，也立即生效，无需执行 SFENCE.VMA 指令。同样，对 satp.ASID 的更改也会立即生效。

以下常见情况通常需要执行 SFENCE.VMA 指令：

当软件回收 ASID（即将其与不同的页表重新关联）时，应首先更改 satp，使其指向使用回收 ASID 的新页表，然后执行 SFENCE.VMA，并将 rs1=x0 和 rs2 设置为回收的 ASID。或者，软件也可以在 satp 中加载不同 ASID 的同时执行相同的 SFENCE.VMA 指令，条件是下一次用回收的 ASID 加载 satp 时，同时用新的页表加载 satp。

如果执行过程不提供 ASID，或者软件选择始终使用 ASID 0，那么在每次写入 satp 后，软件都应执行 SFENCE.VMA，同时 rs1=x0。在没有修改全局译码的常见情况下，rs2 应设置为 x0 以外的寄存器，但其值应为 0，这样全局译码就不会被刷新。

如果软件修改了非叶 PTE，则应在 rs1=x0 时执行 SFENCE.VMA。如果遍历路径上的任何 PTE 设置了 G 位，则 rs2 必须为 x0；否则，rs2 应设置为正在修改翻译的 ASID。

如果软件修改了页 PTE，则应执行 SFENCE.VMA，并将 rs1 设置为页面内的虚拟地址。如果遍历路径上的任何 PTE 设置了 G 位，则 rs2 必须为 x0；否则，rs2 应设置为正在修改转换的 ASID。

对于增加叶 PTE 权限以及将无效 PTE 更改为有效叶的特殊情况，软件可选择延迟执行 SFENCE.VMA。在修改 PTE 后但尚未执行 SFENCE.VMA 时，系统可能会使用新权限或旧权限。如果使用旧权限，可能会导致页面故障异常。在这种情况下，软件应按照之前提到的要点执行 SFENCE.VMA。

如果 硬件线程 使用地址转换缓存，那么该缓存必须是该 硬件线程 的私有缓存。具体而言，ASID 的含义是 硬件线程 的本地含义；软件可以选择使用相同的 ASID 来指代不同 硬件线程 上的不同地址空间。

未来的扩展可以重新定义 ASID，使其在整个 SEE 中具有全局性，从而实现共享翻译缓存和广播 TLB 崩溃硬件支持等选项。不过，随着操作系统的发展，使用新颖的 ASID 管理技术大大缩小了 TLB 崩溃的范围，我们预计本地 ASID 方案仍将因其简单性和更好的可扩展性而具有吸引力。

对于将 satp.MODE 设置为只读 0（始终为空）的实现，尝试执行 SFENCE.VMA 指令可能会引发非法指令异常。

10.3 Sv32：基于页面的 32 位虚拟内存系统

当将 Sv32 写入 satp 寄存器的 MODE 字段（请参阅监视器地址转换和保护 satp 寄存器）时，监视器将在 32 位分页虚拟内存系统中运行。在这种模式下，管理程序和用户虚拟地址通过遍历一个 radix-tree 页表转换为管理程序物理地址。SXLEN=32 时支持 Sv32,其设计包括足以支持基于 Unix 的现代操作系统的机制。

最初的 RISC-V 分页虚拟内存架构是作为支持现有操作系统的直接实现而设计的。我们对页表布局进行了架构设计，以支持硬件页表步行器。软件 TLB 补充是高性能系统的性能瓶颈，对于解耦的专用协处理器尤其麻烦。实施时，可以选择使用机器模式陷阱处理程序来实现软件 TLB 补充，作为M模式的扩展。

某些 ISA 在结构上暴露了虚拟索引、物理标记的高速缓存，即通过不同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高速缓存集内。隐含地说，本规范不允许在架构上暴露这种行为。

Sv32 实现支持 32 位虚拟地址空间，并将其划分为多个页。如 Sv32 虚拟地址所示，Sv32 虚拟地址被划分为虚拟页码（VPN）和页偏移量。如果在 satp 寄存器的 MODE 字段中选择了 Sv32 虚拟内存模式，则会通过一个两级页表将上位虚拟地址转换为上位物理地址。20 位 VPN 被转换为 22 位物理页码 (PPN)，而 12 位页面偏移则不被转换。然后使用任何物理内存保护结构（[pmp]）对生成的监管级物理地址进行检查，然后直接转换为机器级物理地址。如有必要，会将监管级物理地址 0 扩展到实现中的物理地址位数。

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，将直接生成 34 位物理地址，因此无需进行0扩展。当 satp.MODE 的值为 Bare 时，32 位虚拟地址会被转换（不做修改）为 32 位物理地址，然后该物理地址会被0扩展为 34 位机器级物理地址。

Sv32 页表由 210 个页表项 PTE 组成，每个页表项有 4 个字节。页表的大小与页面大小相同，必须始终与页面边界对齐。根页表的物理页码存储在 satp 寄存器中。

Sv32 的 PTE 格式如 Sv32 页面表项所示。V 位表示 PTE 是否有效；如果 V 位为 0，则 PTE 中的所有其他位都不重要，软件可以自由使用。权限位 R、W 和 X 分别表示页面是否可读、可写和可执行。当这三个位都为 0 时，PTE 是指向页表下一级的指针；否则，它就是一个叶 PTE。可写页面也必须标记为可读；相反的组合保留给将来使用。PTE R/W/X 字段的编码。

试图从没有执行权限的页面获取指令时，会引发获取页面故障异常。试图执行有效地址位于无读取权限页面内的加载或加载保留指令，会引发加载页面故障异常。试图执行有效地址位于无写入权限页面内的存储、存储条件或 AMO 指令时，会引发存储页面故障异常。

AMO 从不引发加载页面故障异常。由于任何不可读页面也是不可写的，因此尝试在不可读页面上执行 AMO 时，总是会引发存储页面故障异常。

U 位表示 U 模式是否可以访问页面。U 模式软件只能在 U=1 时访问页面。如果设置了 sstatus 寄存器中的 SUM 位，则监管模式软件也可以访问 U=1 的页面。不过，监管代码通常在 SUM 位清 0 的情况下运行，在这种情况下，监管代码在访问 U 模式页面时会出现故障。与 SUM 无关，监管程序不得在 U=1 的页面上执行代码。

另一种 PTE 格式将支持监管员和用户的不同权限。我们省略了这一功能，因为它在很大程度上与 SUM 机制（参见 sstatus 寄存器中的内存权限）是多余的，而且需要在 PTE 中占用更多的编码空间。

G 位表示全局映射。全局映射是指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后续级别中的所有映射都是全局映射。需要注意的是，不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为全局则是一个软件错误，在切换到一个对该地址范围有不同非全局映射的地址空间后，会不可预测地导致任一映射被使用。

全局映射无需多余地存储在多个 ASID 的地址转换缓存中。此外，当使用 rs2≠x0 执行 SFENCE.VMA 指令时，无需从本地地址转换缓存中刷新这些映射。

RSW 字段保留给监控软件使用；执行时应忽略该字段。

每个叶 PTE 包含一个已访问 (A) 位和一个脏 (D) 位。A 位表示自上次清除 A 位后，虚拟页面已被读取、写入或获取。D 位表示自上次清除 D 位后虚拟页面已被写入。

在 FENCE 之后下令进行的内存访问导致的 PTE 更新本身并不按 FENCE 排序。

较简单的实现方式可能会命令页表项（PTE）更新在所有后续显式内存访问之前进行，而不是确保页表项更新在相关虚拟页面的后续显式内存访问之前精确排序。

本规范之前的版本要求 PTE A 位更新必须精确，但允许 A 位根据推测结果更新，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提示，但不要求功能正确性的精确性。另一方面，由于 D 位会影响页面驱逐的功能正确性，因此 D 位更新仍需精确，并按程序顺序执行。

当然，仍允许实现者仅以精确的方式执行 A 位和 D 位更新。

在这两种情况下，要求原子性可确保 PTE 更新不会被其他写入页表的中断，因为这种中断可能会导致 A/D 位被设置在已被重复用于其他目的的 PTE 上，或设置在已被回收用于其他目的的内存上，等等。简单的实现可能会产生页面故障异常。

执行程序永远不会清除 A 和 D 位。如果上位机软件不依赖已访问位和/或脏位，例如不将内存页交换到二级存储空间，或者内存页被用于映射 I/O 空间，则应始终在 PTE 中将其设为 1，以提高性能。

任何级别的 PTE 都可以是叶 PTE，因此除了 4 KiB 页面外，Sv32 还支持 4 MiB 的巨页面。巨页

面必须在虚拟和物理上与 4 MiB 边界对齐；如果物理地址对齐不足，就会引发页面故障异常。

对于非叶 PTE，D、A 和 U 位保留给未来标准使用。在标准扩展定义它们的用途之前，必须由软件清除它们，以实现前向兼容性。

对于同时使用基于页的虚拟内存和 “A “标准扩展的实现，LR/SC 保留集必须完全位于单个基本物理页（即自然对齐的 4 KiB 物理内存区域）内。

在某些实现中，错位加载、存储和指令获取也可能被分解成多次访问，其中一些访问可能在页面故障异常发生前成功。具体而言，即使另一部分未通过异常检查，通过异常检查的错位存储部分也可能变得可见。对于比 XLEN 位更宽的存储（例如 RV32D 中的 FSD 指令），即使存储地址是自然对齐的，也可能出现同样的行为。

10.3.1 虚拟地址转换过程

虚拟地址 va 转换为物理地址 pa 的过程如下：

设 a 为 satp.ppn×PAGESIZE，i=LEVELS-1（对于 Sv32，PAGESIZE=212，LEVELS=2）。(对于 Sv32，PAGESIZE=212，LEVELS=2。）satp 寄存器必须处于激活状态，即有效权限模式必须是 S 模式或 U 模式。

让 pte 成为地址 a+va.vpn[i]×PTESIZE 处的 PTE 值（对于 Sv32，PTESIZE=4）。如果 pte 的访问违反了 PMA 或 PMP 检查，则引发与原始访问类型相对应的访问故障异常。

如果 pte.v=0，或 pte.r=0 和 pte.w=1，或 pte 中设置了任何保留给未来标准使用的位或编码，则停止并引发与原始访问类型相对应的页面故障异常。

否则，PTE 有效。如果 pte.r=1 或 pte.x=1，则转到步骤 5。否则，该 PTE 是指向下一级页表的指针。让 i=i-1.如果 i<0，则停止并引发与原始访问类型相对应的页面故障异常。否则，让 a=pte.ppn ×PAGESIZE 并转到步骤 2。

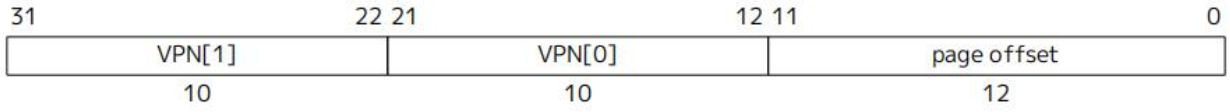


图10.10 Sv32 虚拟地址

发现一个叶 PTE。根据当前权限模式以及 mstatus 寄存器的 SUM 和 MXR 字段的值，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相对应的页面故障异常。

如果 i>0 且 pte.ppn[i-1:0] ≠ 0，则这是一个错位的超级页面；停止并引发与原始访问类型相对应的页面故障异常。

如果 pte.a=0 或原始内存访问为存储且 pte.d=0：

如果执行了 Svade 扩展，则停止并引发与原始访问类型相对应的页面故障异常。

如果存储到 pte 会违反 PMA 或 PMP 检查，则会引发与原始访问类型相对应的访问故障异常。

以原子方式执行以下步骤：

将 pte 与地址 a+va.vpn[i]×PTESIZE 处的 PTE 值进行比较。

如果值匹配，则将 pte.a 设为 1，如果原始内存访问是存储，则将 pte.d 也设为 1。

如果比较失败，返回步骤 2。

翻译成功。翻译后的物理地址如下

如果 i>0，则这是一次超级页平移，pa.ppn[i-1:0] = va.vpn[i-1:0]。

该算法中对地址转换数据结构的所有隐式访问都使用 PTESIZE 宽度。

例如，这意味着 Sv48 实现不能使用两个单独的 4B 读取来非原子地访问单个 8B PTE，而且实现执行的 A/D 位更新将被视为原子地更新整个 PTE，而不仅仅是 A 和/或 D 位（即使 PTE 值没有发生其他变化）。

步骤 2 中的隐式地址转换读取结果可保存在只读、不连贯的地址转换高速缓存中，但不与其他高速缓存共享。地址转换高速缓存可保存任意数量的条目，包括同一地址和 ASID 的任意数量的条目。如果与条目相关的 ASID 与第 0 步加载的 ASID 相匹配，或者条目与全局映射相关联，那么地址转换缓存中的条目就可以满足后续第 2 步的读取。为确保隐式读取遵守对相同内存位置的写入，必须在写入后执行 SFENCE.VMA 指令，以刷新相关的缓存翻译。

在步骤 7 中不能使用地址转换缓存；只能在内存中直接更新访问位和脏位。

允许同一地址同时存在多个地址转换缓存条目。这表明，在传统的 TLB 层次结构中，多个条目有可能与单个地址相匹配，例如，如果页面升级为超级页，而没有首先清除原始非叶 PTE 的有效位并执行 rs1=x0 的 SFENCE.VMA，或者如果在层次结构的给定级别上并行存在多个 TLB。在这种情况下，就像在写入内存管理表和随后隐式读取相同地址之间没有执行 SFENCE.VMA：无法预测是使用旧的非叶子 PTE 还是新的叶子 PTE，但行为在其他方面是明确定义的。

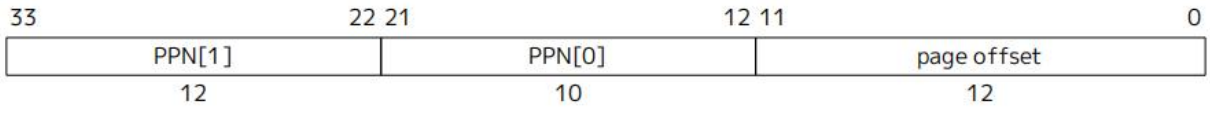


图10.11 Sv32 物理地址

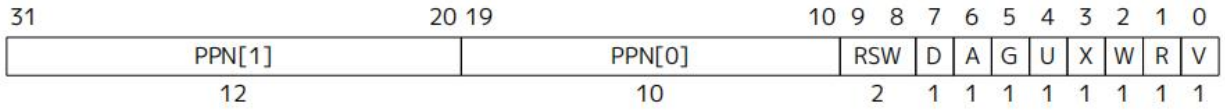


图10.12 Sv32 页表项

只要 satp 处于活动状态（如监视器地址转换和保护 (satp) 寄存器中的定义），执行程序还可以在任何时候对任何虚拟地址执行地址转换算法。这种推测性执行具有预先填充地址转换高速缓存的效果。

地址转换算法的投机执行行为与算法的非投机执行行为相同，但它们不得设置 PTE 的 dirty 位，不得触发异常，也不得创建地址转换高速缓存条目，如果这些条目在算法的投机执行开始后已被 硬件线程 执行的任何 SFENCE.VMA 指令作废。

例如，翻译算法的非推测性和推测性执行都是非法的：开始读取第 2 级页表，在 硬件线程 以 rs1=rs2=x0 执行 SFENCE.VMA 时暂停，然后继续使用现在过期的第 2 级 PTE，因为后续的隐式读取可能会用过期的 PTE 填充地址-翻译缓存。

因此，在许多实现中，rs1=x0 的 SFENCE.VMA 指令要么终止地址转换算法（针对指定的 ASID，如果适用）的所有先前启动的推测执行，要么简单地等待它们完成（在这种情况下，SFENCE.VMA 将酌情使创建的任何地址转换缓存项无效）。同样，rs1≠x0 的 SFENCE.VMA 指令通常必须确保之前启动的地址转换算法（针对指定的 ASID，如适用）的投机执行不会创建映射叶 PTE 的新地址转换缓存项，或者等待它们完成。

允许执行程序任意提前和推测性地读取翻译数据结构的一个后果是，在任何时候，通过执行算法可以到达的所有页表项都可能被加载到地址-翻译缓存中。

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。由于算法只能触及 satp 中指示的根页表所能触及的页表，因此实现的页表走行器所触及的地址范围完全受监视器控制。

该算法不允许在物理地址较窄的情况下忽略高阶 PPN 位。

10.4 Sv39：基于页面的 39 位虚拟内存系统

本节介绍 SXLEN=64 的简单分页虚拟内存系统，该系统支持 39 位虚拟地址空间。Sv39 的设计沿用了 Sv32 的整体方案，本节仅详细介绍两种方案之间的差异。

我们为 RV64 指定了多个虚拟内存系统，以缓解提供大地址空间与最小地址转换成本之间的矛盾。对于许多系统来说，39 位的虚拟地址空间已经足够，因此 Sv39 就足够了。Sv48 将虚拟地址空间增加到 48 位，但会增加页表专用的物理内存容量、页表遍历的延迟以及存储虚拟地址的硬件结构的大小。Sv57 进一步增加了虚拟地址空间、页表容量要求和转换延迟。

10.4.1 寻址和内存保护

Sv39 实现支持 39 位虚拟地址空间，并将其划分为若干页。Sv39 地址的划分如 Sv39 虚拟地址所示。指令获取地址以及加载和存储有效地址均为 64 位，其中 63-39 位必须全部等于 38 位，否则会出现页面故障异常。27 位的 VPN 通过三级页面表转换成 44 位的 PPN，而 12 位的页面偏移未经转换。

在较窄和较宽的地址之间进行映射时，RISC-V 会将较窄的物理地址0扩展到较宽的地址。64 位虚拟地址与 Sv39 的 39 位可用地址空间之间的映射不是基于0扩展，而是遵循一种根深蒂固的惯例，允许操作系统使用全尺寸（64 位）虚拟地址中最重要的一位或几位来快速区分用户和监管者地址区域。

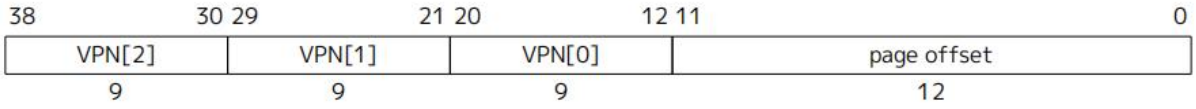


图10.13 Sv32虚拟地址

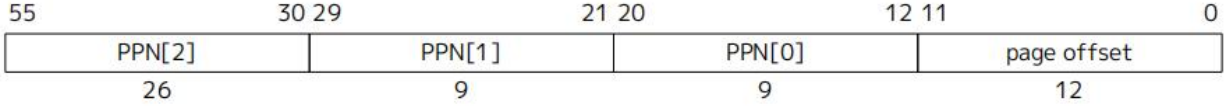


图10.14 Sv32物理地址

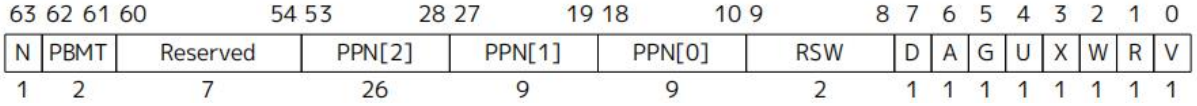


图10.15 Sv39页表项

Sv39 页表包含 29 个页表项（PTE），每个页表项 8 个字节。页表的大小与页面大小相同，必须始终与页面边界对齐。根页表的物理页码存储在 satp 寄存器的 PPN 字段中。

Sv39 的 PTE 格式如 Sv39 页面表项所示。第 9-0 位的含义与 Sv32 相同。第 63 位保留给 1.0 版 “Svnapot “NAPOT 翻译连续性扩展中的 Svnapot 扩展使用。如果未实施 Svnapot，则第 63 位仍为保留位，为了向前兼容，软件必须将其置 0，否则将引发页面故障异常。第 62-61 位保留给 1.0 版基于页面的内存类型扩展 “Svpbmt ”中的 Svpbmt 扩展使用。如果未实施 Svpbmt，则 62-61 位仍为保留位，必须由软件清 0 以实现向前兼容性，否则将引发页面故障异常。第 60-54 位保留给未来标准使用，在某些标准扩展定义其用途之前，为了向前兼容，软件必须将其置 0。如果这些位中的任何一位被设置，都会引发页面故障异常。

我们为可能的扩展预留了几个 PTE 位，通过跳过页表级，减少内存使用和 TLB 重填延迟，提高对稀疏地址空间的支持。这些预留位还可用于促进研究实验。这样做的代价是减少物理地址空间，但目前已经足够。当不再足够时，未分配的保留位可用于扩展物理地址空间。

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 兆页面和 1 GiB 千兆页面，每个页面都必须在虚拟和物理上对齐到与其大小相等的边界。如果物理地址对齐不足，就会引发页面故障异常。

除了 LEVELS 等于 3 和 PTESIZE 等于 8 之外，虚拟地址到物理地址的转换算法与虚拟地址转换过程相同。

10.5 Sv48：基于页面的 48 位虚拟内存系统

本节介绍一种简单的分页虚拟内存系统，其中 SXLEN=64，支持 48 位虚拟地址空间（即 Sv48） 。Sv48 适用于那些 39 位虚拟地址空间（如 Sv39）不足以满足需求的系统。Sv48 在设计上紧随 Sv39，只是比其多增加了一级页表，因此本章主要介绍这两种方案之间的区别。

支持 Sv48 的实现也必须支持 Sv39。

支持 Sv48 的系统基本上也可以免费支持 Sv39，因此应该这样做，以保持与假定使用 Sv39 的监控软件的兼容性。

10.5.1 寻址和内存保护

Sv48 实现支持 48 位虚拟地址空间，分为多个页面。Sv48 地址的划分如 Sv48 虚拟地址所示。指令获取地址以及加载和存储有效地址均为 64 位，其中第 63-48 位必须全部等于第 47 位，否则会出现页面故障异常。36 位 VPN 通过一个四级页表转换成 44 位 PPN，而 12 位页面偏移则不转换。

Sv48 的 PTE 格式如 Sv48 页面表项所示。63-54 位和 9-0 位的含义与 Sv39 相同。任何级别的 PTE 都可以是叶 PTE，因此除了页之外，Sv48 还支持兆页、千兆页和太页，每个页都必须在虚拟和物理上对齐到与其大小相等的边界。如果物理地址未充分对齐，则会引发页面故障异常。

除了 LEVELS 等于 4 和 PTESIZE 等于 8 之外，虚拟地址到物理地址的转换算法与虚拟地址转换过程相同。

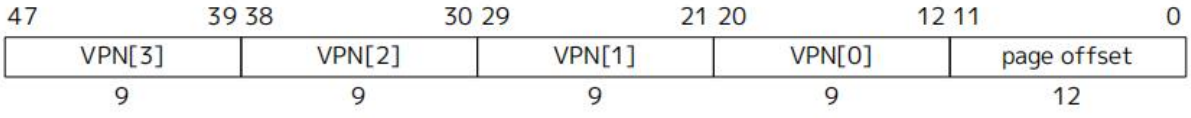


图10.16 Sv48虚拟地址

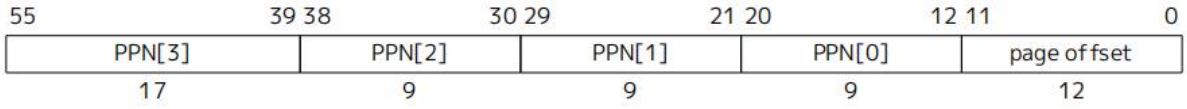


图10.17 Sv48物理地址

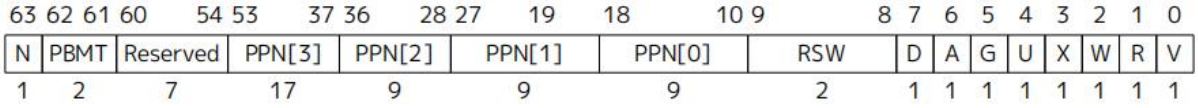


图10.18 Sv48页表项

10.6 Sv57：基于页面的 57 位虚拟内存系统

本节介绍一个为 RV64 系统设计的简单分页虚拟内存系统，支持 57 位虚拟地址空间（即 Sv57）。Sv57 适用于 48 位虚拟地址空间（如 Sv48）无法满足需求的系统。其设计在 Sv48 的基础上增加了一级页表，因此本章主要介绍这两种方案之间的区别。

支持 Sv57 的实现也必须支持 Sv48。

支持 Sv57 的系统基本上也可以免费支持 Sv48，因此应该这样做，以保持与假定使用 Sv48 的监控软件的兼容性。

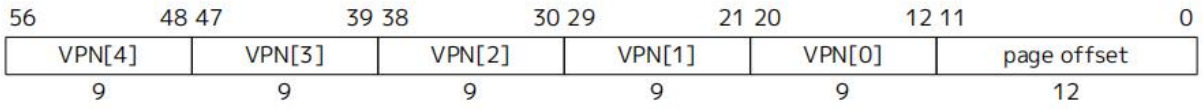


图10.19 Sv57虚拟地址

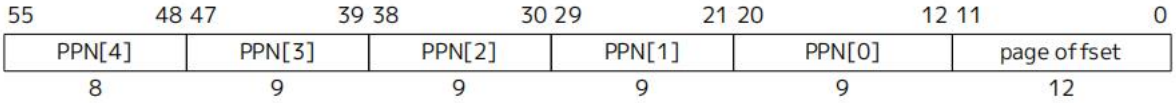


图10.20 Sv57物理地址

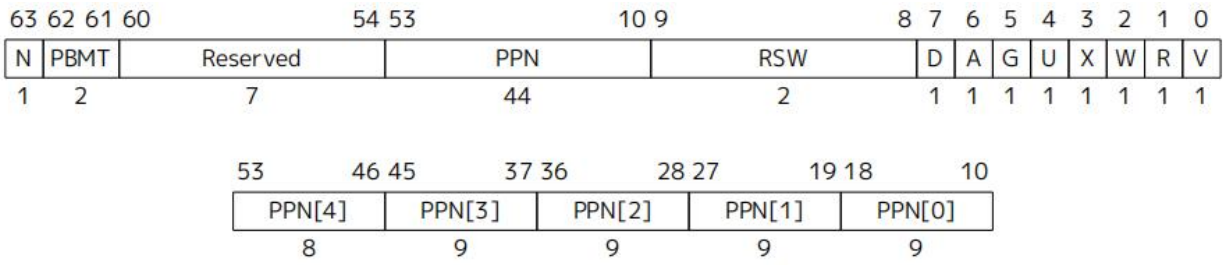


图10.21 Sv57页表项

10.6.1 寻址和内存保护

Sv57 实现支持 57 位虚拟地址空间，并将其划分为若干页。Sv57 地址的划分如 Sv57 虚拟地址所示。指令获取地址以及加载和存储有效地址均为 64 位，其中 63-57 位必须全部等于 56 位，否则会出现页面故障异常。45 位 VPN 通过一个五级页表转换成 44 位 PPN，而 12 位页面偏移则不转换。

Sv57 的 PTE 格式如 Sv57 页面表项所示。63-54 位和 9-0 位的含义与 Sv39 相同。任何级别的 PTE 都可以是叶 PTE，因此除了页之外，Sv57 还支持兆页、千兆页、太页和小页，每个页都必须在虚拟和物理上对齐到与其大小相等的边界。如果物理地址未充分对齐，则会引发页面故障异常。

除了 LEVELS 等于 5 和 PTESIZE 等于 8 之外，虚拟地址到物理地址的转换算法与虚拟地址转换过程相同。

11 用于 NAPOT 翻译连续性的 Svnapot 扩展，V1.0

在 Sv39、Sv48 和 Sv57 中，当 PTE 的 N=1 时，PTE 所代表的转换是 PTE 第 5-0 位具有相同值的连续虚拟到物理转换范围的一部分。此类范围的自然对齐 2 次幂 (NAPOT) 粒度必须大于基本页面大小。Svnapot 扩展依赖于 Sv39。

表 11.1 当 pte.N=1 时，页表条目编码

i	pte.ppn[i]	描述	pte.napotbits
0	x xxxx xxx1	保留	-
0	x xxxx xx1x	保留	-
0	x xxxx x1xx	保留	-
0	x xxxx 1000	64kb 连续区域	4
0	x xxxx 0xxx	保留	-
≥1	x xxxx xxxx	保留	-

在虚拟地址转换流程的地址转换算法中，NAPOT PTE 的行为与非 NAPOT PTE 完全相同，但以下情况除外：

当 pte.N=1 时，如果 pte 中的编码根据页面表项编码是有效的，那么 NAPOT PTE 的隐式读取将不返回 pte 的原始值，而是返回 pte 的副本，其中 pte.ppn[i][pte.napotbits-1:0] 将被 vpn[i][pte.napotbits-1:0] 所替换。当 pte.N=1 时，如果 pte 中的编码是根据页表项编码保留的，则必须引发页面故障异常。

对 NAPOT 页表条目的隐式读取可创建地址转换缓存条目，将 a + j×PTESIZE 映射到 pte 的副本，其中 pte.ppn[i][pte.napotbits-1:0] 被 vpn[i][pte.napotbits-1:0] 取代。napotbits-1:0] 替换，对于任意或所有 j，使得 j napotbits = vpn[i] napotbits，所有这些都是通过步骤 1 加载的 satp 中确定的地址空间。

采用 NAPOT PTE 的原因是，它可以作为一个或多个条目缓存在 TLB 中，这些条目代表连续区域，就好像它是由单个翻译覆盖的单个（大）页面一样。在某些情况下，这种压缩有助于减轻 TLB 的压力。该编码的设计符合现有的 Sv39、Sv48 和 Sv57 PTE 格式，因此不会干扰现有的实施或选择不实施该方案的设计。此外，它的设计也不会使地址转换算法的定义复杂化。

地址转换高速缓存抽象捕获了创建覆盖整个 NAPOT 区域的单个 TLB 条目所产生的行为。该抽象还旨在与支持 NAPOT PTE 的实现保持一致，即通过将 NAPOT 区域拆分为 TLB 条目来覆盖任何更小的 2 次幂区域大小。例如，一个 64 KiB 的 NAPOT PTE 可能会触发创建 16 个标准的 4 KiB TLB 条目，所有条目的内容都由 NAPOT PTE 生成（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，同一区域的 NAPOT PTE 将具有相同的属性、相同的 PPN 和相同的 5-0 位

值。RSW 仍保留用于监控软件控制。操作系统和/或管理程序有责任配置页表，使 NAPOT PTE 与重叠同一地址范围的其他 NAPOT 或非 NAPOT PTE 之间没有不一致。如果需要更新，操作系统一般应首先标记所有 PTE 无效，然后发出 SFENCE.VMA 指令（通过 rs1=x0 的单条 SFENCE.VMA 或 rs1≠x0 的多条 SFENCE.VMA 指令），覆盖该范围内的所有 4 KiB 区域，然后更新 PTE，如 “内存管理栅栏指令监管 ”中所述，除非已知任何不一致是良性的。如果确实存在任何不一致，那么其效果与错误使用 SFENCE.VMA 时的效果相同：将选择其中一种翻译，但选择是不可预测的。

如果实施选择使用 NAPOT PTE（或其缓存版本），则可能根本不会查询虚拟地址转换过程中算法直接指定的 PTE。因此，即使在典型的使用情况下，同一地址范围的所有映射中的 D 位和 A 位也可能不完全相同。操作系统必须查询页面的所有 NAPOT 别名，以确定该页面是否已被访问和/或变脏。如果操作系统手动设置页面的 A 和/或 D 位，建议操作系统也酌情设置其他 NAPOT 别名的 A 和/或 D 位，以避免不必要的陷阱。

与普通 PTE 一样，允许 TLB 缓存 V（有效）位清 0 的 NAPOT PTE。

根据需要，NAPOT 方案将来可能会扩展到其他中间页面大小和/或页面表的其他级别。如果需要，该编码可适应其他 NAPOT 大小。例如

i	pte.ppn[i]	描述	pte.napotbits
0	x xxxx xxx1	8 个相邻区域	1
0	x xxxx xx10	16 个相邻区域	2
0	x xxxx x100	32 KiB 连续区域	3
0	x xxxx 1000	64kb 连续区域	4
0	x xxx1 0000	128 KiB 毗连区	5
…	…	…	…
1	x xxxx xxx1	4 MiB 连续区域	1
1	x xxxx xx10	8 MiB 连续区域	2
…	…	…	…

在这种情况下，实现可能支持也可能不支持所有选项。该扩展的可发现性机制将得到扩展，允许系统软件确定支持哪些尺寸。

其他大小可能仍被故意排除在外，这样，未用于指示有效 NAPOT 区域大小的 PPN 位（例如 pte.ppn[i] 的最小有效位）将来可能会被重新用于其他用途。

不过，如果更细粒度的中间页大小支持被证明没有用处，我们选择第一步只将 64 KiB 支持标准化。

12 基于页面的内存类型的 Svpbmt 扩展，V1.0

在 Sv39、Sv48 和 Sv57 中，叶页表条目的第 62-61 位表示使用基于页的内存类型，该类型覆盖相关内存页的 PMA。PBMT 位的编码见 Sv39、Sv48 和 Sv57 PTE 中 PBMT 字段的编码。
Svpbmt 扩展依赖于 Sv39。

表 12.1 Sv39、Sv48 和 Sv57 PTE 中 PBMT 字段的编码

模式	值	请求内存属性
PMA	0	无特殊属性
NC	1	不可缓存、幂等性、弱序（RVWMO），主内存
IO	2	不可缓存、非幂等性、强序（I/O 排序），I/O 设备
-	3	保留供未来标准使用

实现可覆盖 Sv39、Sv48 和 Sv57 PTE 中 PBMT 字段编码中未明确列出的其他 PMA。例如，为了与典型 I/O 区域的特性保持一致，对 PBMT=IO 的页面进行错位内存访问可能会引发异常，即使底层区域为主存，且 PBMT=PMA 时同样的访问也会成功。

未来的扩展可能会提供更多和/或更细粒度的控制，以控制哪些 PMA 可以被覆盖。

对于非叶 PTE，第 62-61 位保留给未来标准使用。在标准扩展定义其用途之前，必须由软件清除这些位以实现向前兼容性，否则会引发页面故障异常。对于页 PTE，将第 62-61 位设置为值 3 是为将来的标准使用而保留的。在标准扩展定义该值之前，在叶子 PTE 中使用该保留值会引发页面故障异常。

当 PBMT 设置将主存页面覆盖为 I/O 或反之时，对这些页面的内存访问将遵守最终有效属性的内存排序规则，如下所示。

如果页面的基础物理内存属性为 I/O，且页面的 PBMT=NC，则对该页面的访问服从 RVWMO。不过，就 FENCE、.aq 和 .rl 而言，对此类页面的访问被视为 I/O 和主内存访问。

如果页面的底层物理内存属性是主内存，且页面的 PBMT=IO 值为 0，那么对该页面的访问将遵守强通道 0 I/O 排序规则。不过，就 FENCE、.aq 和 .rl 而言，对此类页面的访问被视为 I/O 和主存访问。

如果地址范围以 PBMT=NC 映射，则依赖 I/O 强排序规则编写的设备驱动程序将无法正常运行。因此，不鼓励使用这种配置。

通常情况下，使用 PBMT=NC 映射物理 I/O 区域仍然很有用，这样可以执行写合并和推测访问。在充分谨慎的情况下，此类优化可能会提高性能。

当 Svpbmt 与非 0 PBMT 编码一起使用时，同一物理页面的多个虚拟别名有可能同时存在，并具有不同的内存属性。通过启用 Svpbmt 的 PTE 进行的 U 模式或 S 模式映射，也有可能观察到给定物理内存区域的内存属性与通过 M 模式或 MODE=Bare 对同一页面进行的并发访问不同。在这种情况下，可能会违反由属性决定的行为（包括一致性，否则一致性不会受到影响）。

使用均不可缓存的不同属性（如 NC 和 IO）访问同一位置不会导致一致性丢失，但可能导致内存排序比通常保证的更严格属性更弱。在这种访问之间执行栅栏 iorw、iorw 指令足以防止内存排序丢失。

使用不同的缓存属性访问同一位置可能会导致一致性丢失。在这种访问之间执行以下序列可以防止一致性丢失和内存排序丢失：fence iorw，iorw，然后 cbo.flush 到该位置的地址，接着 fence iorw，iorw。

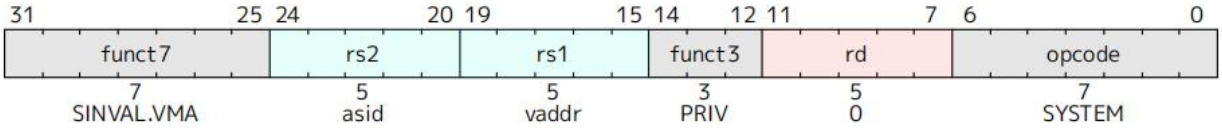
因此，如果以后可能使用原始属性引用同一位置，则必须事先重复此序列。

在某些情况下，较弱的序列可能足以防止一致性的丧失。在即将正式阐述 RVWMO 内存模型与 Zicbom 扩展指令的交互作用后，我们将详细介绍这些情况。

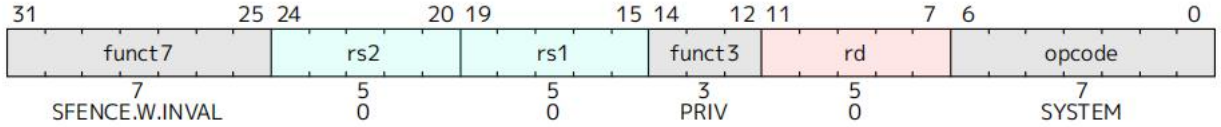
在 H 扩展中启用两阶段地址转换后，基于页面的内存类型也将分两阶段应用。首先，如果 hgatp.MODE 不等于 0，非 0 的 G 阶段 PTE PBMT 位将覆盖 PMA 中的属性，生成一组中间属性。否则，PMA 将作为中间属性。其次，如果 vsatp.MODE 不等于 0，非 0 的 VS 阶段 PTE PBMT 位将覆盖中间属性，生成访问相关页面时使用的最终属性集。否则，中间属性将用作最终属性集。

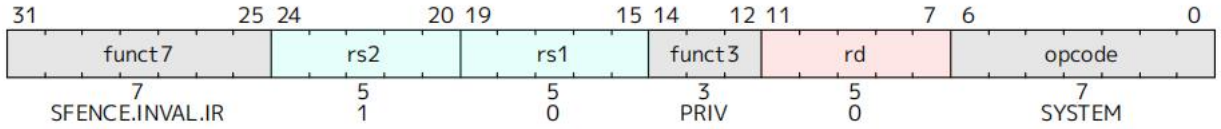
13 用于细粒度地址转换缓存失效的 Svinval 扩展，V1.0

Svinval 扩展将 SFENCE.VMA、HFENCE.VVMA 和 HFENCE.GVMA 指令拆分为粒度更细的失效和排序操作，这些操作可以在某些类别的高性能实现中更有效地进行批处理或流水线处理。



SINVAL.VMA 指令会使 rs1 和 rs2 值相同的 SFENCE.VMA 指令会使之失效的任何地址转换缓存项失效。不过，与 SFENCE.VMA 不同，SINVAL.VMA 指令只相对于 SFENCE.VMA、SFENCE.W.INVAL 和 SFENCE.INVAL.IR 指令排序，具体定义如下。





SFENCE.W.INVAL 指令保证，当前 RISC-V 硬件线程 已可见的任何先前存储都会在同一 硬件线程执行的后续 SINVAL.VMA 指令之前排序。SFENCE.INVAL.IR 指令保证，在当前内存管理数据结构隐式引用之前，对当前内存管理数据结构执行的任何先前 SINVAL.VMA 指令进行排序。

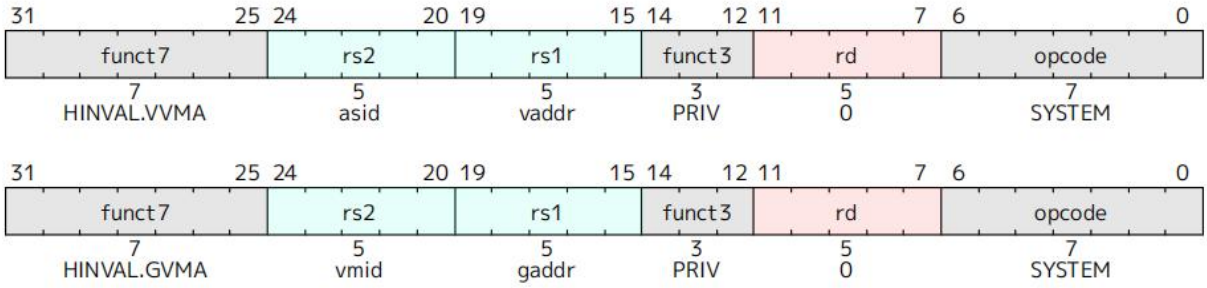
当单个哈特按顺序(但不一定是连续的)执行时,SFENCE.W.INVAL、SINVAL.VMA 和 SFENCE.INVAL.IR 序列与假设的 SFENCE.VMA 指令具有相同的效果：

*SFENCE.VMA 的 rs1 和 rs2 值与 SINVAL.VMA 中使用的值相同、

* SFENCE.W.INVAL 之前的读写被视为 SFENCE.VMA 之前的读写，并且 SFENCE.INVAL.IR 之后的读写被视为 SFENCE.VMA 之后的读写。

如果实施了管理程序扩展,Svinval 扩展还会提供两条额外的指令:HINVAL.VVMA 和 HINVAL.GVMA。这两条指令的语义与 SINVAL.VMA 相同，但它们与 SFENCE.W.INVAL 和 SFENCE.INVAL.IR 结合使用，分别取代了 HFENCE.VVMA 和 HFENCE.GVMA，而不是 SFENCE.VMA。此外，HINVAL.GVMA 使用 VMID 而不是 ASID。

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。具体而言，在 U 模式下尝试执行这些指令时，总是会引发非法指令异常；在 S 模式或 HS 模式下，当 mstatus.TVM=1 时，尝试执行 SINVAL.VMA 或 HINVAL.GVMA 也会引发非法指令异常。如果尝试在 VS 模式或 VU 模式下执行 HINVAL.VVMA 或 HINVAL.GVMA，或在 VU 模式下执行 SINVAL.VMA，都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。



在 U 模式下尝试执行 SFENCE.W.INVAL 或 SFENCE.INVAL.IR 会引发非法指令异常。在 VU 模式下执行会引发虚拟指令异常。SFENCE.W.INVAL 和 SFENCE.INVAL.IR 不受 mstatus.TVM 和 hstatus.VTVM 字段的影响，因此在 S 模式和 VS 模式下始终允许执行。

当 mstatus.TVM=1 或 hstatus.VTVM=1 时，不需要捕获 SFENCE.W.INVAL 和 SFENCE.INVAL.IR 指令，因为它们只有排序效果，而没有可见的副作用。对 SINVAL.VMA 指令的捕获足以模拟预期的整体 TLB 维护功能。

在典型使用中，软件将通过执行 SFENCE.W.INVAL 指令，对相关地址（以及可选的 ASID 或 VMID）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。

高性能实现将能够以流水线方式进行地址转换缓存失效操作，并将任何流水线停滞或其他内存排序执行推迟到 SFENCE.W.INVAL、SFENCE.INVAL.IR、SFENCE.VMA、HFENCE.GVMA 或 HFENCE.VVMA 指令执行时。

较简单的实现可分别与 SFENCE.VMA、HFENCE.VVMA 和 HFENCE.GVMA 相同地实现 SINVAL.VMA、HINVAL.VVMA 和 HINVAL.GVMA，同时将 SFENCE.W.INVAL 和 SFENCE.INVAL.IR 指令作为无操作执行。

14 用于硬件更新 A/D 位的 Svadu 扩展程序，V1.0

Svadu 扩展增加了对 PTE A/D 位硬件更新的支持和 CSR 控制。

如果实施了 Svadu 扩展，则 menvcfg.ADUE 字段是可写的。如果另外实施了管理程序扩展，henvcfg.ADUE 字段也是可写的。有关这些字段的定义，请参阅 [sec:menvcfg] 和 [sec:henvcfg] 。

寻址和内存保护定义了 A/D 位硬件更新的语义。当 A/D 位硬件更新被禁用时，Svade 扩展就会生效，它规定了需要设置 A/D 位时的例外情况。Svade 扩展也在寻址和内存保护中进行了定义。

“Svvptc 标记 PTE 有效后省略内存管理指令的扩展，1.0 版

在实施 Svvptc 扩展后，将叶 PTE 和/或非叶 PTE 的有效位从 0 更新为 1，并对 硬件线程 可见的 硬件线程 显式存储，最终将在一定时限内对该 硬件线程 对此类 PTE 的后续隐式访问可见。

Svvptc 使操作系统无需执行某些内存管理指令，如 SFENCE.VMA 或 SINVAL.VMA，当内存驻留 PTE 从无效变为有效时，这些指令通常用于同步 硬件线程 的地址转换缓存。要使 硬件线程 的地址转换缓存与内存驻留 PTE 的其他形式更新同步，包括当 PTE 从有效变为无效时，需要使用合适的内存管理指令。Svvptc 可保证在一定时间内将 PTE 从无效变为有效，从而使这些内存管理指令的执行变得多余。省略这些指令带来的性能优势超过了偶尔可能发生的无偿额外页面故障的代价。

根据微体系结构的不同，促进 Svvptc 实现的一些可能方法包括：不使用任何地址转换缓存，不在地址转换缓存中存储无效 PTE，使用有界定时器自动驱逐无效 PTE，或使地址转换缓存与修改 PTE 的存储指令一致。

15 “Svvptc“扩展：消除 PTE 生效时的内存管理屏障，V1.0

当实现 Svvptc 扩展时，若某 硬件线程（硬件线程）显式更新叶或非叶PTE的Valid位（从 0 变为 1） 并使其对该 硬件线程 可见，则在该 硬件线程 后续对这些 PTE 的隐式访问中，这些更新将在有限时间内保证可见。

通常，操作系统会在缺页异常或内存映射系统调用后将页表项（PTE）标记为有效（Valid）。在此类场景下，陷阱处理程序通常使用SRET指令从陷阱返回。当实现Svvptc扩展时，操作系统将PTE的Valid位从0改为1的显式存储操作，会在有限时间内对该硬件线程后续的隐式访问（如触发缺页的指令或访问新内存区域的指令）可见。若需强制这些PTE更新对后续所有隐式访问立即可见，传统上需使用内存管理屏障（memory-management fence）。但在支持Svvptc的情况下，有限时间内的可见性已得到保证，因此无需额外插入内存屏障指令。尽管这种方法可能导致偶发的冗余缺页，但省去内存屏障带来的性能优势远超过偶发冗余缺页的开销。

16 “Sstc“扩展：用于监管者模式定时器中断，V1.0.0

当前的特权架构规范仅定义了生成机器模式定时器中断的硬件机制（基于 mtime 和 mtimecmp 寄存器）。这导致 S 模式/HS 模式（以及 VS 模式）的定时器服务必须全部由 M 模式提供——通过从 S/HS 模式向上调用 SBI 到 M 模式（或从 VS 模式调用 HS 模式再到 M 模式）。然后，M 模式软件将这些多个逻辑定时器复用到其一个物理 M 模式定时器设施上，M 模式定时器中断处理程序将定时器中断传递回适当的较低特权模式。

此扩展旨在为监管者模式提供其自己的基于 CSR 的定时器中断设施，使其能够直接管理以提供自己的定时器服务（通过拥有自己的 stimecmp 寄存器）——从而消除了在 M 模式下模拟 S/HS 模式定时器和定时器中断生成的大量开销。此外，此扩展还为虚拟机监视器扩展添加了类似的设施，以支持 VS 模式。为了便于理解与当前 Priv 1.11/1.12 规范的差异，本文以对现有规范段落（或在现有文本中添加段落）的实际修改形式编写。扩展名称为“Sstc“（“Ss“表示特权架构和监管者级别扩展，“tc“表示 timecmp）。此扩展添加了 S 级别的 stimecmp CSR 和 VS 级别的 vstimecmp CSR。

16.1 机器级与管理级扩展功能

16.1.1 监管定时器比较寄存器（stimecmp）

此扩展添加了这个新的 CSR。stimecmp CSR 是一个 64 位寄存器，在所有 RV32 和 RV64 系统上具有 64 位精度。在仅 RV32 系统中，对 stimecmp CSR 的访问访问低 32 位，而对 stimecmph CSR 的访问访问 stimecmp 的高 32 位。stimecmp / stimecmph 的 CSR 编号为 0x14D / 0x15D（位于监管者陷阱设置 CSR 块中）。

当 time 包含的值大于或等于 stimecmp 时，监管者定时器中断将挂起，反映在 mip 和 sip 寄存器中的 STIP 位中，将这些值视为无符号整数。如果此比较的结果发生变化，它最终会反映在 STIP 中，但不一定立即反映。中断将保持挂起状态，直到 stimecmp 大于 time，通常是由于写入 stimecmp。中断将根据标准中断使能和委托规则进行处理。

如果中断处理程序推进stimecmp然后立即返回，则可能会发生虚假的定时器中断，因为在此期间STIP可能尚未下降。所有软件都应编写为假设此事件可能发生，但大多数软件应假设此事件极不可能发生。偶尔发生虚假定时器中断几乎总是比轮询STIP直到其下降更高效。

在监管者执行环境（SEE）通过SBI函数调用提供定时器设施的系统中，此SBI调用将继续支持安排定时器中断的请求。SEE将简单地使用stimecmp，根据需要更改其值。这确保了与使用此SEE设施的现有S模式软件的兼容性，同时新的S模式软件直接利用stimecmp。）

16.1.2 机器中断（mip 和 mie）寄存器

此扩展修改了这些寄存器中 STIP/STIE 位的描述如下：

如果实现了监管者模式,其 mip.STIP 和 mie.STIE 是监管者级别定时器中断的中断挂起和中断使能位。如果未实现 stimecmp 寄存器，STIP 在 mip 中是可写的，M 模式软件可以通过写入它来向 S 模式传递定时器中断。如果实现了 stimecmp（监管者模式定时器比较）寄存器，STIP 在 mip 中是只读的，并反映由 stimecmp 产生的监管者级别定时器中断信号。此定时器中断信号通过将 stimecmp 写入大于当前时间值的值来清除。

16.1.3 监管者中断（sip 和 sie）寄存器

此扩展修改了这些寄存器中 STIP/STIE 位的描述如下：

sip.STIP 和 sie.STIE 位是监管者级别定时器中断的中断挂起和中断使能位。如果实现，STIP 在 sip 中是只读的，并且要么由执行环境设置和清除（如果未实现 stimecmp），要么反映由 stimecmp 产生的定时器中断信号（如果实现了 stimecmp）。sip.STIP 位在响应由 stimecmp 生成的定时器中断时，通过将 stimecmp 写入小于或等于当前时间值的值来设置，并通过写入大于当前时间值的值来清除。

16.1.4 机器计数器使能（mcounteren）寄存器

此扩展对此寄存器中 TM 位的描述进行了如下补充：

此外，当 mcounteren 寄存器中的 TM 位清 0 时，尝试在低于 M 模式的模式下访问 stimecmp 或 vstimecmp 寄存器将导致非法指令异常。当此位置 1 时，如果实现，则允许在 S 模式下访问 stimecmp 或 vstimecmp 寄存器，并且如果实现且未被 hcounteren 中的 TM 位阻止，则允许在 VS 模式下访问 vstimecmp 寄存器（通过 stimecmp）。

16.2 Hypervisor 扩展新增功能

16.2.1 虚拟监管者定时器（vstimecmp）寄存器

此扩展添加了这个新的 CSR。vstimecmp CSR 是一个 64 位寄存器，在所有 RV32 和 RV64 系统上具有 64 位精度。在仅 RV32 系统中，对 vstimecmp CSR 的访问访问低 32 位，而对 vstimecmph CSR 的访问访问 vstimecmp 的高 32 位。vstimecmp / vstimecmph 的 CSR 编号为 0x24D / 0x25D（位于虚拟监管者寄存器 CSR 块中，并镜像 stimecmp/stimecmph 的 CSR 编号）。

当（time + htimedelta）截断为 64 位后包含的值大于或等于 vstimecmp 时，虚拟监管者定时器中断将挂起，反映在 hip 寄存器中的 VSTIP 位中，将这些值视为无符号整数。如果此比较的结果发生变化,它最终会反映在 VSTIP 中,但不一定立即反映。中断将保持挂起状态,直到 vstimecmp 大于(time + htimedelta)，通常是由于写入 vstimecmp。中断将根据标准中断使能和委托规则在 V=1 时进行处理。

在由 HS 模式虚拟机监视器实现的监管者执行环境（SEE）通过 SBI 函数调用提供定时器设施的系统中，此 SBI 调用将继续支持安排定时器中断的请求。SEE 将简单地使用 vstimecmp，根据需要更改其值。这确保了与使用此 SEE 设施的现有客户 VS 模式软件的兼容性，同时新的 VS 模式软件直接利用 vstimecmp。）

16.2.2 虚拟机监视器中断（hvip、hip 和 hie）寄存器

此扩展修改了 hip/hie 寄存器中 VSTIP/VSTIE 位的描述如下：

hip.VSTIP 和 hie.VSTIE 位是 VS 级别定时器中断的中断挂起和中断使能位。VSTIP 在 hip 中是只读的，并且是 hvip.VSTIP 和由 vstimecmp 产生的定时器中断信号（如果实现了 vstimecmp）的逻辑或。hip.VSTIP 位在响应由 vstimecmp 生成的定时器中断时，通过将 vstimecmp 写入小于或等于当前（time + htimedelta）值的值来设置，并通过写入大于当前（time + htimedelta）值的值来清除。hip.VSTIP 位在 V=0 和 V=1 时均保持定义。

16.2.3 虚拟机监视器计数器使能（hcounteren）寄存器

此扩展对此寄存器中 TM 位的描述进行了如下补充：

此外，当 hcounteren 寄存器中的 TM 位清 0 时，尝试在 VS 模式下访问 vstimecmp 寄存器（通过 stimecmp）将导致虚拟指令异常（如果 mcounteren 中的相同位已设置）。当此位和 mcounteren 中的相同位都设置为 1 时，允许在 VS 模式下访问 vstimecmp 寄存器（如果实现）。

16.3 环境配置（menvcfg 和 henvcfg）支持

此扩展的启用/禁用位在新的 menvcfg / henvcfg CSR 中提供。menvcfg 的第 63 位（或 menvcfgh 的第 31 位）——命名为 STCE（STimecmp 启用）——当设置为 1 时，为 S 模式启用 stimecmp，而 henvcfg 的相同位为 VS 模式启用 vstimecmp。这些 STCE 位是 WARL（写任意读合法），并且在此扩展未实现时硬连线为 0。

当此扩展实现且 menvcfg 中的 STCE 为 0 时，尝试在非 M 模式下访问 stimecmp 或 vstimecmp 会引发非法指令异常，henvcfg 中的 STCE 为只读 0，并且 mip 和 sip 中的 STIP 恢复为其定义的行为，就像此扩展未实现一样。此外，如果实现了 H 扩展，则 hip.VSTIP 也恢复为其定义的行为，就像此扩展未实现一样。

但当 menvcfg 中的 STCE 为 1 且 henvcfg 中的 STCE 为0时，尝试在 V = 1 时访问 stimecmp（实际上是 vstimecmp）会引发虚拟指令异常，并且 hip 中的 VSTIP 恢复为其定义的行为，就像此扩展未实现一样。

17 “Sscofpmf“扩展：用于计数溢出和基于模式的过滤，V1.0.0

当前的特权架构规范定义了 mhpmevent CSR，用于选择和控制通过相关的 hpmcounter CSR 进行事件计数，但并未对这些 CSR 中的任何字段进行标准化。对于至少是 Linux 类富操作系统（rich-OS）系统来说，标准化某些广泛需求的基本功能是可取的（这些需求在过去一年多的 RISC-V 列表中已经提出，并且也是过去提案的主题）。这使得可以拥有标准的上游软件支持，从而消除了实现自定义软件支持的需求。

此扩展旨在在现有的 mhpmevent CSR 中实现这一目标（并相应地避免了不必要地创建全新的 CSR 集——除了一个新的 CSR 之外）。

此扩展专注于解决由不同人员提出的两个基本且易于理解的需求。为了便于理解与当前 Priv

1.11/1.12 规范的差异，本文以对现有规范段落（或在现有文本中添加段落）的实际修改形式编写。

扩展名称为“Sscofpmf“（“Ss“表示特权架构和监管者级别扩展，“cofpmf“表示计数溢出和特权模式过滤）。

请注意，新的计数溢出中断将被视为标准本地中断，并分配给 mip/mie/sip/sie 寄存器中的第 13 位。

17.1 计数溢出控制

以下位被添加到 mhpmevent 中：

表 17.1 相关特权模式

简称	描述
OF	计数器溢出时设置的溢出状态和中断禁用位
MINH	如果设置，则禁止 m 模式下的事件计数
SINH	如果设置，则禁止 S/ hs 模式下的事件计数
UINH	如果设置，则禁止 u 模式下的事件计数
VSINH	如果设置，则禁止 vs 模式下的事件计数
VUINH	如果设置，则禁止计数 vu 模式下的事件

对于每个 xINH 位，如果未实现相关的特权模式，则该位为只读 0。五个 xINH 位中的每一个，当设置为 1 时，会抑制在特权模式 x 下的事件计数。这些位全为 0 时，表示在所有模式下计数事件。

当相应的 hpmcounter 溢出时，OF 位被设置，并保持设置状态，直到被软件写入。由于 hpmcounter 的值是无符号值，溢出被定义为实现的计数器位的无符号溢出。请注意，溢出后不会丢失信息，因为计数器会回绕并继续计数，而粘性 OF 位保持设置状态。

如果实现了监管者模式，32 位的 scountovf 寄存器包含所有 32 个 mhpmevent 寄存器中 OF 位的只读影子副本。

如果 hpmcounter 在关联的 OF 位为 0 时溢出，则会生成“计数溢出中断请求”。如果 OF 位为 1，则不会生成中断请求。因此，OF 位还充当关联 hpmcounter 的计数溢出中断禁用功能。

计数溢出永远不会由对 mhpmcountern 或 mhpmeventn 寄存器的写入引起，而只会由计数器寄存器的硬件增量引起。

此计数溢出中断请求信号被视为标准本地中断，对应于 mip/mie/sip/sie 寄存器中的第 13 位。mip/sip 中的 LCOFIP 位和 mie/sie 中的 LCOFIE 位分别是此中断的中断挂起和中断使能位。(“LCOFI“ 代表 “Local Count Overflow Interrupt”。)

由 hpmcounter 生成的计数溢出中断请求会设置关联的 OF 位。当 OF 位被设置时，它最终（但不一定立即）会设置 mip/sip 寄存器中的 LCOFIP 位。LCOFIP 位由软件在服务由一个或多个计数溢出引起的中断之前清除。mideleg 寄存器控制此中断是委托给 S 模式还是 M 模式。

没有单独的溢出状态和溢出中断使能位。实际上，启用溢出中断生成（通过清除 OF 位）与将计数器初始化为起始值一起完成。一旦计数器溢出，必须重新初始化它和 OF 位，才能生成另一个溢出中断。

软件可以通过维护一个反映哪些计数器处于活动状态并最终会溢出的位掩码，来区分新溢出的计数器（尚未由溢出中断处理程序处理）与已经处理过或配置为在溢出时不生成中断的溢出计数器。

17.2 监管者计数溢出 (scountovf) 寄存器

此扩展添加了 scountovf CSR，一个 32 位只读寄存器，包含 29 个 mhpmevent CSR（mhpmevent3 - mhpmevent31）中 OF 位的影子副本——其中 scountovf 位 X 对应于 mhpmeventX。

此寄存器使监管者级别的溢出中断处理程序软件能够快速轻松地确定哪些计数器已溢出(无需进行执行环境调用或一系列最终到达 M 模式的调用）。

对位 X 的读取访问受与 S 模式（或 VS 模式）访问 hpmcounter CSR 相同的 mcounteren（或 mcounteren 和 hcounteren）CSR 的约束。在 M 模式下，scountovf 位 X 始终可读。在 S/HS 模式下，当 mcounteren 位 X 设置时，scountovf 位 X 可读，否则读取为 0。同样，在 VS 模式下，当 mcounteren 位 X 和 hcounteren 位 X 都设置时，scountovf 位 X 可读，否则读取为 0。

18 “H“扩展：用于支持虚拟机监控器（Hypervisor）的功能扩展，V1.0

本章节阐述了 RISC-V 虚拟机监视器（hypervisor）扩展，该扩展通过虚拟化监管模式（supervisor-level）架构，以支持在类型 1 或类型 2 虚拟机监视器之上高效托管客户操作系统。此扩展将监管模式转变为支持虚拟机监控的扩展监管模式（HS-mode，简称 hypervisor 模式），在此模式下运行的是具备托管能力的操作系统或虚拟机监视器本身。此外，hypervisor 扩展还引入了另一层地址转换机制，即从客户物理地址到监管物理地址的转换，以此来为运行在虚拟机中的操作系统虚拟化内存及内存映射 I/O 子系统。HS 模式在功能上与 S-mode 相同，但增加了控制新地址转换阶段和支持在虚拟 S 模式（VS 模式）下托管客户操作系统的额外指令和控制状态寄存器（CSRs）。标准的 S 模式操作系统无需修改即可在 HS 模式下运行，或作为 VS 模式下的客户系统运行。

如同操作系统通常在 S 模式下所做的那样，在 HS 模式下，操作系统或虚拟机监视器通过相同的 SBI（系统二进制接口）与机器交互。预计 HS 模式的虚拟机监视器会为其 VS 模式的客户操作系统实现相应的 SBI 接口。

虚拟机监视器扩展依赖于具有 32 个 x 寄存器的“I“基础整数指令集架构（RV32I 或 RV64I），而非仅有 16 个 x 寄存器的 RV32E 或 RV64E。控制状态寄存器 mtval 不能是只读且值为 0 的，并且必须支持基于页面的标准地址转换机制，对于 RV32 是 Sv32，而对于 RV64 至少需要支持 Sv39。

通过设置控制状态寄存器 misa 中的第 7 位（对应字母 H），可以启用虚拟机监视器扩展。鼓励实现此扩展的 RISC-V 硬件线程不要将 misa[7]固定为某一值，以便能够根据需要禁用该扩展功能。

基础特权架构旨在简化经典虚拟化技术的应用，即让客户操作系统在用户级别运行，因为少数特权指令能够被轻易检测并捕获。虚拟机监视器扩展通过减少这类捕获操作的频率，从而提升了虚拟化的性能表现。

虚拟机监视器扩展在设计时已考虑到在未实现该扩展的平台上的高效模拟能力，通过在S模式下运行虚拟机监视器，并在访问虚拟机监视器的控制状态寄存器（CSR）及维护影子页表时陷入M模式来实现。对于类型2的虚拟机监视器而言，大多数CSR访问属于有效的S模式访问，因此无需陷入。同样地，虚拟机监视器也能以类似方式支持嵌套虚拟化。

18.1 特权模式

当前的虚拟化模式，用 V 表示，指示硬件线程当前是否正在客户环境中执行。当 V=1 时，硬件线程要么处于虚拟 S 模式（VS 模式），要么位于运行在 VS 模式上的客户操作系统之上的虚拟 U 模式（VU 模式）。当 V=0 时，硬件线程可能处于 M 模式、HS 模式，或者在运行于 HS 模式的操作系统之上的 U 模式。虚拟化模式同时也标志着两阶段地址转换是否激活（V=1）或未激活（V=0）。HPrivModes 列出了

支持虚拟机监视器扩展的 RISC-V 硬件线程可能拥有的特权模式，如表 1.1。

表 18.1 具有管理程序扩展的特权模式

虚拟化模式 (V)	标称特权	缩写	名称	两级地址转换
0 0 0	U S M	U 模式 HS 模式 M 模式	用户模式 虚拟机监视器扩展的监管模式 机器模式	关 关 关
1 1	U S	VU 模式 VS 模式	虚拟用户模式 虚拟机监视器模式	开 开

对于特权模式 U 和 VU，nominal privilege mode 为 U，而对于特权模式 HS 和 VS，其名义特权模式为 S。HS 模式比 VS 模式级别更高，VS 模式比 VU 模式级别更高。在 U 模式下执行时，VS 模式中断被全局禁用。

此描述未考虑用户模式（U模式）或虚拟用户模式（VU模式）中断的可能性，若未来采纳了关于用户级别中断的扩展，将会对此进行修订。

18.2 管理程序和虚拟主管 CSR

在 HS 模式下运行的操作系统或管理程序使用监管 CSR 与异常、中断和地址转换子系统进行交互。为 HS 模式提供了额外的控制状态寄存器（CSR），但不对 VS 模式开放，这些寄存器用于管理两阶段地址转换及控制 VS 模式客户系统的行为，包括：hstatus、hedeleg、hideleg、hvip、hip、hie、hgeip、hgeie、henvcfg、henvcfgh、hcounteren、htimedelta、htimedeltah、htval、htinst 以及 hgatp。

此外，几个虚拟监管者控制状态寄存器（VS CSR）是普通监管者 CSR 的副本。例如，vsstatus 是 VS CSR，它复制了通常的 sstatus CSR。

当 V=1 时，虚拟监管者控制状态寄存器（VS CSR）将替代相应的监管者 CSR，接管通常监管者 CSR 的所有功能，除非另有规定。在正常情况下读取或修改监管者 CSR 的指令，此时将转而访问对应的 VS CSR。当 V=1 时，尝试通过其独立的 CSR 地址直接读取或写入 VS CSR 将引发虚拟指令异常。（从用户模式（U 模式）尝试访问则如常引发非法指令异常。）VS CSR 仅能在机器模式（M 模式）或超级监管者模式（HS 模式）下以其自身地址被访问。

当 V=1 时，被 VS CSR 取代的常规 HS 级别监管者 CSR 将保留其值，但不会影响机器的行为，除非有特别说明。相反，当 V=0 时，VS CSR 通常不会影响机器的行为，除了可以通过 CSR 指令进行读取和写入操作。

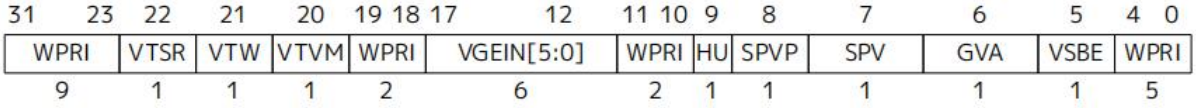
一些标准的监管者 CSR(例如 senvcfg、scounteren 和 scontext，可能还有其他)没有对应的 VS CSR。这些监管者 CSR 即使在 V=1 时也继续保留其常规功能和可访问性，只是用 VS 模式和 VU 模式分别替代 HS 模式和 U 模式。虚拟机监控软件（Hypervisor）需要根据需要手动交换这些寄存器的内容。

匹配与CSR（控制和状态寄存器）仅存在于那些必须被复制的监管者CSR中，这些主要是那些在陷阱发生时自动写入的寄存器，或者在陷阱进入后和/或SRET指令执行前立即影响指令执行的寄存器，当软件无法在精确的时机交换CSR时。目前，大多数监管者CSR都属于这一类，但未来的CSR可能不会。

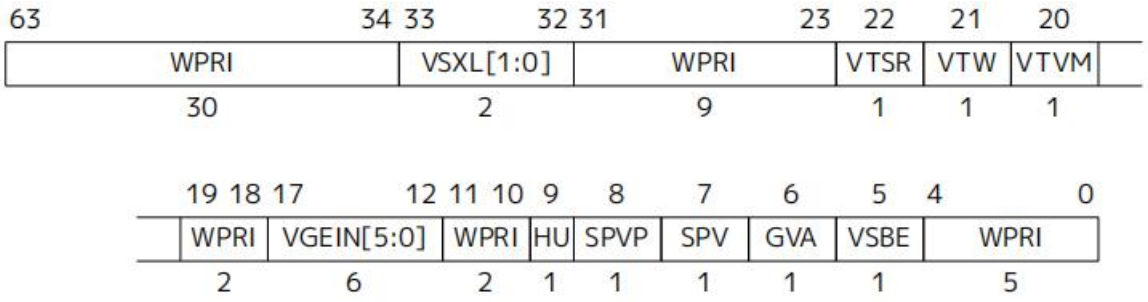
在本章节中，我们使用术语 HSXLEN 来指代在 HS 模式下执行时的有效 XLEN，而 VSXLEN 则指代在 VS 模式下执行时的有效 XLEN。

18.2.1 管理程序状态（hstatus）寄存器

hstatus 寄存器是一个 HSXLEN 位的读写寄存器，其格式在 HSXLEN=32 时如图 18.1 所示，在 HSXLEN=64 时如图 18.2 所示。hstatus 寄存器提供了类似于 mstatus 寄存器的功能，用于追踪和控制 VS 模式客体的异常行为。



18.1 HSXLEN=32 时的虚拟机监视器状态寄存器（hstatus）



18.2 HSXLEN=64 时的虚拟机监视器状态寄存器（hstatus）

VSXL 字段控制着 VS 模式下的有效 XLEN（称为 VSXLEN），它可能与 HS 模式下的 XLEN（HSXLEN）不同。当 HSXLEN=32 时，VSXL 字段不存在，且 VSXLEN=32。当 HSXLEN=64 时，VSXL 是一个 WARL 字段，其编码方式与 misa 的 MXL 字段相同，如 misabase 所示。具体而言，实现可以将 VSXL 设置为只读字段，其值始终确保 VSXLEN=HSXLEN。如果 HSXLEN 从 32 位更改为更宽的宽度，并且如果字段 VSXL 未被限制为单一值，则它将获得对应于不超过新 HSXLEN 的最宽支持宽度的值。

hstatus 寄存器的字段 VTSR、VTW 和 VTVM 与 mstatus 寄存器的字段 TSR、TW 和 TVM 定义类似，但它们仅在 VS 模式下影响执行，并引发虚拟指令异常而非非法指令异常。当 VTSR=1 时，在 VS 模式下尝试执行 SRET 会引发虚拟指令异常。当 VTW=1 时（并假设 mstatus.TW=0），在 VS 模式下尝试执行 WFI 如果未在实现特定的有限时间限制内完成，则会引发虚拟指令异常。实现可以在 VTW=1（且 mstatus.TW=0）时，使 WFI 在 VS 模式下总是引发虚拟指令异常，即使执行该指令时有挂起的全局禁用中断。当 VTVM=1 时，在 VS 模式下尝试执行 SFENCE.VMA 或 SINVAL.VMA 或访问 CSR satp 会引发虚拟指令异常。

VGEIN（虚拟客体外中断号）字段用于选择 VS 级别外中断的客体外中断源。VGEIN 是一个 WLRL 字段，必须能够容纳从 0 到最大客体外中断号（称为 GEILEN）之间的值，包括 0 和 GEILEN。当 VGEIN=0 时，没有选择任何客体外中断源用于 VS 级别外中断。GEILEN 可能为 0，此时 VGEIN 可能为只读的 0。客体外中断的解释见 hgeinterruptregs，VGEIN 的使用在 hinterruptregs 中有进一步说明。

字段 HU（U 模式下的超级用户）控制虚拟机加载/存储指令 HLV、HLVX 和 HSV 是否也可以在用户模式（U 模式）下使用。当 HU=1 时，这些指令可以在 U 模式下执行，与在 HS 模式下相同。当 HU=0 时，所有超级用户指令在 U 模式下都会引发非法指令异常。

HU位允许超级用户的一部分在用户模式（U模式）下运行，以增强对软件错误的防护，同时仍保留对虚拟机内存的访问权限。

SPV 位（超级用户先前的虚拟化模式）在发生陷阱进入 HS 模式时由实现写入。正如 sstatus 中的 SPP 位在陷阱发生时被设置为（名义上的）特权模式一样，hstatus 中的 SPV 位在陷阱发生时被设置为虚拟化模式 V 的值。当 V=0 时执行 SRET 指令，V 将被设置为 SPV。

当 V=1 且发生陷阱进入 HS 模式时，SPVP 位（超级用户先前的虚拟特权模式）被设置为陷阱发生时的名义特权模式，与 sstatus.SPP 相同。但如果陷阱发生前 V=0，则在陷阱进入时 SPVP 保持不变。SPVP 控制由虚拟机加载/存储指令 HLV、HLVX 和 HSV 执行的显式内存访问的有效特权。

如果没有SPVP，如果指令HLV、HLVX和HSV改为依赖sstatus.SPP来确定其内存访问的有效特权，那么即使HU=1，用户模式（U模式）也无法访问VS级别的虚拟机内存，因为使用SRET进入U模式总是会使得SPP=0。与SPP不同，字段SPVP在HS模式和U模式之间来回切换时不会被改变。

字段 GVA（客户虚拟地址）在发生陷阱进入 HS 模式时由实现写入。对于任何将客户虚拟地址写入 stval 的陷阱（断点、地址未对齐、访问故障、页面故障或客户页面故障），GVA 设置为 1。对于任何其他进入 HS 模式的陷阱，GVA 被设置为 0。

对于将非0值写入stval的断点和内存访问陷阱，GVA与字段SPV是冗余的（这两个位的设置相同），除非HLV、HLVX或HSV指令的显式内存访问导致故障。在这种情况下，SPV=0但GVA=1。

VSBE 位是一个 WARL 字段，用于控制从 VS 模式进行的显式内存访问的字节序。如果 VSBE=0，从 VS 模式进行的显式加载和存储内存访问为小端序，如果 VSBE=1，则为大端序。VSBE 还控制对所有 VS 级内存管理数据结构（如页表）的隐式访问的字节序。实现可以将 VSBE 设置为只读字段，始终指定与 HS 模式相同的字节序。

18.2.2 管理程序陷阱委派（“hedeleg“和“hideleg”）寄存器

寄存器 hedeleg 是一个 64 位的读写寄存器，其格式如 hedelegreg 所示。寄存器 hideleg 是一个 HSXLEN 位的读写寄存器，其格式如 hidelegreg 所示。默认情况下，所有特权级别的陷阱都在 M 模式下处理,尽管 M 模式通常使用 medeleg 和 mideleg CSR 将一些陷阱委托给 HS 模式。hedeleg 和 hideleg CSR 允许将这些陷阱进一步委托给 VS 模式客户；它们的布局与 medeleg 和 mideleg 相同。

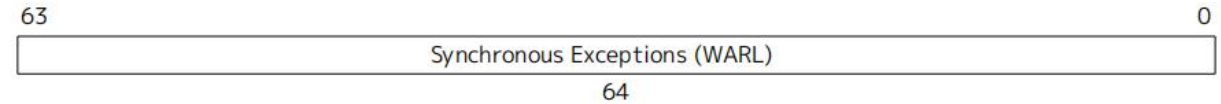


图18.3监管异常委托寄存器（hedeleg）

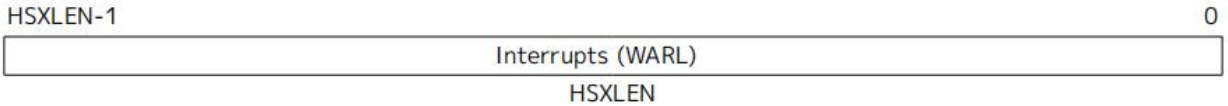


图18.4监管异常委托寄存器（hideleg）

已委托给 HS 模式的同步陷阱（使用 medeleg）如果陷阱发生前 V=1 且相应的 hedeleg 位被设置，则进一步委托给 VS 模式。hedeleg 的每一位应为可写或只读 0。hedeleg 的许多位被特别要求为可写或 0，如 hedeleg-bits 中所列举。位 0 对应于指令地址未对齐异常，如果 IALIGN=32，则必须为可写。

要求hedeleg的某些位为可写，减轻了超级用户处理实现变体的部分负担。

当 XLEN=32 时，hedelegh 是一个 32 位的读写寄存器，它是 hedeleg 的 63:32 位的别名。
当 XLEN=64 时，寄存器 hedelegh 不存在。已委托给 HS 模式的中断（使用 mideleg）如果相应的 hideleg 位被设置，则进一步委托给 VS 模式。在 hideleg 的 15:0 位中，位 10、6 和 2（对应于标准的 VS 级别中断）是可写的，而位 12、9、5 和 1（对应于标准的 S 级别中断）是只读 0。

当虚拟超级用户外部中断（代码 10）被委托给 VS 模式时，机器会自动将其转换为 VS 模式的超级用户外部中断（代码 9），包括在中断陷阱时写入 vscause 的值。同样，虚拟超级用户定时器中断（6）被转换为 VS 模式的超级用户定时器中断（5），虚拟超级用户软件中断（2）被转换为 VS 模式的超级用户软件中断（1）。对于平台中断原因（代码 16 及以上），可能会或可能不会进行类似的转换，如表 1.2。

表 18.2 必须可写或必须只读为 0 的 hedeleg 位

比特	属性	相应的异常
0	(See text)	指令地址未对齐
1	可写入	指令存取故障
2	可写入	非法指令
3	可写入	断点
4	可写入	负载地址错位
5	可写入	负载接入故障
6	可写入	仓库/AMO 地址未对齐
7	可写入	存储/AMO 访问故障
8	只读 0	u 模式或 vu 模式下的环境调用
9	只读 0	从 hs 模式调用环境
10	只读 0	从 vs 模式进行环境调用
11	可写入	从 m 模式调用环境
12	可写入	说明页故障
13	可写入	加载页面故障
15	只读 0	Store/AMO 页面错误
16	可写入	双重陷阱
18	可写入	软件检查
19	只读 0	硬件错误
20	只读 0	说明来宾页面错误
21	只读 0	加载访客页面错误
22	只读 0	虚拟指令
23		商店/AMO 用户页面错误

18.2.3 管理程序中断（“hvip“、“hip“和“hie”）寄存器

寄存器 hvip 是一个 HSXLEN 位的读写寄存器，超级用户可以写入该寄存器以指示针对 VS 模式的虚拟中断。hvip 中不可写的位为只读 0。

hvip 的标准部分（位 15:0）的格式如 hvipreg-standard 所示。hvip 的 VSEIP、VSTIP 和 VSSIP 位是可写的。在 hvip 中设置 VSEIP=1 会触发 VS 级别外部中断；设置 VSTIP 会触发 VS 级别定时器中断；设置 VSSIP 会触发 VS 级别软件中断。

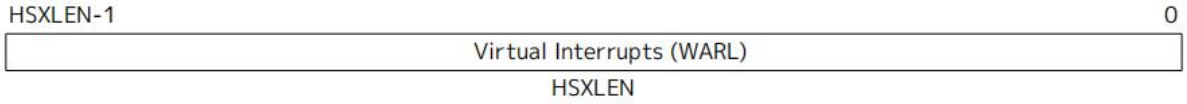


图18.5 Hypervisor 虚拟中断待处理寄存器

寄存器 hip 和 hie 是 HSXLEN 位的读写寄存器，分别补充 HS 级别的 sip 和 sie。hip 寄存器指示挂起的 VS 级别和超级用户特定的中断，而 hie 包含相同中断的使能位。对于 sie 中的每个可写位，hip 和 hie 中的相应位应为只读 0。因此，sie 和 hie 中的非 0 位始终互斥，sip 和 hip 也是如此。

hip和hie的有效位不能放置在HS级别的sip和sie中，因为这样做会使软件无法在未实现硬件超级用户扩展的平台上模拟该扩展。

当以下所有条件为真时，中断 i 将陷入 HS 模式：(a) 当前操作模式为 HS 模式且 sstatus 寄存器中的 SIE 位被设置，或当前操作模式的特权低于 HS 模式；(b) 位 i 在 sip 和 sie 中均被设置，或在 hip 和 hie 中均被设置；以及(c) 位 i 未在 hideleg 中设置。

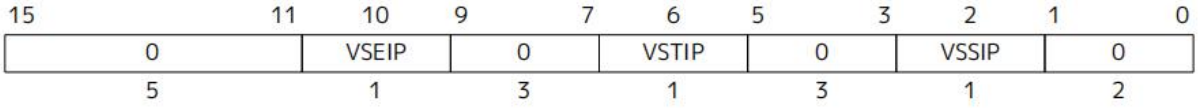


图18.6 Hypervisor 虚拟中断待处理寄存器

如果 sie 的位 i 为只读 0，则寄存器 hip 中的相同位可以是可写的或只读的。当 hip 中的位 i 为可写时，可以通过向该位写入 0 来清除挂起的中断 i。如果中断 i 可以在 hip 中挂起但 hip 中的位 i 为只读，则可以通过清除 hvip 的位 i 来清除中断，或者实现必须提供其他机制来清除挂起的中断（这可能涉及调用执行环境）。

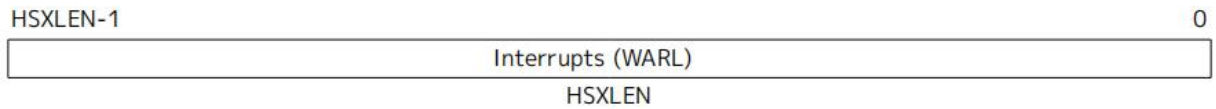


图18.7 Hypervisor中断待处理寄存器 (hip)

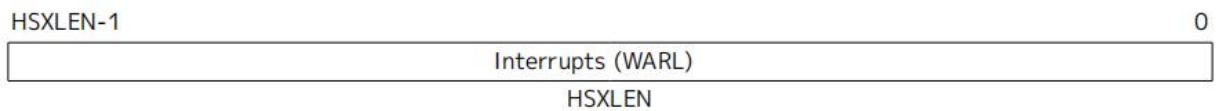


图18.8 Hypervisor中断待处理寄存器 (hie)

如果相应的中断可以在 hip 中挂起，则 hie 中的位应为可写的。hie 中不可写的位应为只读 0。寄存器 hip 和 hie 的标准部分（位 15:0）的格式分别如图 18.9 和 18.10 所示。位 hip.SGEIP 和 hie.SGEIE 是超级用户级别（HS 级别）的客体外中断的中断挂起和中断使能位。SGEIP 在 hip 中是只读的，当且仅当 CSR hgeip 和 hgeie 的按位逻辑与在任何位中非 0 时，SGEIP 为 1。（参见 hgeinterruptregs。）

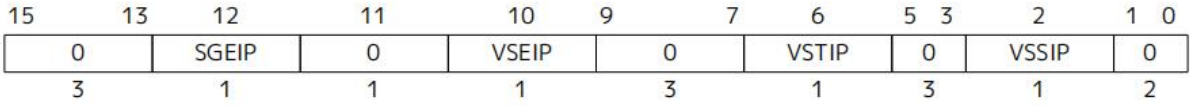


图18.9 hip寄存器的标准部分（位15:0）

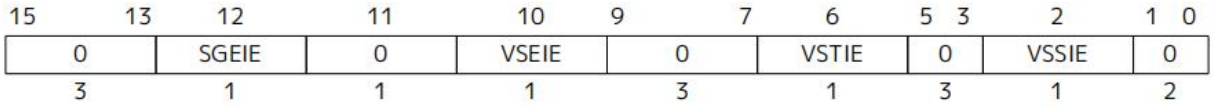


图18.10 hie寄存器的标准部分（位15:0）

位 hip.VSEIP 和 hie.VSEIE 是 VS 级别外部中断的中断挂起和中断使能位。VSEIP 在 hip 中是只读的，并且是以下中断源的逻辑：

位 hip.VSTIP 和 hie.VSTIE 是 VS 级别定时器中断的中断挂起和中断使能位。VSTIP 在 hip 中是只读的，并且是 hvip.VSTIP 和任何其他定向到 VS 级别的平台特定定时器中断信号的逻辑或。

位 hip.VSSIP 和 hie.VSSIE 是 VS 级别软件中断的中断挂起和中断使能位。hip 中的 VSSIP 是 hvip 中相同位的别名（可写）。多个同时发生且目标为 HS 模式的中断按以下优先级递减顺序处理：SEI、SSI、STI、SGEI、VSEI、VSSI、VSTI。

18.2.4 管理程序访客外部中断寄存器（“hgeip“和“hgeie”）

hgeip 寄存器是一个 HSXLEN 位的只读寄存器，其格式如 hgeipreg 所示，用于指示此硬件线程挂起的客体外中断。hgeie 寄存器是一个 HSXLEN 位的读写寄存器，其格式如 hgeiereg 所示，包含此硬件线程客体外中断的使能位。客体外中断号 i 对应于 hgeip 和 hgeie 中的位 i。

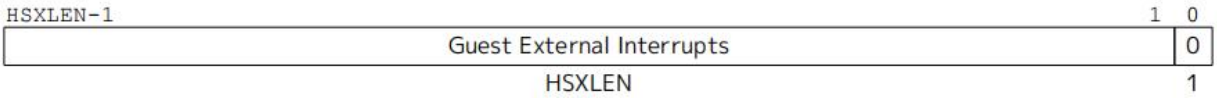


图18.11 Hypervisor客户外部中断待处理寄存器（hgeip）

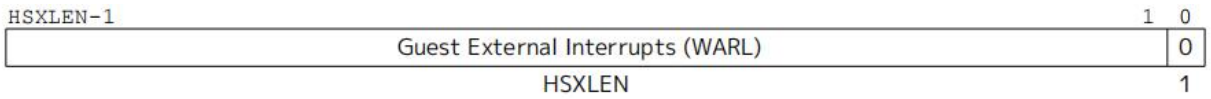


图18.12 Hypervisor客户外部中断待处理寄存器（hgeie）

客体外中断表示定向到 VS 级别单个虚拟机的中断。如果 RISC-V 平台支持将物理设备直接置于客户操作系统的控制下，且超级用户干预最少（称为虚拟机与物理设备之间的直通或直接分配），那么在这种情况下，来自设备的中断是针对特定虚拟机的。hgeip 的每一位汇总了定向到一个虚拟硬件线程的所有挂起中断，这些中断由中断控制器收集和报告。为了区分来自多个设备的特定挂起中断，软件必须查询中断控制器。

支持客体外中断需要一个能够将虚拟机定向的中断与其他中断分开收集的中断控制器。

hgeip 和 hgeie 中实现的客体外中断位数未指定，可能为 0。该数字称为 GEILEN。最低有效位首先实现，除了位 0。因此，如果 GEILEN 不为 0，则位 GEILEN:1 在 hgeie 中应为可写的，并且所有其他位在 hgeip 和 hgeie 中都应为只读 0。

在一个物理硬件线程上接收和处理的客体外中断集可能与其他硬件线程上接收的中断集不同。一个物理硬件线程上的客体外中断号i通常预期与任何其他硬件线程上的客体外中断i不同。对于任何一个物理硬件线程，可以直接接收客体外中断的虚拟硬件线程的最大数量受GEILEN限制。对于任何实现，每个物理硬件线程的最大数量在RV32中为31，在RV64中为63。

超级用户始终可以自由地为任意数量的虚拟硬件线程模拟设备，而不受GEILEN的限制。只有中断的直接直通（直接分配）受GEILEN限制，并且该限制是针对接收此类中断的虚拟硬件线程数量，而不是接收的不同中断数量。单个虚拟硬件线程可以接收的不同中断数量由中断控制器决定。

寄存器 hgeie 选择导致超级用户级别（HS 级别）客体外中断的客体外中断子集。hgeie 中的使能位不影响由 hstatus.VGEIN 从 hgeip 中选择的 VS 级别外部中断信号。

henvcfg CSR 是一个 64 位的读写寄存器，其格式如 henvcfg 所示，用于在虚拟化模式 V=1 时控制执行环境的某些特性，如下图 18.13。

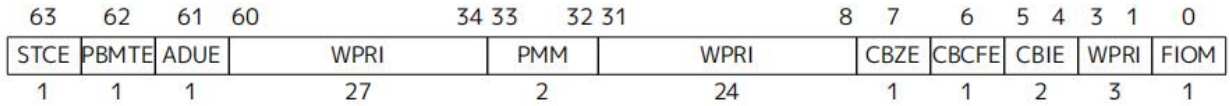


图18.13 Hypervisor环境配置寄存器 (henvcfg)

如果 henvcfg 中的 FIOM 位（I/O 的 Fence 隐含内存）设置为 1，则在 V=1 时执行的 FENCE 指令会被修改，使得对设备 I/O 访问的排序要求也隐含对主内存访问的排序要求。henvcfg-FIOM 详细说明了当 FIOM=1 且 V=1 时，FENCE 指令的 PI、PO、SI 和 SO 位的修改解释。

类似地，当 FIOM=1 且 V=1 时，如果访问被排序为设备 I/O 区域的原子指令设置了其 aq 和/或 rl 位，则该指令的排序方式就像它同时访问设备 I/O 和内存一样。

当 FIOM=1 和虚拟化模式 V=1 时，对 FENCE 前导集和后继集的解释进行了修改，表 1.3。

表 18.3 指令为含义

指令位	设定时的含义
PI	前置设备输入和内存读取（隐含 PR）
PO	前导设备输出和内存写入（隐含 PW）
SI	后继设备输入和存储器读取（隐含 SR）
SO	后继设备输出和内存写入（SW 隐含）

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，实现的行为就像 Svpbmt 未在 VS 级地址转换中实现一样。如果未实现

Svpbmt，则 PBMTE 为只读 0。

如果实现了 Svadu 扩展，ADUE 位控制是否在 VS 级地址转换期间启用 PTE A/D 位的硬件更新。当 ADUE=1 时，在 VS 级地址转换期间启用 PTE A/D 位的硬件更新，并且实现的行为就像 Svade 扩展未在 VS 模式地址转换中实现一样。当 ADUE=0 时，实现的行为就像 Svade 在 VS 级地址转换中实现一样。如果未实现 Svadu，则 ADUE 为只读 0。STCE 字段的定义由 Sstc 扩展提供。Zicboz 扩展提供了 CBZE 字段的定义。CBCFE 和 CBIE 字段的定义由 Zicbom 扩展提供。

PMM 字段的定义由 Ssnpm 扩展提供。

Zicfilp 扩展在 henvcfg 中添加了 LPE 字段。当 LPE 字段设置为 1 时，Zicfilp 扩展在 VS 模式下启用。当 LPE 字段为 0 时，Zicfilp 扩展在 VS 模式下未启用，并且以下规则适用于 VS 模式：

硬件线程不会更新 ELP 状态；它保持为 NOLPEXPECTED。LPAD 指令的操作等同于空操作（no-op）

Zicfiss 扩展在 henvcfg 中添加了 SSE 字段。如果 SSE 字段设置为 1，则 Zicfiss 扩展在 VS 模式下激活。当 SSE 字段为 0 时，Zicfiss 扩展在 VS 模式下保持未激活状态，并且在 V=1 时适用以下规则：

32 位 Zicfiss 指令将恢复为 Zimop 定义的行为。16 位 Zicfiss 指令将恢复为 Zcmop 定义的行为。VS 级页表中的 pte.xwr=010b 编码变为保留。senvcfg.SSE 字段将读取为 0 并且是只读的。

当 menvcfg.SSE 为 1 时，SSAMOSWAP.W/D 会引发虚拟指令异常。Ssdbltrp 扩展在 henvcfg 中添加了双陷阱使能（DTE）字段。当 henvcfg.DTE 为 0 时，实现的行为就像 Ssdbltrp 未在 VS 模式下实现一样，并且 vsstatus.SDT 位为只读 0。

当 XLEN=32 时，henvcfgh 是一个 32 位的读写寄存器，它是 henvcfg 的 63:32 位的别名。当 XLEN=64 时，寄存器 henvcfgh 不存在。

18.2.5 管理程序计数器启用（“hcountren”）寄存器

计数器使能寄存器 hcounteren 是一个 32 位寄存器，用于控制硬件性能监控计数器对客户虚拟机的可用性。

当 hcounteren 寄存器中的 CY、TM、IR 或 HPMn 位清 0 时，如果在 V=1 时尝试读取 cycle、time、instret 或 hpmcounter n 寄存器，且 mcounteren 中的相同位为 1，则会导致虚拟指令异常。当这些位中的某一位被设置时，除非因其他原因被阻止，否则在 V=1 时允许访问相应的寄存器。在 VU 模式下，除非在 hcounteren 和 scounteren 中都设置了适用的位，否则计数器不可读。然而，任何位都可以是只读 0，表示在 V=1 时读取相应的计数器将导致异常。因此，它们实际上是 WARL 字段。

htimedelta CSR 是一个 64 位的读写寄存器，包含 time CSR 的值与在 VS 模式或 VU 模式下返回的值之间的差值。也就是说，在 VS 或 VU 模式下读取 time CSR 会返回 htimedelta 的内容与 time 的实际值之和。

由于在将 htimedelta 和 time 相加时忽略溢出，因此可以使用较大的 htimedelta 值来表示负时间偏移。

当 XLEN=32 时，htimedeltah 是一个 32 位的读写寄存器，它是 htimedelta 的 63:32 位的别名。当 XLEN=64 时，寄存器 htimedeltah 不存在。如果实现了 time CSR，则必须实现 htimedelta（对于 XLEN=32，还需实现 htimedeltah）。

18.2.6 管理程序陷阱值（“htval”）寄存器

htval 寄存器是一个 HSXLEN 位的读写寄存器，其格式如图 18.14 所示。当发生陷阱进入 HS 模式

时，htval 会与 stval 一起写入额外的异常特定信息，以帮助软件处理陷阱。

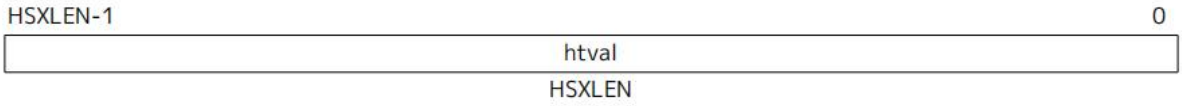


图18.14 Hypervisor 陷阱值寄存器 (htval)

当客户页错误陷阱进入 HS 模式时，htval 会被写入 0 或出错的客户物理地址，并右移 2 位。对于其他陷阱，htval 被设置为 0，但未来的标准或扩展可能会重新定义 htval 对其他陷阱的设置。

客户页错误可能由于第一阶段（VS 阶段）地址转换期间的隐式内存访问而产生，在这种情况下，写入 htval 的客户物理地址是出错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 阶段转换未能完成时，与原始虚拟地址对应的客户物理地址未知。）CSR htinst 中提供了额外的信息以消除此类情况的歧义。

否则，对于导致客户页错误的未对齐加载和存储，htval 中的非 0 客户物理地址对应于 stval 中虚拟地址指示的访问出错部分。对于具有可变长度指令的系统上的指令客户页错误，非 0htval 对应于 stval 中虚拟地址指示的指令出错部分。

写入htval的客户物理地址右移2位，以适应比当前XLEN更宽的地址。对于RV32，超级用户扩展允许客户物理地址宽达34位，htval报告地址的33:2位。这种客户物理地址的右移2位编码与PMP地址寄存器（pmp）和页表项（sv32、sv39、sv48和sv57）中的物理地址编码相匹配。

如果需要出错客户物理地址的最低两位，这些位通常与stval中出错虚拟地址的最低两位相同。对于由于VS阶段地址转换的隐式内存访问导致的错误，最低两位为0。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，必须能够保存 0，并且可能只能保存其他 2 位移位客户物理地址的任意子集（如果有的话）。

除非有理由假设其他情况（例如平台标准），否则向htval写入值的软件应从htval读取以确认存储的值。

18.2.7 管理程序陷阱指令（htinst）寄存器

htinst 寄存器是一个 HSXLEN 位的读写寄存器，其格式如 htinstreg 所示。当发生陷阱进入 HS 模式时，htinst 会被写入一个值，如果该值非 0，则提供有关触发陷阱的指令的信息，以帮助软件处理陷阱。tinst-vals 中记录了在陷阱时可能写入 htinst 的值。

htinst 是一个 WARL 寄存器，只需要能够保存在陷阱时实现可能自动写入的值。

18.2.8 管理程序访客地址转换和保护（hgatp）寄存器

hgatp 寄存器是一个 HSXLEN 位的读写寄存器，其格式在 HSXLEN=32 时如图 18.15 所示，在 HSXLEN=64 时如 rv64hgatp 所示，它控制 G 级地址转换和保护，即客户虚拟地址的两级转换的第二阶段（见 two-stage-translation）。与 CSR satp 类似，该寄存器保存客户物理根页表的物理页号（PPN）；一个虚拟机标识符（VMID），用于在每个虚拟机的层面实现地址转换隔离；以及 MODE 字段，用于选择客户物理地址的地址转换方案。当 mstatus.TVM=1 时，在 HS 模式下执行时尝试读取或写入 hgatp 将引发

非法指令异常。

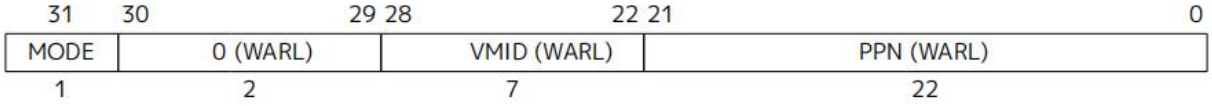


图18.15 HSXLEN=32 时的虚拟机监视器客户地址转换与保护寄存器 hgatp

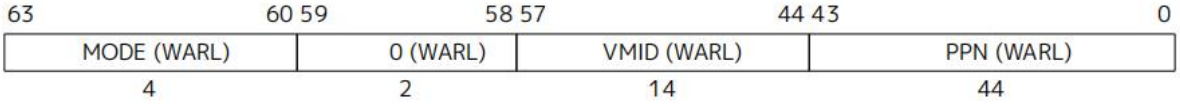


图18.16 HSXLEN=64 时的虚拟机监视器客户地址转换与保护寄存器 hgatp（支持 Bare、Sv39x4 和 Sv57x4 模式）

当 HSXLEN=32 时，管理程序访客地址转换和保护寄存器“hgatp”。

当 HSXLEN=64 时，对于 MODE 值 Bare、Sv39x4 和 Sv57x4，管理程序访客地址转换和保护寄存器“hgatp”。

hgatp 模式 显示了当 HSXLEN=32 和 HSXLEN=64 时 MODE 字段的编码。当 MODE=Bare 时，客户物理地址等于超级用户物理地址，并且除了 pmp 中描述的物理内存保护方案外，没有对客户虚拟机的进一步内存保护。在这种情况下，hgatp 中的其余字段必须设置为 0。

当 HSXLEN=32 时，MODE 的唯一其他有效设置是 Sv32x4，这是对通常的 Sv32 分页虚拟内存方案的修改，扩展为支持 34 位客户物理地址。当 HSXLEN=64 时，模式 Sv39x4、Sv48x4 和 Sv57x4 被定义为对 Sv39、Sv48 和 Sv57 分页虚拟内存方案的修改。所有这些分页虚拟内存方案在 guest-addr-translation 中描述。

当 HSXLEN=64 时，其余的 MODE 设置保留供将来使用，并且可能定义 hgatp 中其他字段的不同解释。

表 18.4 hgatp MODE 字段的编码

类别	值	名称	描述
HSXLEN=32	0	Bare	没有翻译或保护。
	1	Sv32x4	基于页面的 34 位虚拟寻址（Sv32 的 2 位扩展）。
HSXLEN=64	0	Bare	没有翻译或保护
	1-7	—	保留
	8	Sv39x4	基于页面的 41 位虚拟寻址（Sv39 的 2 位扩展）
	9	Sv48x4	基于页面的 50 位虚拟寻址（Sv48 的 2 位扩展）
	10	Sv57x4	基于页面的 59 位虚拟寻址（Sv57 的 2 位扩展）。
	11-15	—	保留

当 HSXLEN=64 时，实现不需要支持所有定义的 MODE 设置。使用不支持的 MODE 值写入 hgatp 不会像 satp 那样被忽略。相反，hgatp 的字段在正常情况下是 WARL 的，如所示。

如 guest-addr-translation 中所述，对于分页虚拟内存方案（Sv32x4、Sv39x4、Sv48x4 和 Sv57x4），根页表为 16 KiB，并且必须对齐到 16 KiB 边界。在这些模式下，hgatp 中物理页号（PPN）的最低两位始终读取为 0。仅支持定义的分页虚拟内存方案和/或 Bare 的实现可以使 PPN[1:0] 为只读 0。

VMID 位的数量未指定，可能为 0。实现的 VMID 位数，称为 VMIDLEN，可以通过向 VMID 字段的每个位位置写入 1，然后读取 hgatp 中的值来确定 VMID 字段中哪些位位置保持为 1。VMID 的最低有效位首先实现：也就是说，如果 VMIDLEN＞0，则 VMID[VMIDLEN-1:0] 是可写的。VMIDLEN 的最大值，称为 VMIDMAX，对于 Sv32x4 为 7，对于 Sv39x4、Sv48x4 和 Sv57x4 为 14。

除非有效特权模式为 U 且 hstatus.HU=0，否则 hgatp 寄存器在地址转换算法中被视为活动的。

此定义简化了 HLV、HLVX 和 HSV 指令的推测执行的实现。

请注意，写入 hgatp 并不意味着页表更新和后续 G 级地址转换之间存在任何排序约束。如果新虚拟机的客户物理页表已被修改，或者如果 VMID 被重用，则可能需要在写入 hgatp 之前或之后执行 HFENCE.GVMA 指令（参见 hfence.vma）。

18.2.9 虚拟主管状态（“vsstatus”）注册

vsstatus 寄存器是一个 VSXLEN 位的读写寄存器，它是 VS 模式下的超级用户寄存器 sstatus 的版本，其格式在 VSXLEN=32 时如 vsstatusreg-rv32 所示，在 VSXLEN=64 时如 vsstatusreg 所示。当 V=1 时，vsstatus 替代通常的 sstatus，因此通常读取或修改 sstatus 的指令实际上访问的是 vsstatus。

当 VSXLEN=32 时，虚拟监控器状态（“vsstatus”）会被注册。

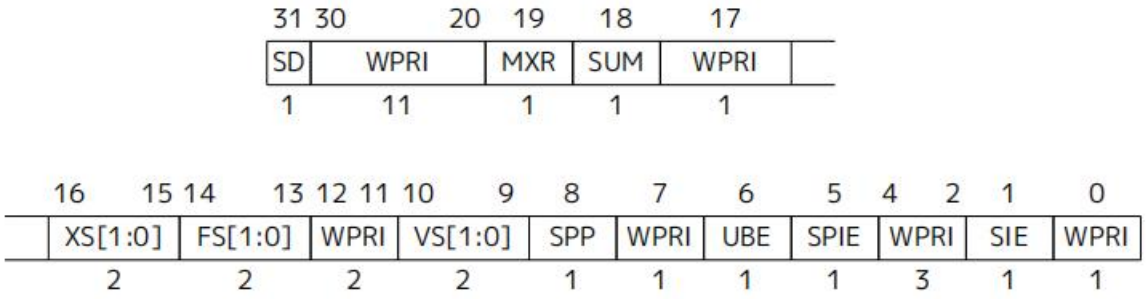


图18.17 VSXLEN=32 时的虚拟监督者状态寄存器 (vsstatus)

当 VSXLEN=64 时，虚拟监控器状态（“vsstatus”）会注册。

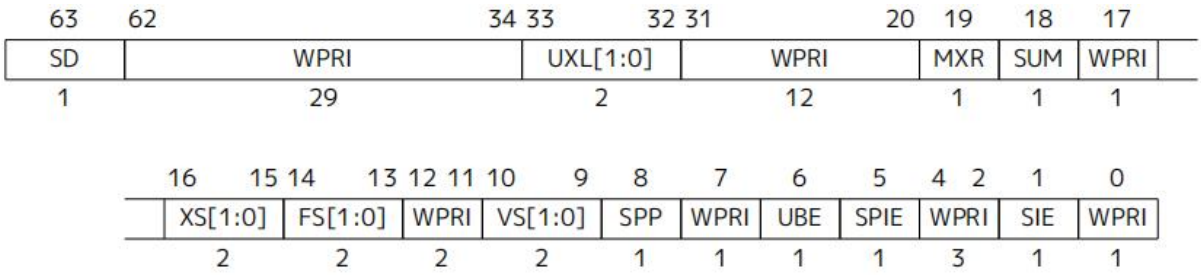


图18.18 VSXLEN=64时的虚拟监督者状态寄存器 (vsstatus)

UXL 字段控制 VU 模式的有效 XLEN，它可能与 VS 模式的 XLEN（VSXLEN）不同。当 VSXLEN=32 时，UXL 字段不存在，VU 模式 XLEN=32。当 VSXLEN=64 时，UXL 是一个 WARL 字段，其编码与 misa 的 MXL 字段相同，参见 misabase 。具体而言，实现可以使 UXL 成为 hstatus 的 VSXL 字段的只读副本，强

制 VU 模式 XLEN=VSXLEN。

如果 VSXLEN 从 32 更改为更宽的宽度，并且如果字段 UXL 未被限制为单一值，则将获得对应于不超过新 VSXLEN 的最宽支持宽度的值。

当 V=1 时，vsstatus.FS 和 HS 级别的 sstatus.FS 都有效。当任一字段为 0（关闭）时尝试执行浮点指令会引发非法指令异常。当 V=1 时修改浮点状态会导致两个字段都设置为 3（Dirty）。

为了使超级用户从扩展上下文状态中受益，必须在 HS 级别的 sstatus 中拥有自己的副本，独立于在 VS 模式下运行的客户操作系统进行维护。虽然 vsstatus 中显然必须存在 VS 模式的扩展上下文状态版本，但由于 VS 级别软件可以任意更改 vsstatus.FS，超级用户不能依赖此版本的正确维护。如果 HS 级别的 sstatus.FS 在 V=1 时没有独立激活并由硬件与 vsstatus.FS 并行维护，超级用户在虚拟机之间进行上下文切换时将始终被迫保守地交换所有浮点状态。

类似地，当 V=1 时，vsstatus.VS 和 HS 级别的 sstatus.VS 都有效。当任一字段为 0（关闭）时尝试执行向量指令会引发非法指令异常。当 V=1 时修改向量状态会导致两个字段都设置为 3（Dirty）。

只读字段 SD 和 XS 总结了仅对 VS 模式可见的扩展上下文状态。例如,HS 级别的 sstatus.FS 的值不会影响 vsstatus.SD。实现可以使字段 UBE 成为 hstatus.VSBE 的只读副本。

当 V=0 时，vsstatus 不会直接影响机器的行为，除非使用虚拟机加载/存储（HLV、HLVX 或 HSV）或 mstatus 寄存器中的 MPRV 功能来执行加载或存储 仿佛 V=1。

Zicfilp 扩展添加了 SPELP 字段，该字段保存先前的 ELP，并按照 ZICFILPFORWARDTRAPS 中的规定进行更新。SPELP 字段的编码如下：

0 - NOLPEXPECTED - 不期望出现着陆垫指令。
1 - LPEXPECTED - 期望出现着陆垫指令。

Ssdbltrp 添加了一个 S 模式禁用陷阱（SDT）字段扩展，以解决 VS 模式中的双陷阱问题（参见 supv-double-trap）。

18.2.10 虚拟监控器中断（“vsip“和“vsie”）寄存器

vsip 和 vsie 寄存器是 VSXLEN 位的读写寄存器，它们是 VS 模式下超级用户 CSR sip 和 sie 的版本,其格式分别参见 图 18.19 和 图 18.20 所示。当 V=1 时,vsip 和 vsie 替代了通常的 sip 和 sie，因此通常读取或修改 sip/sie 的指令实际上访问的是 vsip/vsie。然而，当 V=1 时，定向到 HS 级别的中断继续在 HS 级别的 sip 寄存器中指示，而不是在 vsip 中。

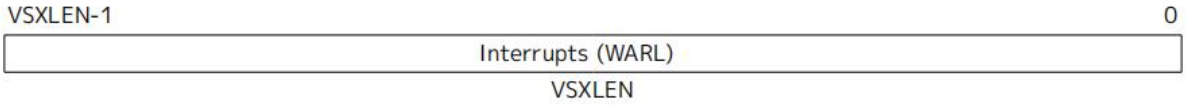


图18.19 虚拟监督者中断待处理寄存器 (vsip)

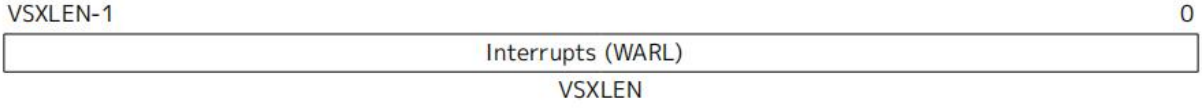


图18.20 虚拟监督者中断待处理寄存器 (vsie)

寄存器 vsip 和 vsie 的标准部分（位 15:0）的格式分别如图 18.21 和图 18.22 所示。

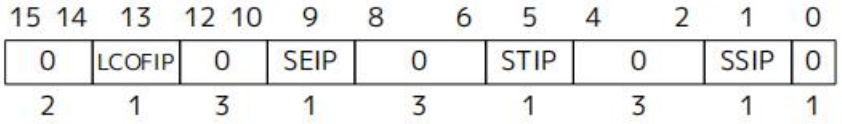


图18.21 vsip 寄存器的标准部分（位 15:0）

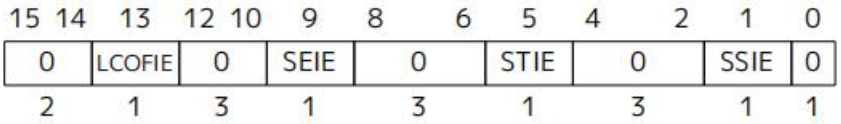


图18.22 vsie 寄存器的标准部分（位 15:0）

扩展 Shlcofideleg 支持将 LCOFI 中断委托给 VS 模式。如果实现了 Shlcofideleg 扩展，hideleg 的位 13 是可写的；否则，它是只读 0。当 hideleg 的位 13 为 0 时，vsip.LCOFIP 和 vsie.LCOFIE 是只读 0。否则，vsip.LCOFIP 和 vsie.LCOFIE 是 sip.LCOFIP 和 sie.LCOFIE 的别名。

当 hideleg 的位 10 为 0 时，vsip.SEIP 和 vsie.SEIE 是只读 0。否则，vsip.SEIP 和 vsie.SEIE 是 hip.VSEIP 和 hie.VSEIE 的别名。

当 hideleg 的位 6 为 0 时，vsip.STIP 和 vsie.STIE 是只读 0。否则，vsip.STIP 和 vsie.STIE 是 hip.VSTIP 和 hie.VSTIE 的别名。

当 hideleg 的位 2 为 0 时，vsip.SSIP 和 vsie.SSIE 是只读 0。否则，vsip.SSIP 和 vsie.SSIE 是 hip.VSSIP 和 hie.VSSIE 的别名。

18.2.11 虚拟监控器陷阱向量基址（vstvec）寄存器

vstvec 寄存器是一个 VSXLEN 位的读写寄存器，它是 VS 模式下的 stvec 寄存器的版本，其格式如图 18.23 所示。当 V=1 时，vstvec 会替代通常的 stvec，因此通常读取或修改 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18.23 虚拟监督者陷阱向量基地址寄存器 (vstvec)

18.2.12 虚拟监管者暂存（vsscratch）寄存器

vsscratch 寄存器是一个 VSXLEN 位的读写寄存器，它是 VS 模式下的 sscratch 寄存器的版本，其格式参见图 18.24 所示。当 V=1 时，vsscratch 会替代通常的 sscratch，因此通常读取或修改 sscratch 的指令实际上会访问 vsscratch。vsscratch 寄存器的内容永远不会直接影响机器的行为。

图18.24 虚拟管理员scratch寄存器vsscratch

18.2.13 虚拟监管者异常程序计数器（vsepc）寄存器

vsepc 寄存器是一个 VSXLEN 位的读写寄存器，它是 VS 模式下的 sepc 寄存器的版本，其格式参图 18.25 。当 V=1 时，vsepc 会替代通常的 sepc，因此通常读取或修改 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（Write-Any-Read-Legal，写任意值读合法值）寄存器，它必须能够存储与 sepc 寄存器相同的值集。

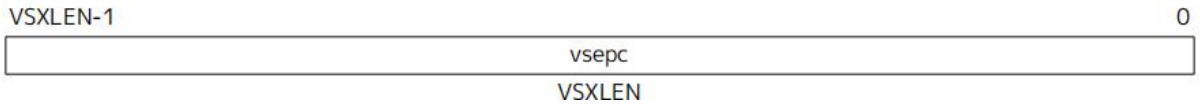


图18.25 虚拟监督者异常程序计数器（vsepc）

18.2.14 虚拟监管者异常原因（vscause）寄存器

vscause 寄存器是一个 VSXLEN 位的读写寄存器，它是 VS 模式下的 scause 寄存器的版本，其格式如 图 18.26 所示。当 V=1 时，vscause 会替代通常的 scause，因此通常读取或修改 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 WLRL（Write-Legal-Read-Legal，写合法值读合法值）寄存器，它必须能够存储与 scause 寄存器相同的值集。

图18.26 虚拟管理员原因寄存器（vscause）

18.2.15 虚拟监管者陷阱值（vstval）寄存器

vstval 寄存器是一个 VSXLEN 位的读写寄存器，它是 VS 模式下的 stval 寄存器的版本，其格式如图 18.28 所示。当 V=1 时，vstval 会替代通常的 stval，因此通常读取或修改 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（Write-Any-Read-Legal，写任意值读合法值）寄存器，它必须能够存储与 stval 寄存器相同的值集。

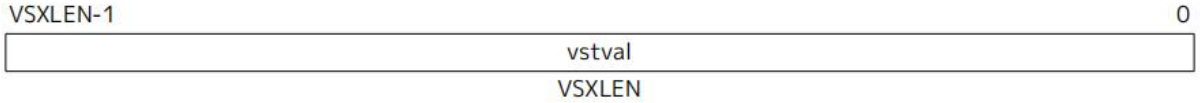


图18.27 虚拟管理器陷阱值寄存器（vstval）

18.2.16 虚拟监管者地址转换与保护（vsatp）寄存器

vsatp 寄存器是一个 VSXLEN 位的读写寄存器，它是 VS 模式下的 satp 寄存器的版本，其格式在 VSXLEN=32 时如 rv32vsatpreg 所示，在 VSXLEN=64 时如 rv64vsatpreg 所示。当 V=1 时，vsatp 会替代通常的 satp，因此通常读取或修改 satp 的指令实际上会访问 vsatp。vsatp 控制 VS 阶段的地址转换，这是客户虚拟地址两级转换的第一阶段（参见 two-stage-translation）。

虚拟监管者地址转换与保护 vsatp 寄存器（当 VSXLEN=32 时）。

虚拟监管者地址转换与保护 vsatp 寄存器（当 VSXLEN=64 时）。

vsatp 寄存器在地址转换算法中被视为 有效，除非 有效特权模式为 U 且 hstatus.HU=0。然而，

即使 vsatp 有效，VS 阶段的页表项的 A 位（访问位）也不得因推测执行而被设置，除非有效特权模式为 VS 或 VU。

具体而言，错误推测执行的虚拟机加载/存储指令（HLV、HLVX 或 HSV）不得导致 VS 阶段的 A 位（访问位）被设置。

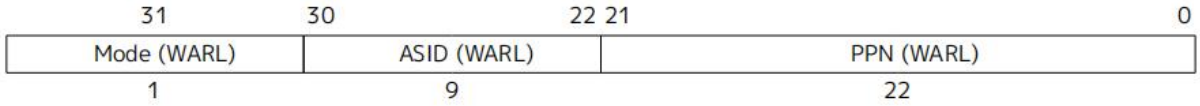


图18.28 VSXLEN=32时的虚拟监督者地址转换与保护寄存器 （vsatp）

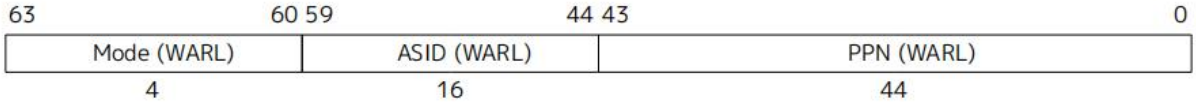


图18.29 VSXLEN=64时的虚拟监督者地址转换与保护寄存器 （vstval）

18.3 管理程序说明

虚拟机扩展增加了虚拟机加载和存储指令以及两条特权隔离指令。

虚拟机监视器的虚拟机加载与存储指令仅M模式或 HS 模式下有效,或在U模式下当 hstatus.HU=1 时有效。每条指令执行显式内存访问时，V=1；即使用适用于 VS 模式或 VU 模式的内存访问的地址转换与保护机制以及字节序。hstatus 的 SPVP 字段控制访问的特权级别。当 SPVP=0 时，显式内存访问在 VU 模式下进行；当 SPVP=1 时，仿佛在 VS 模式下进行。与通常 V=1 时一样，应用两级地址转换，并且忽略 HS 级别的 sstatus.SUM。HS 级别的 sstatus.MXR 使得仅可执行页面在地址转换的两个阶段（VS 阶段和 G 阶段）对显式加载可读，而 vsstatus.MXR 仅影响第一个转换阶段（VS 阶段）。

对于每条 RV32I 或 RV64I 加载指令（LB、LBU、LH、LHU、LW、LWU 和 LD），都有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。对于每条 RV32I 或 RV64I 存储指令（SB、SH、SW 和 SD），都有对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 在 RV32 中无效。

指令 HLVX.HU 和 HLVX.WU 与 HLV.HU 和 HLV.WU 相同，除了在地址转换期间用 执行 权限替代读取 权限。也就是说，被读取的内存必须在地址转换的两个阶段中都是可执行的，但不需要读取权限。对于地址转换后得到的监管者物理地址，监管者物理内存属性必须同时授予 执行 和 读取 权限。（监管者物理内存属性 是机器的物理内存属性，经过物理内存保护（pmp）在监管者级别修改后的结果。）

HLVX 指令无法覆盖机器级别的物理内存保护（PMP），因此尝试读取被 PMP 标记为仅可执行的内存仍然会导致访问故障异常。

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，而不是引发取指异常。

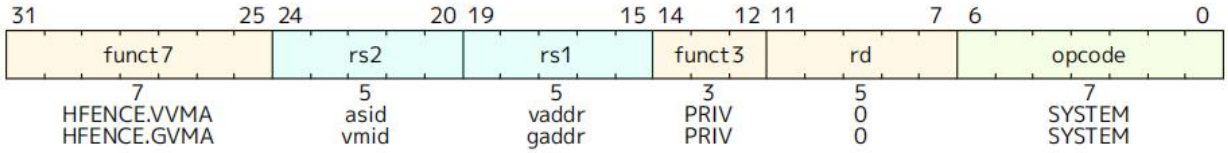
HLVX.WU 在 RV32 中有效，尽管 LWU 和 HLV.WU 在 RV32 中无效。（对于 RV32，HLVX.WU 可以视为 HLV.W 的变体，因为符号扩展对 32 位值无关紧要。）

当 V=1 时，尝试执行虚拟机加载/存储指令（HLV、HLVX 或 HSV）会导致虚拟指令异常。当

hstatus.HU=0 时，尝试从 U 模式执行这些指令会导致非法指令异常。

18.3.1 虚拟机监视器内存管理隔离指令

虚拟机监视器内存管理隔离指令 HFENCE.VVMA 和 HFENCE.GVMA 的功能类似于 SFENCE.VMA，但分别适用于由 CSR vsatp 控制的 VS 级别内存管理数据结构（HFENCE.VVMA）或由 CSR hgatp 控制的客户物理内存管理数据结构（HFENCE.GVMA）。指令 SFENCE.VMA 仅适用于由当前 satp 控制的内存管理数据结构（当 V=0 时为 HS 级别的 satp，当 V=1 时为 vsatp）。



HFENCE.VVMA 仅在 M 模式或 HS 模式下有效。其效果类似于暂时进入 VS 模式并执行 SFENCE.VMA。执行 HFENCE.VVMA 可以确保当前硬件线程之前的所有存储操作在该硬件线程为 VS 阶段地址转换执行的隐式读取之前完成，这些隐式读取用于满足以下条件的指令：

在 HFENCE.VVMA 之后执行，并且

在 hgatp.VMID 与 HFENCE.VVMA 执行时设置相同的情况下执行。

当 hgatp.VMID 与 HFENCE.VVMA 执行时不同时，隐式读取不需要排序。如果操作数 rs1≠x0，它指定一个客户虚拟地址；如果操作数 rs2≠x0，它指定一个客户地址空间标识符（ASID）。

HFENCE.VVMA 指令仅适用于由 HFENCE.VVMA 执行时 hgatp.VMID 的设置标识的单个虚拟机。

当 rs2≠x0 时，寄存器 rs2 中值的 XLEN-1:ASIDMAX 位保留供未来标准使用。在标准扩展定义其用途之前，软件应将其置 0，当前实现应忽略这些位。此外，如果 ASIDLEN < ASIDMAX，实现应忽略寄存器 rs2 中值的 ASIDMAX-1:ASIDLEN 位。

更简单的 HFENCE.VVMA 实现可以忽略 rs1 中的客户虚拟地址和 rs2 中的客户 ASID 值，以及 hgatp.VMID，并始终为所有虚拟机的 VS 级别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。

mstatus.TVM 和 hstatus.VTVM 都不会导致 HFENCE.VVMA 产生陷阱。

HFENCE.GVMA 仅在 mstatus.TVM=0 时的 HS 模式或 M 模式下有效(无论 mstatus.TVM 的值如何)。执行 HFENCE.GVMA 指令可以确保在当前 硬件线程 上已经可见的任何先前存储操作,都在该 硬件线程为后续指令的 G 阶段地址转换所做的所有隐式读取操作之前被排序。如果操作数 rs1≠x0，它指定一个客户物理地址，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：一个将客户虚拟地址映射到客户物理地址，另一个将客户物理地址映射到监管者物理地址。HFENCE.GVMA 不需要刷新前者缓存，但它必须刷新与 HFENCE.GVMA 的地址和 VMID 参数匹配的后者缓存中的条目。

更常见的情况是，实现中包含的地址转换缓存直接将客户虚拟地址映射到监管者物理地址，从而减少了一层间接性。对于这种实现，任何客户虚拟地址映射到与 HFENCE.GVMA 的地址和 VMID 参数匹配

的客户物理地址的条目都必须被刷新。以这种方式选择性刷新条目需要为它们标记客户物理地址，这是成本较高的操作，因此一种常见的技术是刷新与 HFENCE.GVMA 的 VMID 参数匹配的所有条目，而不管地址参数如何。

与在陷阱时写入 htval 的客户物理地址类似，在 rs1 中指定的客户物理地址会右移 2 位，以适应比当前 XLEN 更宽的地址。

当 rs2≠x0 时，rs2 中值的 XLEN-1:VMIDMAX 位保留供未来标准使用。在标准扩展定义其用途之前，软件应将这些位清 0，并且当前实现应忽略这些位。此外，如果 VMIDLEN < VMIDMAX，实现应忽略 rs2 中值的 VMIDMAX-1:VMIDLEN 位。

HFENCE.GVMA 的简单实现可以忽略 rs1 中的客户物理地址和 rs2 中的 VMID 值，并始终对所有虚拟机的客户物理内存管理执行全局屏障，甚至对所有内存管理数据结构执行全局屏障。

如果为给定的 VMID 更改了 hgatp.MODE，则必须执行一条 rs1=x0（且 rs2 设置为 x0 或 VMID）的 HFENCE.GVMA 指令，以确保后续的客户地址转换与 MODE 更改的顺序一致——即使旧的 MODE 或新的 MODE 是 Bare 模式。

当 V=1 时尝试执行 HFENCE.VVMA 或 HFENCE.GVMA 会导致虚拟指令异常，而在 U 模式下尝试执行这些指令则会导致非法指令异常。在 HS 模式下，当 mstatus.TVM=1 时尝试执行 HFENCE.GVMA 也会导致非法指令异常。

18.4 机器级控制状态寄存器

虚拟机监视器扩展向机器级 mstatus 或 mstatush 控制状态寄存器(CSR)添加了两个字段,MPV 和 GVA，并修改了多个现有 mstatus 字段的行为。hypervisor-mstatus 展示了在实现虚拟机监视器扩展且 MXLEN=64 时修改后的 mstatus 寄存器。当 MXLEN=32 时，虚拟机监视器扩展将 MPV 和 GVA 添加到 mstatush 而非 mstatus。hypervisor-mstatush 展示了在实现虚拟机监视器扩展且 MXLEN=32 时的 mstatush 寄存器。实现管理程序扩展时，为 RV64 注册机器状态（“mstatus”）。实现管理程序扩展时，为 RV32 注册其他机器状态（“mstatush”）。RV32 的“mstatus“格式不变。

MPV 位（机器先前虚拟化模式）在每次陷入 M 模式时由实现写入。类似于 MPP 字段在陷入时被设置为（名义上的）特权模式一样，MPV 位也被设置为陷入时的虚拟化模式 V 的值。当执行 MRET 指令时，虚拟化模式 V 被设置为 MPV，除非 MPP=3，在这种情况下 V 保持为 0。字段 GVA（客户虚拟地址）在每次陷入 M 模式时由实现写入。对于任何将客户虚拟地址写入 mtval 的陷入（断点、地址未对齐、访问故障、页面故障或客户页面故障),GVA 被设置为 1。对于任何其他陷入 M 模式的情况,GVA 被设置为 0。mstatus 的 TSR 和 TVM 字段仅在 HS 模式下影响执行，而不在 VS 模式下影响执行。TW 字段在所有模式（除 M 模式外）下影响执行。设置 TVM=1 会阻止 HS 模式访问 hgatp 或执行 HFENCE.GVMA 或 HINVAL.GVMA，但对访问 vsatp 或执行 HFENCE.VVMA 或 HINVAL.VVMA 没有影响。

TVM位存在于mstatus中，是为了允许机器级软件修改由监管级操作系统（OS）管理的地址转换，通常是为了在OS控制的地址转换之下插入另一级地址转换。TVM=1启用的指令陷入允许机器级软件同时接管satp和hgatp，并用影子页表替换OS选择的页表转换与机器级低阶段转换的合并结果，而这一切都在OS不知情的情况下完成。机器级软件需要这种能力，不仅是为了在不支持虚拟化管理器扩展的情况下模拟它，还为了模拟未来可能修改或增加地址转换阶段的RISC-V扩展，例如，可能是为了改进对嵌套虚拟化管理器的支持，即在其他虚拟化管理器之上运行虚拟化管理器。

然而，设置TVM=1并不会导致对vsatp的访问或执行HFENCE.VVMA或HINVAL.VVMA指令的陷入，也不会对在VS模式下采取的任何操作触发陷入，因为机器级软件不需要介入VS阶段的地址转换。对于虚拟机来说，保持VS阶段的地址转换不变，并将所有其他转换阶段合并到由hgatp控制的G阶段影子页表中，应该是足够的，并且在大多数情况下也会更快。这一假设确实对当前机器能够高效模拟的未来RISC-V扩展施加了一些限制。

虚拟化管理器扩展改变了 mstatus 中修改特权字段 MPRV 的行为。当 MPRV=0 时，地址转换和保护行为正常进行。当 MPRV=1 时，显式内存访问的地址转换、保护以及字节序处理会按照当前虚拟化模式设置为 MPV 且当前名义特权模式设置为 MPP 的方式执行。表 1.5 列举了这些情况。

表 18.5 MPRV 对外显记忆访问的翻译和保护的影响

MPRV	MPV	MPP	Effect
0	-	-	正常访问；当前特权模式适用
1	0	0	仅使用 HS 级地址转换和保护的 U 级访问
1	0	1	仅使用 HS 级地址转换和保护的 HS 级访问
1	-	3	无地址转换的 M 级访问
1	1	0	使用两级地址转换和保护的 VU 级访问。HS 级的 MXR 位使任何可执行页面可读。vsstatus.MXR 使在 VS 转换阶段标记为可执行的页面可读，但仅当在客户物理转换阶段可读时
1	1	1	使用两级地址转换和保护的 VS 级访问。HS 级的 MXR 位使任何可执行页面可读。vsstatus.MXR 使在 VS 转换阶段标记为可执行的页面可读，但仅当在客户物理转换阶段可读时。vsstatus.SUM 取代 HS 级的 SUM 位。

|1 |1 |0 |使用两级地址转换和保护的 VU 级访问。HS 级的 MXR 位使任何可执行页面可读。vsstatus.MXR 使在 VS 转换阶段标记为可执行的页面可读，但仅当在客户物理转换阶段可读时。

|1 |1 |1 |使用两级地址转换和保护的 VS 级访问。HS 级的 MXR 位使任何可执行页面可读。vsstatus.MXR 使在 VS 转换阶段标记为可执行的页面可读，但仅当在客户物理转换阶段可读时。vsstatus.SUM 取代 HS 级的 SUM 位。

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始终表现为 V=1 且名义特权模式为 hstatus.SPVP，覆盖 MPRV。mstatus 寄存器是 HS 级 sstatus 寄存器的超集，但不是 vsstatus 的超集。

18.4.1 机器中断委派（“mideleg”）寄存器

当实现了虚拟化管理器扩展时，mideleg 的第 10、6 和 2 位（对应于标准的 VS 级中断）均为只读的 1。此外，如果实现了任何客户外部中断（GEILEN 不为 0），mideleg 的第 12 位（对应于监管级客户外部中断）也是只读的 1。VS 级中断和客户外部中断始终被委托绕过 M 模式到 HS 模式。

对于 mideleg 中为 0 的位，hideleg、hip 和 hie 中的相应位为只读的 0。

18.4.2 机器中断（“mip“和“mie”）寄存器

虚拟机监视器扩展为寄存器 mip 和 mie 增加了额外的有效位，用于处理虚拟机监视器新增的中断。图 18.30 和 图 18.31 展示了在实现虚拟机监视器扩展时，寄存器 mip 和 mie 的标准部分（位 15:0）。

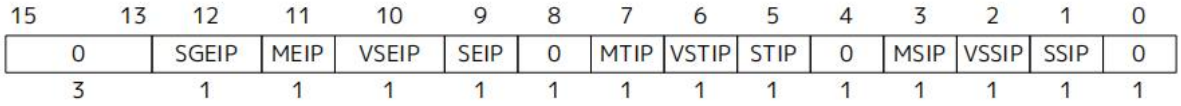


图18.30 mip 寄存器的标准部分（位 15:0）

图18.31 mie 寄存器的标准部分（位 15:0）

“mip”中的位 SGEIP、VSEIP、VSTIP 和 VSSIP 是相同的别名管理程序 CSR“hip”中的位，而 SGEIE、VSEIE、VSTIE 和 VSSIE 中的位 mie 是 hie 中相同位的别名。

18.4.3 机器第二陷阱值（“mtval2”）寄存器

mtval2 寄存器是一个 MXLEN 位的读/写寄存器，其格式如图 18.32 所示。当发生陷阱并进入 M 模式时，mtval2 会与 mtval 一起被写入额外的异常特定信息，以协助软件处理该陷阱。

当发生客户页错误陷阱并进入 M 模式时，mtval2 会被写入 0 或导致错误的客户物理地址，该地址右移 2 位。对于其他陷阱，mtval2 被设置为 0，但未来的标准或扩展可能会重新定义 mtval2 在其他陷阱中的设置。

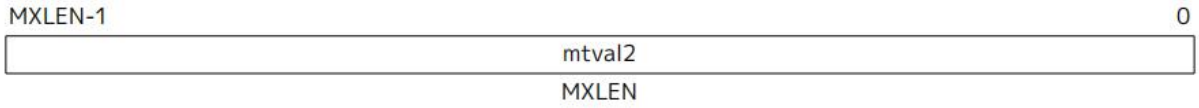


图18.32 机器模式第二陷阱值寄存器 (mtval2)

如果客户页错误是由于第一阶段（VS 阶段）地址转换期间的隐式内存访问引起的，写入 mtval2 的客户物理地址是导致错误的隐式内存访问的地址。CSR mtinst 中提供了额外的信息，以消除此类情况的歧义。

否则，对于导致客户页错误的未对齐加载和存储操作，mtval2 中的非 0 客户物理地址对应于由 mtval 中的虚拟地址指示的访问的错误部分。对于具有可变长度指令的系统上的指令客户页错误，mtval2 中的非 0 值对应于由 mtval 中的虚拟地址指示的指令的错误部分。mtval2 是一个 WARL 寄存器，必须能够保存 0，并且可能只能保存其他 2 位移位客户物理地址的任意子集（如果有）。

Ssdbltrap 扩展（参见 ssdbltrp）要求实现 mtval2 CSR。

18.4.4 机器陷阱指令（“mtinst”）寄存器

mtinst 寄存器是一个 MXLEN 位的读/写寄存器，其格式如图 18.33 所示。当发生陷阱并进入 M 模式时，mtinst 会被写入一个值，如果该值非 0，则提供有关触发陷阱的指令的信息，以协助软件处理该陷阱。

图18.33 机器模式陷阱指令寄存器 (mtinst)

mtinst 是一个 WARL 寄存器，仅需能够保存实现中在发生陷阱时可能自动写入的值。

18.5 两阶段地址转换

当当前的虚拟化模式 V 为 1 时，两级地址转换和保护机制生效。对于任何虚拟内存访问，原始的虚拟地址在第一阶段通过 VS 级地址转换（由 vsatp 寄存器控制）转换为客户物理地址。然后，客户物理地址在第二阶段通过客户物理地址转换（由 hgatp 寄存器控制）转换为监管者物理地址。这两个阶段分别称为VS 级转换和 G 级转换。尽管在 V=1 时无法禁用两级地址转换，但可以通过将相应的 vsatp 或 hgatp 寄存器置 0 有效禁用某一级转换。

vsstatus 字段中的 MXR 位（使仅可执行的页面通过显式加载变为可读）仅覆盖 VS 级页面保护。在 VS 级设置 MXR 不会覆盖客户物理页面保护。然而，在 HS 级设置 MXR 会同时覆盖 VS 级和 G 级的仅可执行权限。

当 V=1 时，通常会绕过地址转换的内存访问仅受 G 级地址转换的约束。这包括为支持 VS 级地址转换而进行的内存访问，例如对 VS 级页表的读写。

机器级的物理内存保护适用于监管者物理地址，并且无论虚拟化模式如何都生效。

18.5.1 访客物理地址转换

客户物理地址到监管者物理地址的映射由 CSR hgatp 控制（参见 hgatp）。当 hgatp 的 MODE 字段选择的地址转换方案为 Bare 时，客户物理地址与监管者物理地址相等，无需修改，并且在客户物理地址到监管者物理地址的简单转换中不应用内存保护。

当 hgatp.MODE 指定为 Sv32x4、Sv39x4、Sv48x4 或 Sv57x4 转换方案时，G 级地址转换是基于 Sv32、Sv39、Sv48 或 Sv57 的常规页式虚拟地址转换方案的变体。在每种情况下，输入地址的大小增加了 2 位（分别变为 34、41、50 或 59 位）。为了容纳这 2 个额外的位，根页表（仅根页表）扩展了四倍，变为 16 KiB，而不是通常的 4 KiB。与其更大的尺寸相匹配，根页表还必须对齐到 16 KiB 边界，而不是通常的 4 KiB 页边界。除非另有说明，Sv32、Sv39、Sv48 或 Sv57 的所有其他方面在 G 级转换中保持不变。非根页表和所有页表项（PTE）的格式与 sv32、sv39、sv48 和 sv57 中记录的格式相同。

对于 Sv32x4，输入的客户物理地址被划分为虚拟页号（VPN）和页偏移量，如 sv32x4va 所示。这种划分与 Sv32 虚拟地址的划分（如 sv32va 所示）相同，只是在 VPN[1]的高端多了 2 位。（请注意，划分后的客户物理地址的字段也与 Sv32 分配给物理地址的结构一一对应，如 sv32va 所示。）

对于 Sv39x4，输入的客户物理地址被划分为如 sv39x4va 所示。这种划分与 Sv39 虚拟地址的划分（如 sv39va 所示）相同，只是在 VPN[2]的高端多了 2 位。地址位 63:41 必须全部为 0，否则会发生客户页错误异常。

对于 Sv48x4，输入的客户物理地址被划分为如 sv48x4va 所示。这种划分与 Sv48 虚拟地址的划分（如 sv48va 所示）相同，只是在 VPN[3]的高端多了 2 位。地址位 63:50 必须全部为 0，否则会发生客户页错误异常。

对于 Sv57x4，输入的客户物理地址被划分为如 sv57x4va 所示。这种划分与 Sv57 虚拟地址的划分（如 sv57va 所示）相同，只是在 VPN[4]的高端多了 2 位。地址位 63:59 必须全部为 0，否则会发生客户页错误异常。

基于页的G级地址转换方案RV32 Sv32x4被定义为支持34位客户物理地址，以便RV32管理程序在虚拟化实际的32位RISC-V机器时不受限制，即使是那些具有33位或34位物理地址的机器。这可能包括机器虚拟化自身的可能性，如果它恰好使用33位或34位物理地址。将根页表的大小和对齐方式扩大四倍是将Sv32扩展到覆盖34位地址的最经济的方式。对于大多数（可能是所有）实际用途，根页表过大可能导致的12 KiB浪费预计可以忽略不计。

一致地支持虚拟化具有物理地址空间是虚拟地址空间四倍的机器，对于RV64也被认为具有一定的实用性。例如，对于实现39位虚拟地址（Sv39）的机器，这允许管理程序扩展支持高达41位的客户物理地址空间，而无需硬件支持48位虚拟地址（Sv48）或回退到使用影子页表模拟更大的地址空间。

Sv32x4、Sv39x4、Sv48x4 或 Sv57x4 客户物理地址的转换通过使用与 Sv32、Sv39、Sv48 或 Sv57 相同的算法完成，如 sv32algorithm 所示，但有以下例外：

hgatp 替代了通常的 satp；

为了开始转换，有效特权模式必须是 VS 模式或 VU 模式；

在检查 U 位时，当前特权模式始终被视为 U 模式；并且引发客户页错误异常，而不是常规页错误异常。

对于 G 级地址转换，所有内存访问（包括为访问 VS 级地址转换数据结构而进行的访问）都被视为用户级访问，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 级转换期间进行检查，与 VS 级转换相同。对于为支持 VS 级地址转换而进行的内存访问（例如读取/写入 VS 级页表），权限和在 G 级设置 A 和/或 D 位的需求被检查，就像对于隐式加载或存储，而不是原始访问类型。然而，任何异常总是针对原始访问类型（指令、加载或存储/AMO）报告。

所有 G 级 PTE 中的 G 位保留供未来标准使用。在标准扩展定义其用途之前，软件应清除该位以确保向前兼容性，并且硬件必须忽略该位。

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，这是因为G级转换和常规HS级地址转换之间有可能共享部分（或全部）页表。无论这种用法是否会变得普遍，选择不排除这种可能性。

18.5.2 访客页面错误

客户页错误陷阱可以从 M 模式委托到 HS 模式，受 CSR medeleg 的控制，但不能委托到其他特权模式。在客户页错误发生时，CSR mtval 或 stval 会像往常一样写入导致错误的客户虚拟地址，而 mtval2 或 htval 会写入 0 或导致错误的客户物理地址，该地址右移 2 位。CSR mtinst 或 htinst 也可能写入有关导致错误的指令或访问原因的信息，如 tinst-vals 中所述。

当指令获取或未对齐的内存访问跨越页边界时，涉及两种不同的地址转换。在这种情况下发生客户页错误时，写入 mtval/stval 的错误虚拟地址与常规页错误所需的地址相同。因此，错误虚拟地址可能是高于指令原始虚拟地址的页边界地址，如果该页边界的字节在被访问的字节中。

当客户页错误不是由于 VS 级地址转换的隐式内存访问引起时，写入 mtval2/htval 的非 0 客户物理地址应与写入 mtval/stval 的确切虚拟地址相对应。

18.5.3 内存管理围栏

SFENCE.VMA 指令的行为受当前虚拟化模式 V 的影响。当 V=0 时，虚拟地址参数是 HS 级虚拟地址，ASID 参数是 HS 级 ASID。该指令仅对 HS 级地址转换结构的存储进行排序，并随后进行 HS 级地址转换。

当 V=1 时，SFENCE.VMA 的虚拟地址参数是当前虚拟机内的客户虚拟地址，ASID 参数是当前虚拟机内的 VS 级 ASID。当前虚拟机由 CSR hgatp 的 VMID 字段标识，有效 ASID 可以被视为该 VMID 与 VS 级 ASID 的组合。SFENCE.VMA 指令仅对同一虚拟机的 VS 级地址转换结构的存储进行排序，并随后进行 VS 级地址转换，即仅在 hgatp.VMID 与执行 SFENCE.VMA 时相同时才生效。

管理程序指令 HFENCE.VVMA 和 HFENCE.GVMA 提供了额外的内存管理栅栏，以补充 SFENCE.VMA。pmp-vmem 讨论了物理内存保护（PMP）与基于页的地址转换之间的交集。文中指出，当以影响保存页表

的物理内存或页表指向的物理内存的方式修改 PMP 设置时，M 模式软件必须将 PMP 设置与虚拟内存系统同步。对于 HS 级地址转换，这是在 M 模式下执行 SFENCE.VMA 指令（rs1=x0 和 rs2=x0）后完成的，在 PMP CSR 被写入之后。与 G 级和 VS 级数据结构的同步也是必要的。执行 HFENCE.GVMA 指令（rs1=x0 和 rs2=x0）刷新所有与最终转换的监管者物理地址对应的、缓存了 PMP 设置的 G 级或 VS 级地址转换缓存条目。不需要 HFENCE.VVMA 指令。

类似地，如果 menvcfg 中的 PBMTE 位被更改，执行 HFENCE.GVMA 指令（rs1=x0 和 rs2=x0）足以同步 G 级和 VS 级 PTE 的 PBMT 字段的更改解释。

相比之下，如果 henvcfg 中的 PBMTE 位被更改，执行 HFENCE.VVMA 指令（rs1=x0 和 rs2=x0）足以同步当前活动 VMID 的 VS 级 PTE 的 PBMT 字段的更改解释。

注意：没有提供机制来原子地更改 vsatp 和 hgatp。因此，为了防止推测执行导致一个客户的 VS 级转换在另一个客户的 VMID 下被缓存，世界切换代码应将 vsatp 置 0，然后交换 hgatp，最后写入新的 vsatp 值。类似地，如果需要切换 henvcfg.PBMTE，应在将 vsatp 置 0 后但在写入新的 vsatp 值之前进行切换，从而无需执行 HFENCE.VVMA 指令。

18.6 陷阱

18.6.1 陷阱原因代码

管理程序扩展对陷阱原因编码进行了扩充。hcauses 列出了在实现管理程序扩展时可能的 M 模式和 HS 模式陷阱原因代码。新增了 VS 级中断（中断 2、6、10）、监管者级客户外部中断（中断 12）、虚拟指令异常（异常 22）以及客户页错误（异常 20、21、23）的代码。此外，来自 VS 模式的环境调用被分配原因代码 10，而来自 HS 模式或 S 模式的环境调用则像往常一样使用原因代码 9。

表 18.6 实现 Hypervisor 扩展时的机器模式与管理模式异常原因寄存器（mcause/scause）值

中断	异常代码	描述
1	0	保留
1	1	监控软件中断
1	2	虚拟监控软件中断
1	3	机器软件中断
1	4	保留
1	5	监控定时器中断
1	6	虚拟监控定时器中断
1	7	机器定时器中断
1	8	保留
1	9	主管外部中断
1	10	虚拟管理器外部中断
1	11	机器外部中断
1	12	主管客人外部中断
1	13	为反溢出中断保留
1	14-15	保留
1	≥16	指定作平台用途
0	0	指令地址错位
0	1	指令存取故障
0	2	非法指令
0	3	断点

0	4	负载地址错位
0	5	负载接入故障
0	6	仓库/AMO 地址未对齐
0	7	存储/AMO 访问故障
0	8	u 模式或 vu 模式下的环境调用
0	9	从 hs 模式调用环境
0	10	从 vs 模式进行环境调用
0	11	从 m 模式调用环境
0	12	说明页故障
0	13	加载页面故障
0	14	保留
0	15	Store/AMO 页面错误
0	16-19	保留
0	20	说明来宾页面错误
0	21	加载访客页面错误
0	22	虚拟指令
0	23	商店/AMO 客人页面错误
0	24-31	指定自定义使用
0	32-47	保留
0	48-63	指定自定义使用
0	≥64	保留

HS 模式和 VS 模式的 ECALL 使用不同的原因值，以便可以分别委托。

当 V=1 时，如果尝试执行的指令是 HS 特定类别的指令但由于权限不足或由于监管者或管理程序 CSR（如 scounteren 或 hcounteren）明确禁用该指令而无法执行时，通常会引发虚拟指令异常（代码 22）而不是非法指令异常。如果指令在 HS 模式下（对于指令的寄存器操作数的某些值）执行是有效的，假设 CSR mstatus 的字段 TSR 和 TVM 都为 0，该指令属于 HS 特定类别的指令。

对于访问 32 位高半 CSR（如 cycleh 和 htimedeltah）的 CSR 指令，适用特殊规则。当 V=1 且 XLEN=32 时，如果对相应低半 CSR（例如 cycle 或 htimedelta）的相同 CSR 指令是 HS 限定的，则尝试访问高半 CSR 的无效操作会引发虚拟指令异常而不是非法指令异常。

当XLEN>32时，试图访问高半CSR总是会引发非法指令异常。

具体来说，以下情况会引发虚拟指令异常：

在 VS 模式下，当 hcounteren 中的相应位为 0 且 mcounteren 中的相同位为 1 时，尝试访问非高半计数器 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，尝试访问高半计数器 CSR；

在 VU 模式下，当 hcounteren 或 scounteren 中的相应位为 0 且 mcounteren 中的相同位为 1 时，尝试访问非高半计数器 CSR；

在 VU 模式下，如果 XLEN=32，当 hcounteren 或 scounteren 中的相应位为 0 且 mcounteren 中的相

同位为 1 时，尝试访问高半计数器 CSR；

在 VS 模式或 VU 模式下，尝试执行管理程序指令（HLV、HLVX、HSV 或 HFENCE）；

在 VS 模式或 VU 模式下，尝试访问已实现的非高半管理程序 CSR 或 VS CSR，假设 mstatus.TVM=0，且相同的访问（读/写）在 HS 模式下是允许的；

在 VS 模式或 VU 模式下，如果 XLEN=32，尝试访问已实现的高半管理程序 CSR 或高半 VS CSR，假设 mstatus.TVM=0，且对 CSR 的低半伙伴的相同访问（读/写）在 HS 模式下是允许的；

在 VU 模式下，当 mstatus.TW=0 时，尝试执行 WFI，或尝试执行监管者指令（SRET 或 SFENCE）；

在 VU 模式下，尝试访问已实现的非高半监管者 CSR，假设 mstatus.TVM=0，且相同的访问（读/写）在 HS 模式下是允许的；

在 VU 模式下，如果 XLEN=32，尝试访问已实现的高半监管者 CSR，假设 mstatus.TVM=0，且对 CSR 的低半伙伴的相同访问在 HS 模式下是允许的；

在 VS 模式下，当 hstatus.VTW=1 且 mstatus.TW=0 时，尝试执行 WFI，除非指令在实现特定的有限时间内完成；

在 VS 模式下，当 hstatus.VTSR=1 时，尝试执行 SRET；以及

在 VS 模式下，当 hstatus.VTVM=1 时，尝试执行 SFENCE.VMA 或 SINVAL.VMA 指令或访问 satp。

RISC-V 特权架构的其他扩展可能会增加在 V=1 时引发虚拟指令异常的情况。

在虚拟指令陷阱发生时，mtval 或 stval 的写入方式与非法指令陷阱相同。

管理程序通常需要模拟引发虚拟指令异常的指令，以支持嵌套管理程序或其他原因。机器级别通常会将虚拟指令陷阱直接委托给HS级别，而非法指令陷阱可能会首先在M模式下处理，然后有条件地（由软件）委托给HS级别。因此，虚拟指令陷阱通常预计比非法指令陷阱处理得更快。

当不模拟陷阱指令时，管理程序应将虚拟指令陷阱转换为客户虚拟机的非法指令异常。

由于 mstatus 中的 TSR 和 TVM 仅影响 S 模式（HS 模式），因此在确定 VS 模式中的异常时可以忽略这些字段。

寄存器 sstatus 和 vsstatus 中的 FS 和 VS 字段偏离了通常的 HS 限定规则。如果由于 sstatus 或 vsstatus 中的 FS 或 VS 为 0 而阻止指令执行，引发的异常始终是非法指令异常，而不是虚拟指令异常，如表 18.7。

早期的H扩展实现以这种方式特殊处理sstatus和vsstatus中的FS和VS，这种行为已被规范化以保持软件的兼容性。

表 18.7 实现管理程序扩展时的同步异常优先级

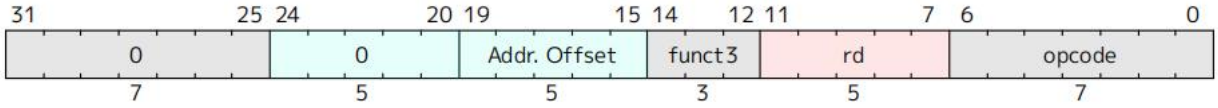
优先级	Exc code	描述
最高级	3	指令地址断点
	12, 20, 1	在指令地址转换过程中： 首次遇到页面错误、访客页面错误或访问错误
	1	有物理地址的：

		指令存取故障
	2 22 0 8, 9, 10, 11 3 3	非法指令 虚拟指令 指令地址错位 环境调用 环境破坏 加载/存储/AMO 地址断点
	4,6	(可选): 加载/存储/AMO 地址未对齐
	13, 15, 21, 23, 5, 7	在显式内存访问的地址转换期间: 首次遇到页面错误、访客页面错误或访问错误
	5, 7	带物理地址的显式内存访问: 加载/存储/AMO 访问故障
最低级	4, 6	如果没有更高的优先级: 加载/存储/AMO 地址未对齐

如果一条指令可能引发多个同步异常，HSyncExcPrio 中的优先级递减顺序指示了哪个异常会被捕获并报告在 mcause 或 scause 中。

18.6.2 陷阱进入

当在 HS 模式或 U 模式下发生陷阱时，会进入 M 模式，除非被 medeleg 或 mideleg 委托，在这种情况下会进入 HS 模式。当在 VS 模式或 VU 模式下发生陷阱时，会进入 M 模式，除非被 medeleg 或 mideleg 委托，在这种情况下会进入 HS 模式，除非进一步被 hedeleg 或 hideleg 委托，在这种情况下会进入 VS 模式。



当陷阱进入 M 模式时，虚拟化模式 V 被设置为 0，并且 mstatus（或 mstatush）中的字段 MPV 和 MPP 根据 h-mpp 进行设置。进入 M 模式的陷阱还会写入 mstatus/mstatush 中的字段 GVA、MPIE 和 MIE，并写入 CSR mepc、mcause、mtval、mtval2 和 mtinst。

进入 M 模式后 mstatus/mstatush 字段 MPV 和 MPP 的值。在陷阱返回时，当 MPP=3 时忽略 MPV。

表 18.8 M 模式时 MPV 和 MPP 字段的值

Previous Mode	MPV	MPP
U 模式	0	0
HS 模式	0	1
M 模式	0	3
VU 模式	1	0
VS 模式	1	1

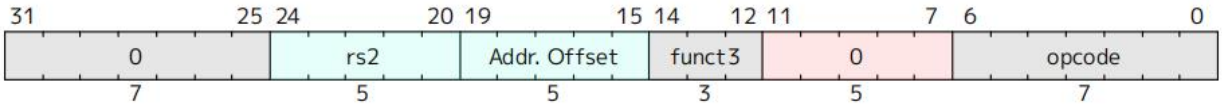
当陷阱进入 HS 模式时，虚拟化模式 V 被设置为 0，并且 hstatus.SPV 和 sstatus.SPP 根据 h-spp 进行设置。如果陷阱发生前 V 为 1，则 hstatus 中的字段 SPVP 设置为与 sstatus.SPP 相同；否则，SPVP

保持不变。进入 HS 模式的陷阱还会写入 hstatus 中的字段 GVA、sstatus 中的字段 SPIE 和 SIE，以及 CSR sepc、scause、stval、htval 和 htinst。

表 18.9 陷入 HS 模式后 hstatus.SPV 与 sstatus.SPP 字段的值

Previous Mode	SVP	SPP
U 模式	0	0
HS 模式	0	1
VU 模式	1	0
VS 模式	1	1

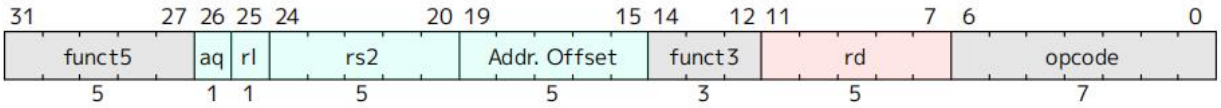
当陷阱进入 VS 模式时，vsstatus.SPP 根据 h-vspp 进行设置。寄存器 hstatus 和 HS 级别的 sstatus 不会被修改，虚拟化模式 V 保持为 1。进入 VS 模式的陷阱还会写入 vsstatus 中的字段 SPIE 和 SIE，并写入 CSR vsepc、vscause 和 vstval。



18.6.3 “mtinst“或“htinst“的转换指令或伪指令

在任何进入 M 模式或 HS 模式的陷阱中，以下值之一会自动写入相应的陷阱指令 CSR mtinst 或 htinst：

陷阱指令的转换；自定义值（仅在陷阱指令是非标准指令时允许）；或特殊的伪指令。



除非需要伪指令值（稍后描述），否则写入 mtinst 或 htinst 的值可能始终为 0，表示硬件在此特定陷阱中未提供寄存器中的信息。

写入陷阱指令CSR的值有两个目的。第一个目的是提高陷阱处理程序中指令模拟的速度，部分原因是允许处理程序跳过从内存加载陷阱指令，部分原因是避免了解码和执行指令的部分工作。第二个目的是通过伪指令提供有关由VS级地址转换的隐式内存访问引起的客户页错误异常的额外信息。

写入陷阱指令的转换而不是简单地复制原始指令，是为了最小化硬件的负担，同时仍然为陷阱处理程序提供模拟指令所需的信息。实现可以随时通过用0替换转换后的指令来减少其工作量。

在中断时，写入陷阱指令寄存器的值始终为 0。在同步异常时，如果写入非 0 值，则该值必须满足以下条件之一：

位 0 为 1，并且将位 1 替换为 1 会使该值成为标准指令的有效编码。

在这种情况下，陷阱指令的类型与寄存器值指示的类型相同，并且寄存器值是陷阱指令的转换，如后面所定义。例如，如果位 1:0 为二进制 11，并且寄存器值是标准 LW（加载字）指令的编码，则陷阱指令是 LW，并且寄存器值是陷阱 LW 指令的转换。

这是一个自定义值。陷阱指令是非标准指令。自定义值的解释不由本标准另行规定。该值是后面定义的特殊伪指令之一，所有这些伪指令的位 1:0 等于 00。

这三种情况排除了许多其他可能的值，例如所有位 1:0 等于二进制 10 的值。未来的标准或扩展可能会定义其他情况，从而允许当前被排除的值。软件可以安全地将陷阱指令寄存器中无法识别的值视为

0。

为了与未来修订的本标准向前兼容，解释mtinst或htinst中非0值的软件必须完全验证该值是否符合上述列出的情况之一。例如，对于RV64，发现mtinst的位6:0为0000011且位14:12为010并不足以确定第一种情况适用且陷阱指令是标准LW指令；相反，软件还必须确认mtinst的位63:32全为0。未来的标准可能会为64位mtinst定义新值，这些值在位63:32中非0，但可能在位31:0中巧合地具有与标准RV64指令相同的位模式。

与标准指令不同，对于自定义值，没有要求其指令编码与陷阱指令的“类型“相同（甚至与陷阱指令有任何关联）。

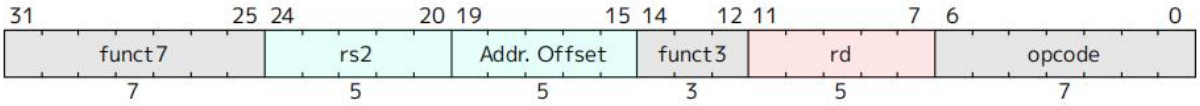


表 18.10 显示了可能自动写入陷阱指令寄存器的每个标准异常原因的值。对于阻止指令获取的异常，只能写入 0 或伪指令值。只有在陷阱指令是非标准指令时，才能自动写入自定义值。未来的标准或扩展可能会允许写入其他值，这些值从先前建立的允许值集合中选择。

可以自动写入异常陷阱上的陷阱指令（“mtinst“或“htinst”）寄存器的值。

表 18.10 可以自动写入异常陷阱上的陷阱指令（“mtinst”或“htinst”）寄存器的值

异常	0	转换标准指令	传统值	伪指令值
指令地址错位	是	否	是	否
通用场景	是	否	否	否
指令存取故障	是	否	否	否
非法指令	是	否	是	否
断点	是	否	是	否
虚拟指令				
负载地址错位	是	是	是	否
负载接入故障	是	是	是	否
仓库地址未对齐	是	是	是	否
存储/AMO 访问故障	是	是	是	否
环境调用	是	否	是	否
说明页故障	是	否	否	否
加载页面故障	是	是	是	否
Store/AMO 页面错误	是	是	是	否
说明来宾页面错误	是	否	否	是
加载访客页面错误	是	是	是	是
商店/AMO 用户页面错误	是	是	是	是

如表中所列，同步异常可能会将陷阱指令的标准转换写入陷阱指令寄存器，仅适用于由显式内存访问（来自加载、存储和 AMO 指令）引起的异常。因此，目前仅为这些内存访问指令定义了标准转换。如果发生同步陷阱的标准指令未定义转换，则陷阱指令寄存器应写入 0（或在某些情况下，写入特殊的伪指令值）。

对于不是压缩指令且是 LB、LBU、LH、LHU、LW、LWU、LD、FLW、FLD、FLQ 或 FLH 的标准加载指令，

转换后的指令具有 transformedloadinst 中所示的格式。

.转换后的非压缩加载指令（LB、LBU、LH、LHU、LW、LWU、LD、FLW、FLD、FLQ 或 FLH）。字段 funct3、rd 和 opcode 与陷阱加载指令相同。

对于不是压缩指令且是 SB、SH、SW、SD、FSW、FSD、FSQ 或 FSH 的标准存储指令，转换后的指令具有 transformedstoreinst 中所示的格式。

.转换后的非压缩存储指令（SB、SH、SW、SD、FSW、FSD、FSQ 或 FSH）。字段 rs2、funct3 和 opcode 与陷阱存储指令相同。

对于标准原子指令（加载保留、存储条件或 AMO 指令），转换后的指令具有 transformedatomicinst 中所示的格式。

.转换后的原子指令（加载保留、存储条件或 AMO 指令）。除位 19:15（Addr. Offset）外，所有字段与陷阱指令相同。

对于标准虚拟机加载/存储指令（HLV、HLVX 或 HSV），转换后的指令具有.转换后的虚拟机加载/存储指令（HLV、HLVX、HSV）。除位 19:15（Addr. Offset）外，所有字段与陷阱指令相同。

在上述所有转换后的指令中，替换指令 rs1 字段的 Addr. Offset 字段（位 19:15）是故障虚拟地址（写入 mtval 或 stval）与原始虚拟地址的正差。此差值仅在对齐错误的内存访问中可能非 0。还要注意，对于基本加载和存储指令，转换将指令的立即偏移字段替换为 0。

对于标准压缩指令（16 位大小），转换后的指令按以下步骤找到：

将压缩指令扩展为其 32 位等效指令。

转换 32 位等效指令。

将位 1 替换为 0。

如果陷阱指令是压缩指令，则转换后的标准指令的位 1:0 为二进制 01；如果不是，则为 11。

在解码mtinst或htinst的内容时，一旦软件确定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，就不需要再确认立即偏移字段（31:25，以及24:20或11:7）是否为0。知道寄存器的值是基本加载/存储的编码足以证明陷阱指令是相同类型的。

本标准的未来版本可能会在当前为0的字段中添加信息。然而，为了向后兼容，任何此类信息将仅用于性能目的，并且可以安全地忽略。

对于客户页错误，如果满足以下条件，陷阱指令寄存器将写入特殊的伪指令值：(a) 错误是由 VS 级地址转换的隐式内存访问引起的，并且(b) 非 0 值（错误的客户物理地址）被写入 mtval2 或 htval。如果这两个条件都满足，写入 mtinst 或 htinst 的值必须取自 pseudoinsts；不允许为 0。

访客页面错误的特殊伪指令值。当 VSXLEN=32 时使用 RV32 值，当 VSXLEN=64 时使用 RV64 值。

定义的伪指令值设计为与基本加载和存储的编码紧密对应，如表 18.11 所示。与伪指令的特殊伪指令相对应的标准指令。

表 18.11 特殊伪指令相对应的标准指令

编码	指令
0x00002003	lw x0,0(x0)
0x00002023	sw x0,0(x0)
0x00003003	ld x0,0(x0)
0x00003023	sd x0,0(x0)

写入伪指令（0x00002020 或 0x00003020）用于机器尝试自动更新 VS 级页表中的 A 和/或 D 位的情况。所有其他用于 VS 级地址转换的隐式内存访问将是读取。如果机器从不自动更新 VS 级页表中的 A

或 D 位（将此留给软件），则写入情况永远不会发生。伪指令忽略了此类页表更新实际上必须是原子的，而不仅仅是简单写入的事实。

如果M模式下可能出现需要伪指令值的条件，则mtinst不能完全为只读0；同样适用于HS模式和htinst。然而，在这种情况下，陷阱指令寄存器可能仅支持值0和0x00002000或0x00003000，以及可能的0x00002020或0x00003020，每个寄存器在硬件中只需要一两个触发器。

在这里忽略页表更新的原子性要求没有害处，因为在这些情况下，管理程序不期望模拟失败的隐式内存访问。相反，管理程序获得了足够的信息来使内存可访问（例如，通过恢复缺失的虚拟内存页），然后通过重试故障指令恢复执行。

18.6.4 陷阱返回

MRET 指令用于从进入 M 模式的陷阱返回。MRET 首先根据 mstatus 或 mstatush 中的 MPP 和 MPV 值确定新的特权模式，如 h-mpp 中编码所示。然后，MRET 在 mstatus/mstatush 中设置 MPV=0、MPP=0、MIE=MPIE 和 MPIE=1。最后，MRET 设置先前确定的特权模式，并设置 pc=mepc。

SRET 指令用于从进入 HS 模式或 VS 模式的陷阱返回。其行为取决于当前的虚拟化模式。

在 M 模式或 HS 模式下执行时（即 V=0），SRET 首先根据 hstatus.SPV 和 sstatus.SPP 的值确定新的特权模式，如 h-spp 中编码所示。然后，SRET 设置 hstatus.SPV=0，并在 sstatus 中设置 SPP=0、SIE=SPIE 和 SPIE=1。最后，SRET 设置先前确定的特权模式，并设置 pc=sepc。

在 VS 模式下执行时（即 V=1），SRET 根据 h-vspp 设置特权模式，在 vsstatus 中设置 SPP=0、SIE=SPIE 和 SPIE=1，最后设置 pc=vsepc。

如果实现了 Ssdbltrp 扩展，当在 HS 模式下执行 SRET 时，如果新的特权模式是 VU，则 SRET 指令将 vsstatus.SDT 设置为 0。在 VS 模式下执行时，vsstatus.SDT 被设置为 0。

19 RISC-V 特权指令集列表

本章列出了RISC-V特权架构中定义的所有指令的指令集列表。非特权指令的指令集列表（包括ECALL和EBREAK指令）在本手册的第一卷中提供。

31 27	26 25 24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type

陷阱返回指令

0001000	00010	00000	000	00000	1110011	SRET
0011000	00010	00000	000	00000	1110011	MRET
0111000	00010	00000	000	00000	1110011	MNRET

中断管理指令

0001000	00101	00000	000	00000	1110011	WFI

监督程序内存管理指令

0001001	rs2	rs1	000	00000	1110011	SFENCE.VMA

虚拟机监控程序内存管理指令

0010001	rs2	rs1	000	00000	1110011	HFENCE.VVMA
0110001	rs2	rs1	000	00000	1110011	HFENCE.GVMA

虚拟机监控程序虚拟机加载和存储指令

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

虚拟机监控程序虚拟机加载和存储指令，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内存管理扩展

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

20 历史

20.1 加州大学伯克利分校的研究资助

RISC-V 架构及实现的研发工作得到了以下资助方的部分支持：

• 并行计算实验室（Par Lab）：研究资金来自微软（奖项编号#024263）和英特尔（奖项编号#024894），以及加州大学探索计划（奖项编号#DIG07-10227）的配套资助。额外支持来自 Par Lab 合作企业：诺基亚、英伟达、甲骨文和三星。

• Isis 项目：美国能源部奖项 DE-SC0003624。

• ASPIRE 实验室：美国国防高级研究计划局（DARPA）PERFECT 计划，奖项 HR0011-12-2-0016。DARPA POEM 计划，奖项 HR0011-11-C-0100。未来架构研究中心（C-FAR），由半导体研究协会（SRC）资助的 STARnet 中心。以及来自 ASPIRE 产业合作伙伴英特尔，以及成员企业谷歌、华为、诺基亚、英伟达、甲骨文和三星的额外支持。

本文内容不代表美国政府立场或政策，亦不暗示其官方认可。