第 2 讲：中断、异常与系统调用

第二节：从 OS 角度再理解 RISC-V

向勇、陈渝、李国良

清华大学计算机系

xyong,yuchen,liguoliang@tsinghua.edu.cn

2021 年 9 月 16 日

提纲

1. 第二节：从 OS 角度看 RISC-V
   - 主流 CPU 指令集
   - RISC-V 系统模式：概述
   - RISC-V 系统编程：M 模式
   - RISC-V 系统编程：S 模式

主流 CPU 指令集比较 [*Waterman*2017]

x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Endianness	Little

ARM architectures

Designer	ARM Holdings
Bits	32-bit, 64-bit
Introduced	1985; 31 years ago
Design	RISC
Type	Register-Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 user-space compatibility[1]
Endianness	Bi (little as default)

RISC-V

Designer	University of California, Berkeley
Bits	32, 64, 128
Introduced	2010
Version	2.2
Design	RISC
Type	Load-store
Encoding	Variable
Branching	Compare-and-branch
Endianness	Little

主流 CPU 指令集比较 [*Waterman*2017]

x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Endianness	Little

ARM architectures

Designer	ARM Holdings
Bits	32-bit, 64-bit
Introduced	1985; 31 years ago
Design	RISC
Type	Register-Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 user-space compatibility[1]
Endianness	Bi (little as default)

RISC-V

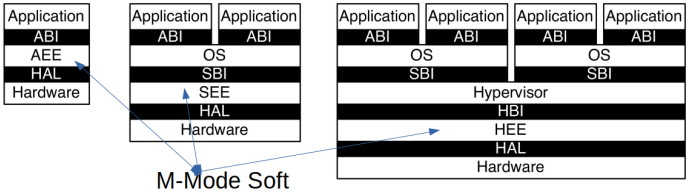
Designer	University of California, Berkeley
Bits	32, 64, 128
Introduced	2010
Version	2.2
Design	RISC
Type	Load-store
Encoding	Variable
Branching	Compare-and-branch
Endianness	Little

ISA	Pages	Words	Hours to read	Weeks to read
RISC-V	236	76,702	6	0.2
ARM-32	2736	895,032	79	1.9
x86-32	2198	2,186,259	182	4.5

提纲

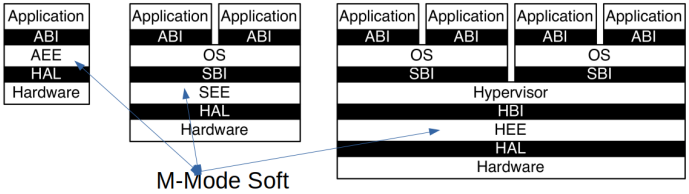
1. 第二节：从 OS 角度看 RISC-V
   - 主流 CPU 指令集
   - RISC-V 系统模式：概述
   - RISC-V 系统编程：M 模式
   - RISC-V 系统编程：S 模式

RISC-V 系统模式：概述



- RISC-V 系统模式即 RISC-V 的特权级模式
- 现代处理器一般具有多个特权级的模式
- 每一个特权级所能够执行的指令以及能够访问的处理器资源是不同的

RISC-V 系统模式：概述

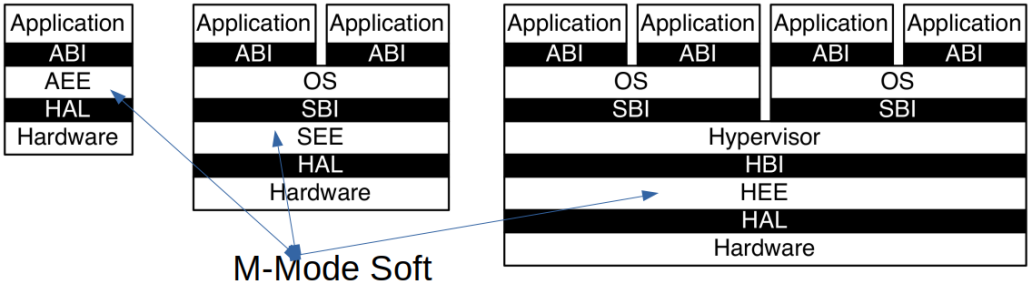


RISC-V 系统模式：内核态特权级

简单情况下：

- 用户态专门用来执行应用程序
- 内核态专门用来执行操作系统
- 内核态的操作系统具有完全的硬件控制能力

RISC-V 系统模式：隔离



- **不同软件层有清晰的硬件隔离**
- AEE：Application Execution Environment
- ABI：Application Bianry Interface
- **MODE – U**：User | **S**: Supervisor | **H**: Hypervisor | **M**: Machine

RISC-V 系统模式：模式组合

0	00	User/Application	U
1	01	Supervisor	S
2	10	Hypervisor	H
3	11	Machine	M

- M, S, U for systems running Unix-like general operating systems

RISC-V 系统模式：控制状态寄存器

- 设置 CSR(控制状态寄存器) 实现隔离
  - 防止应用程序访问设备和敏感的 CPU 寄存器
  - 例如地址空间配置寄存器
- 强制隔离以避免对整个系统的可用性/可靠性/安全影响
  - 运行的程序通常是隔离的单元
  - 防止恶意程序、病毒、木马破坏或监视应用程序或干扰操作系统
    - 读/写内存: mstatus/satp CSR
    - 使用 100%的 CPU： mstatus/stvec CSR
    - mstatus/satp/stvec CSR 页表异常处理

RISC-V 系统编程

- 系统编程即据了解处理器的特权级架构，熟悉各个特权级能够访问的寄存器资源，内存资源，外设资源
- 编写内核级代码，构造操作系统，支持应用程序执行
  - 内存管理　进程调度
  - 异常处理　中断处理
  - 系统调用　外设控制
- 系统编程通常没有广泛用户编程库和方便的动态调试手段的支持
- 本课程的系统编程主要集中在 RISC-V 的 S-Mode 和 U-Mode

RISC-V 系统编程：特权操作

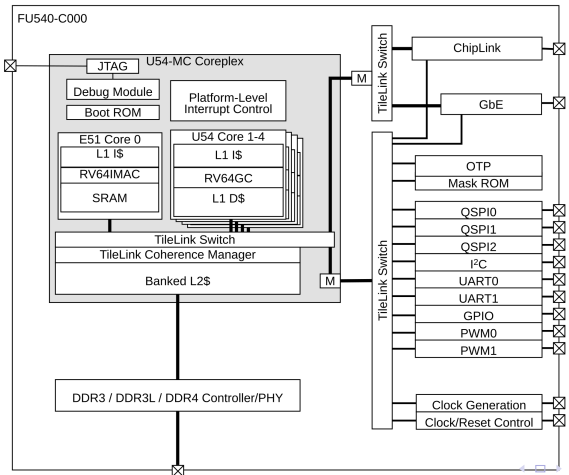
- 特权操作：包括特权指令以及相应的指令码和操作数
- 指令码非常少：
  - mret 机器模式返回　　sret 监管者模式返回
  - wfi 等待中断　　sfense.vma 虚拟地址屏障指令
- 很多功能通过控制状态寄存器（CSR：Control State Register）来实现

RISC-V 系统编程：M-Mode

- M 模式是 RISC-V 中 hart（硬件线程，hardware thread）的最高权限模式
- M 模式下，hart 对内存，I/O，启动和配置的底层功能有完全的使用权
- 标准 RISC-V 处理器都必须实现的权限模式
- M 模式最重要的特性是拦截和处理中断和异常
  - 异常：执行期间产生，访问无效的寄存器地址，或执行无效操作码的指令
  - 中断：指令流异步的外部事件，中断，如时钟中断
- RISC-V 要求实现精确异常：保证异常之前的所有指令都完整执行，后续指令都没有开始执行

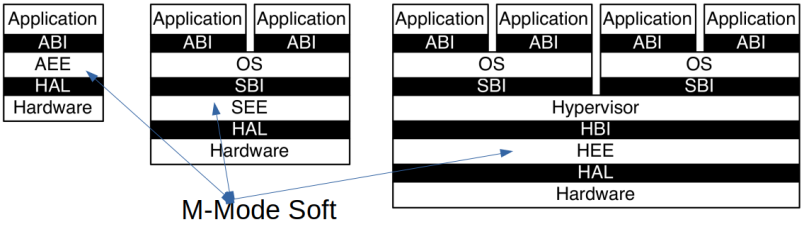
RISC-V 系统编程：M-Mode：RISC-V 中断机制

- 中断是异步发生，是来自处理器外部的 I/O 设备的信号的结果。
- Timer 可以稳定定时地产生中断
  - 防止应用程序死占着 CPU 不放，让 OS Kernel 能得到执行权...

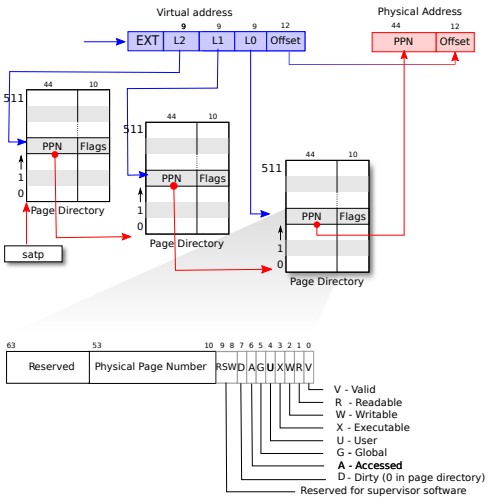


RISC-V 系统编程：M-Mode：RISC-V 中断机制

- 中断是异步发生，是来自处理器外部的 I/O 设备的信号的结果。
- Timer 可以稳定定时地产生中断
  - 防止应用程序死占着 CPU 不放，让 OS Kernel 能得到执行权...
  - 由高特权模式下的软件获得 CPU 控制权
  - 也可由高特权模式下的软件授权低特权模式软件处理中断



RISC-V 系统编程：S-Mode: 虚拟内存



- 有虚拟内存的好处/坏处
  - 灵活的不连续内存分配
  - 多个运行程序的地址空间相互隔离/共享
  - 进一步隔离应用程序与 OS 内核
  - 多了访问页表的开销

RISC-V 系统编程：RISC-V 的异常与中断

RISC-V 的异常：通过 mcause 寄存器的不同位来表示

0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store address misaligned
0	7	Store access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	15	Store page fault

RISC-V 系统编程：RISC-V 的异常与中断

RISC-V 的中断：通过 mcause 寄存器的不同位来表示

- 软件中断：通过向内存映射寄存器写入数据来触发，一个 hart 中断另外一个 hart（处理器间中断）
- 时钟中断：hart 的时间计数器寄存器 mtime 大于时间比较寄存器 mtimecmp
- 外部中断：由中断控制器触发，大部分情况下的外设都会连到这个中断控制器

1	1	Supervisor software interrupt
1	3	Machine software interrupt
1	5	Supervisor timer interrupt
1	7	Machine timer interrupt
1	9	Supervisor external interrupt
1	11	Machine external interrupt

RISC-V 系统编程：RISC-V 的系统调用

- U-Mode 下的应用程序不能够直接使用计算机的物理资源
- 环境调用异常：在执行 ecall 的时候发生，相当于系统调用
- 操作系统可以直接访问物理资源
- 如果应用程序需要使用硬件资源怎么办？
  - 在屏幕上打印” hello world”
  - 从文件中读入数据

RISC-V 系统编程：RISC-V 的系统调用

- U-Mode 下的应用程序不能够直接使用计算机的物理资源
- 环境调用异常：在执行 ecall 的时候发生，相当于系统调用
- 操作系统可以直接访问物理资源
- 如果应用程序需要使用硬件资源怎么办？
  - 在屏幕上打印” hello world”
  - 从文件中读入数据
- 通过系统调用从操作系统中获得服务

RISC-V 系统编程：思考题

- 如何通过断点异常来实现调试器的断点调试功能？
- 如何实现单步跟踪？

提纲

1. 第二节：从 OS 角度看 RISC-V
   - 主流 CPU 指令集
   - RISC-V 系统模式：概述
   - RISC-V 系统编程：M 模式
   - RISC-V 系统编程：S 模式

RISC-V 系统编程：M 模式下的异常处理

8 个控制状态寄存器（CSR）是机器模式下异常处理的必要部分:part1

- mtvec(Machine Trap Vector)：发生异常时处理器需要跳转到的地址
- mepc(Machine Exception PC)：指向发生异常的指令
- mcause(Machine Exception Cause)：发生异常的种类
- mie(Machine Interrupt Enable)：处理器目前能处理和必须忽略的中断

RISC-V 系统编程：M 模式下的异常处理

8 个控制状态寄存器（CSR）是机器模式下异常处理的必要部分:part2

- mip(Machine Interrupt Pending)：正准备处理的中断
- mtval(Machine Trap Value)：附加信息：地址异常中出错的地址，非法指令异常的指令等
- mscratch(Machine Scratch)：暂时存放一个字大小的数据
- mstatus(Machine Status)：机器的状态

RISC-V 系统编程：M 模式下的异常处理

mstatus 控制状态寄存器

- 在仅有机器模式，且没有 F 和 V 扩展的简单处理器中，有效的域只有全局使能，MIE 和 MPIE（它在异常发生后保存 MIE 的旧值）
- 处理器在 M 模式下运行时，只有在全局中断使能位 mstatus.MIE 置 1 时才会产生中断

XLEN-1	XLEN-2	23	22	21	20	19	18	17	
SD	0		TSR	TW	TVM	MXR	SUM	MPRV	
1	XLEN-24		1	1	1	1	1	1	

16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
XS	FS	MPP	0	SPP	MPIE	0	SPIE	UPIE	MIE	0	SIE	UIE
2	2	2	2	1	1	1	1	1	1	1	1	1

图：mstatus 寄存器

RISC-V 系统编程：M 模式下的中断相关寄存器

- M 模式中断寄存器。它们是宽为 XLEN 位的读/写寄存器，用于保存待处理的中断（mip）和中断使能位（mie）CSR。
- 只有与 mip 中的位对应的低权限软件中断（USIP，SSIP）、时钟中断（UTIP，STIP）和外部中断（UEIP，SEIP）的位才能通过该 CSR 的地址写入；其余的位是只读的。

XLEN-1	12	11	10	9	8	7	6	5	4	3	2	1	0
WIRI		MEIP	WIRI	SEIP	UEIP	MTIP	WIRI	STIP	UTIP	MSIP	WIRI	SSIP	USIP
WPRI		MEIE	WPRI	SEIE	UEIE	MTIE	WPRI	STIE	UTIE	MSIE	WPRI	SSIE	USIE
XLEN-12		1	1	1	1	1	1	1	1	1	1	1	1

图: mie & mip 寄存器

RISC-V 系统编程：M 模式下的中断相关寄存器

- 机器模式和监管模式异常向量（trap-vector）基地址寄存器（mtvec 和 stvec）CSR。他们是位宽为 XLEN 的读/写寄存器，用于保存异常向量的配置，包括向量基址（BASE）和向量模式（MODE）。BASE 域中的值必须按 4 字节对齐。MODE = 0 表示所有异常都把 PC 设置为 BASE。MODE = 1 会在异步中断时将 PC 设置为 (base+(4 * cause))。

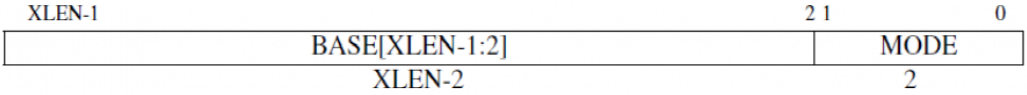


图: mtvec & stvec 寄存器

RISC-V 系统编程：M 模式下的中断相关寄存器

- 机器模式和监管模式 cause（mcause 和 scause）CSR。当处理发生异常时，CSR 中被写入一个指示导致异常的事件的代码。如果异常由中断引起，则置上中断位。“异常代码”字段包含指示最后一个异常的代码。

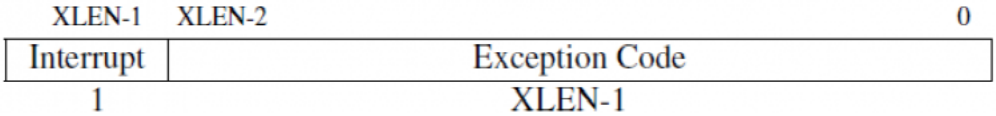
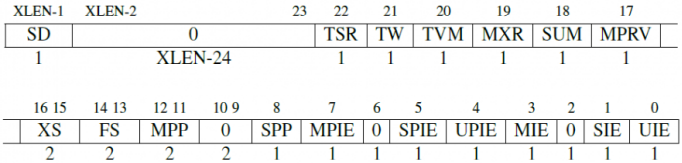


图: mcause & scause 寄存器

RISC-V 系统编程：M 模式下的异常处理

硬件执行内容

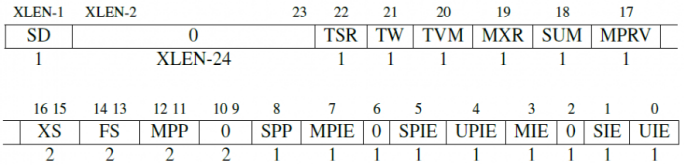
- 异常指令的 PC 被保存在 mepc 中，pc 被置为 mtvec。（对于同步异常，mepc 指向导致异常的指令；对于中断它指向中断处理后应该恢复执行的位置）
- 根据异常来源设置 mcause，并将 mtval 设置为出错的地址或者其它使用特定异常的信息字
- 把控制状态寄存器 mstatus 中的 MIE 位置零以禁用中断，并把先前的 MIE 值保留到 MPIE 中
- 发生异常前的权限模式保留在 mstatus 的 MPP 域中，再把权限模式更改为 M



RISC-V 系统编程：M 模式下的异常处理

软件执行内容 (如时钟中断)

- 保存上下文：保存各种寄存器到异常处理程序用到的特定内存 (栈) 中
- 根据异常来源进行异常或中断的处理 (如改 mtimecmp)
- mie[7] 对应于 M 模式中的时钟中断，mip 指示当前待处理的中断
- 如果 mstatus.MIE $= 1$，mie[7] $= 1$，且 mip[7] $= 1$，则可处理机器的时钟中断
- 恢复上下文：恢复各种寄存器，把 mstatus 中的 MPIE 等写到 MIE 等中
- 执行 mret 回到被打断的地方继续执行



RISC-V 系统编程：M 模式下的异常处理

软件执行内容: 可抢占式异常处理

- 在处理异常的过程中转到处理更高优先级的中断
- mepc, mcause, mtval，mstatus,mscratch 等寄存器只有一个副本
- 可抢占的中断处理程序在启动中断之前把这些寄存器保存到内存中的栈
- 然后再退出之前，禁用中断并从栈中恢复寄存器

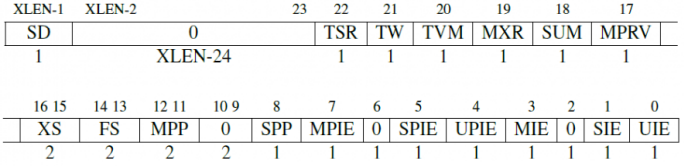
XLEN-1	XLEN-2	23	22	21	20	19	18	17
SD	0		TSR	TW	TVM	MXR	SUM	MPRV
1	XLEN-24		1	1	1	1	1	1

16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
XS	FS	MPP	0	SPP	MPIE	0	SPIE	UPIE	MIE	0	SIE	UIE
2	2	2	2	1	1	1	1	1	1	1	1	1

RISC-V 系统编程：M 模式下的隔离

特权指令隔离

- 通过设置用户模式（不可信的代码）来阻止用户执行特权指令（例如 mret）和访问控制状态寄存器（如 mstatus）
- 用户模式拒绝执行这些指令，会产生非法指令异常
- mstatus.MPP 设置为 U，然后执行 mret 指令，软件可以从 M 模式进入 U 模式
- 如果在 U 模式下发生异常，则把控制权移交给 M 模式



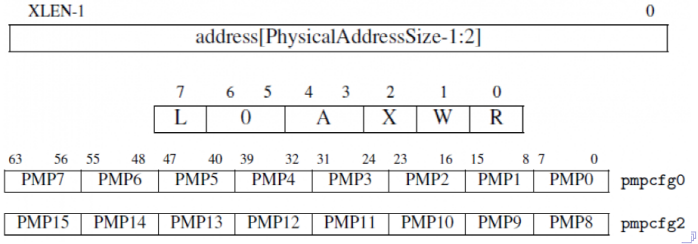
XLEN-1	XLEN-2	23	22	21	20	19	18	17
SD	0		TSR	TW	TVM	MXR	SUM	MPRV
1	XLEN-24		1	1	1	1	1	1

16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
XS	FS	MPP	0	SPP	MPIE	0	SPIE	UPIE	MIE	0	SIE	UIE
2	2	2	2	1	1	1	1	1	1	1	1	1

RISC-V 系统编程：M 模式下的隔离

内存隔离

- 实现 M 和 U 模式，物理内存保护（PMP：Physical Memory Protection），M 模式指定 U 模式可以访问的内存地址
- pmpaddr0　pmpaddrN 地址寄存器，pmpcfg 配置寄存器，地址从小到大排列
- 如果地址大于等于 PMP 地址 i，但小于 PMP 地址 i+1，则 PMP i+1 的配置寄存器决定该访问是否可以继续，如果不能将会引发访问异常。
- A: available, X: executable, W: writable, R: readable，L: lock



提纲

1. 第二节：从 OS 角度看 RISC-V
   - 主流 CPU 指令集
   - RISC-V 系统模式：概述
   - RISC-V 系统编程：M 模式
   - RISC-V 系统编程：S 模式

RISC-V 系统编程：S 模式下的隔离

- S 模式比 U 模式权限更高，但是比 M 模式权限低
- S 模式下运行的软件不能使用 M 模式的 CSR 和指令，并受到 PMP 的限制
- 支持基于页面的虚拟内存

RISC-V 系统编程：S 模式下的异常处理

- 默认情况下，所有的异常都使得控制权移交到 M 模式的异常处理程序
- M 模式的异常处理程序可以将异常重新导向 S 模式，但是这些额外的操作会减慢异常的处理速度
- RISC-V 提供一种异常委托机制，通过该机制可以选择性地将中断和同步异常交给 S 模式处理，而完全绕过 M 模式

RISC-V 系统编程：异常委托寄存器

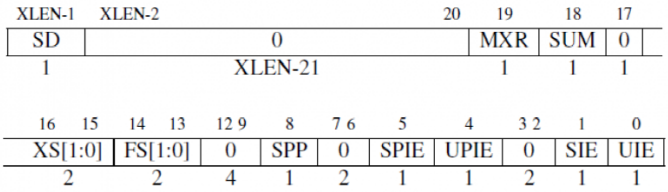
- mideleg (Machine Interrupt Delegation) CSR 控制将哪些中断委托给 S 模式处理
- 与 mip 和 mie 一样，mideleg 中的每个为对应一个异常
  - 如 mideleg[5] 对应于 S 模式的时钟中断，如果把它置位，S 模式的时钟中断将会移交 S 模式的异常处理器程序，而不是 M 模式的异常处理程序
  - 委托给 S 模式的任何中断都可以被 S 模式的软件屏蔽。sie(Supervisor Interrupt Enable) 和 sip (Supervisor Interrupt Pending) CSR 是 S 模式的控制状态寄存器，是 mie 和 mip 的子集。这两个寄存器和 M 模式下有相同的布局。sie 和 sip 中只有与由 mideleg 委托的中断对应的位才能读写，没有委派的中断对应位总是 0

RISC-V 系统编程：异常委托寄存器

- M 模式还可以通过 medeleg CSR 将同步异常委托给 S 模式。
- 另外，发生异常时控制权都不会移交给权限更低的模式
  - 例如 medeleg[15] 会把 store page fault 委托给 S 模式
  - M 模式下发生的异常总是在 M 模式下处理
  - S 模式下发生的异常总是在 M 或 S 模式下处理
  - 上述两种模式发生的异常不会由 U 模式处理 (除非有 U 模式中断处理机制)

RISC-V 系统编程：S 模式的 CSR 寄存器

- S 模式有异常处理 CSR：sepc, stvec, scause, sscratch, stval 和 sstatus，与 M 模式的寄存器相对应
- sret 返回指令和 mret 指令也类似，但是它作用于 S 模式的异常处理 CSR



图：sstatus 寄存器

RISC-V 系统编程：S 模式下的中断相关寄存器

- S 模式中断寄存器。它们是宽为 XLEN 位的读/写寄存器，用于保存待处理的中断（sip）和中断使能位（sie）CSR。

XLEN-1 10	9	8	7 6	5	4	3 2	1	0
WIRI	SEIP	UEIP	WIRI	STIP	UTIP	WIRI	SSIP	USIP
WPRI	SEIE	UEIE	WPRI	STIE	UTIE	WPRI	SSIE	USIE
XLEN-10	1	1	2	1	1	2	1	1

图: sie & sip 寄存器

RISC-V 系统编程：S 模式下的中断相关寄存器

- 机器模式和监管模式异常向量（trap-vector）基地址寄存器（mtvec 和 stvec）CSR。他们是位宽为 XLEN 的读/写寄存器，用于保存异常向量的配置，包括向量基址（BASE）和向量模式（MODE）。BASE 域中的值必须按 4 字节对齐。MODE = 0 表示所有异常都把 PC 设置为 BASE。MODE = 1 会在异步中断时将 PC 设置为 (base+(4 * cause))。

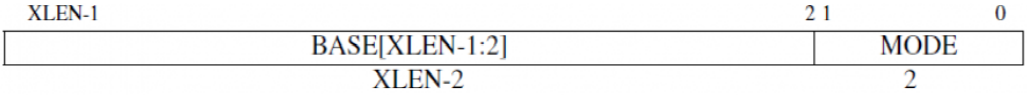
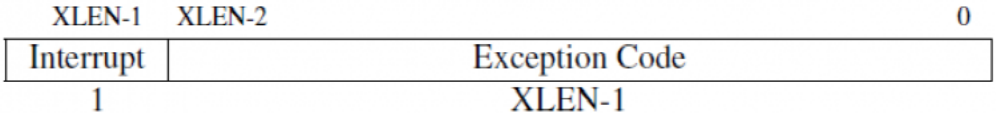


图: mtvec & stvec 寄存器

RISC-V 系统编程：S 模式下的中断相关寄存器

- 机器模式和监管模式 cause（mcause 和 scause）CSR。当处理发生异常时，CSR 中被写入一个指示导致异常的事件的代码。如果异常由中断引起，则置上中断位。“异常代码”字段包含指示最后一个异常的代码。



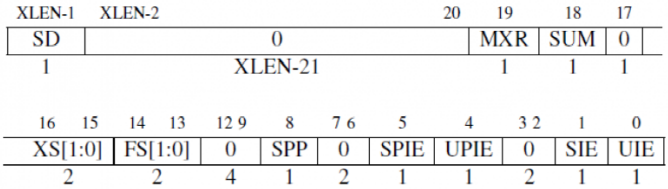
图：mcause & scause 寄存器

RISC-V 系统编程：S 模式的异常处理

硬件执行内容

hart 接受了异常，并需要委派给 S 模式，那么硬件会作为一个原子操作完成下面的状态转换

- 发生异常的指令 PC 被存入 sepc，且 PC 被设置为 stvec
- scause 根据异常设置类型，stval 被设置为出错的地址或者异常相关信息字
- 把 sstatus CSR 中的 SIE 置零，屏蔽中断，且 SIE 之前的值被保存在 SPIE 中
- 发生例外前的特权模式被保存在 sstatus 的 SPP 域，然后设置当前特权模式为 S 模式



RISC-V 系统编程：S 模式的异常处理

相关指令

- ecall：触发中断，进入更高一层的中断处理流程之中。用户态进行系统调用进入内核态中断处理流程，内核态进行 SBI 调用进入机器态中断处理流程，使用的都是这条指令。
- sret：从内核态返回用户态，同时将 pc 的值设置为 sepc。（如果需要返回到 sepc 后一条指令，就需要在 sret 之前修改 sepc 的值）
- ebreak：触发一个断点。
- mret：从机器态返回内核态，同时将 pc 的值设置为 mepc。

RISC-V 系统编程：S 模式的异常处理

操作 CSR

- 只有一系列特殊的指令（CSR Instruction）可以读写 CSR。尽管所有模式都可以使用这些指令，用户态只能只读的访问某几个寄存器。
- 为了让操作 CSR 的指令不被干扰，许多 CSR 指令都是结合了读写的原子操作。

RISC-V 系统编程：S 模式的异常处理

操作 CSR

- csrrw dst, csr, src（CSR Read Write）：同时读写的原子操作，将指定 CSR 的值写入 dst，同时将 src 的值写入 CSR。
- csrr dst, csr（CSR Read）：读取一个 CSR 寄存器。
- csrw csr, src（CSR Write）：仅写入一个 CSR 寄存器。
- csrc(i) csr, rs1（CSR Clear）：将 CSR 寄存器中指定的位清零，csrc 使用通用寄存器作为 mask，csrci 则使用立即数。
- csrs(i) csr, rs1（CSR Set）：将 CSR 寄存器中指定的位置 1，csrc 使用通用寄存器作为 mask，csrci 则使用立即数。

小结

- 了解 RISC-V 的 M-Mode 和 S-Mode 的特征
- 了解系统软件在 M-Mode 和 S-Mode 下如何访问控制计算机系统
- 了解如何在 M-Mode<–>S-Mode<–>U-Mode 之间进行切换