RISC-V的系统模式

2020年秋

RISC-V系统模式

- 什么是系统模式?
- RISC-V的特权级架构：机器模式，监管者模式
- 异常与中断处理
- 用户模式与进程隔离
- 内存管理：内存区域保护，页式虚拟内存管理
- 总结

现代处理器的多特权级模式

- 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
- 例如：处理器可以具有用户态和内核态两个特权级模式
  - 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
  - 应用程序不能访问操作系统的数据结构，不能直接使用外设资源
  - 内核态的操作系统具有完全的硬件控制能力

系统架构的编程

- 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
- 系统编程往往没有编程库的支持，需要跟底层的硬件进行交互

RISC-V的特权级模式

- 用户模式（user mode）：执行用户应用程序的模式，本学期的大部分指令都执行在用户模式
- 机器模式（machine mode）：运行最可信的代码，直接接触硬件
- 监管者模式（supervisor mode）：为现代操作系统例如Linux等提供支持
- 机器模式和监管者模式都比用户模式有着更高的权限

层级数目	支持的特权级模式	目标软件
1	M	简单的嵌入式系统
2	M, U	安全的嵌入式系统
4	M, S, U	运行操作系统

权限

- 更高权限的模式通常可以使用更低权限模式的功能，反之不成立
- 更高权限的模式有额外的功能
  - 处理终端，执行输入输出
- 处理器大部分时间运行在低权限，处理中断和异常时会将控制权移交到更高的权限模式
- 嵌入式系统运行时（runtime）和操作系统使用高权限
  - 相应外部事件
  - 支持多任务处理，任务隔离
  - 抽象和虚拟化硬件功能

RISC-V特权指令

- 特权指令以及相应的指令码
- 指令码非常少，很多功能通过控制状态寄存器（CSR：Control State Register）来实现
- 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，见后面

RV32/64 Privileged Instructions

machine-mode
supervisor-mode } trap return

supervisor-mode fence.virtual memory address

wait for interrupt

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
0001000		00010	00000	000	00000	1110011	R sret
0011000		00010	00000	000	00000	1110011	R mret
0001000		00101	00000	000	00000	1110011	R wfi
0001001		rs2	rs1	000	00000	1110011	R sfence.vma

简单嵌入式系统的机器模式 M-mode

- M模式是RISC-V中hart（硬件线程，hardware thread）可以执行的最高权限模式
- 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
- 机器模式最重要的特性是拦截和处理异常
  - 同步异常：执行期间产生，访问无效的寄存器地址，或者执行了具有无效操作码的指令
  - 异步异常：指令流异步的外部事件，中断，如鼠标点击
  - 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

硬件线程hart

- 大多数处理器核心都只有一个hart
- hart的概念与软件线程是相对的
- hart是硬件的上下文执行环境
- 如何在一个处理器核心实现多个hart?
  - 多个硬件的上下文环境，多套寄存器（共用相同的功能单元）

RISC-V的异常和中断

Interrupt / Exception mcause[XLEN-1]	Exception Code mcause[XLEN-2:0]	Description
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
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

中断时 mcause 的最高有效位置 1，同步异常时置 0，且低有效位
标识了中断或异常的具体原因。只有在实现了监管者模式时才能处理监管者模式中断和页面错误异常

同步异常

- 访问错误异常：物理内存不支持访问类型（写入ROM）时发生
- 断点异常：执行ebrak指令，或者地址与数据与调试触发器匹配时发生
- 环境调用异常：在执行ecall的时候发生，相当于系统调用
- 非法指令异常：在译码中发现无效操作码时发生
- 非对齐地址异常：有效地址不能被访问大小整除时发生

系统调用

- 应用程序不能够直接使用计算机的物理资源
- 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

断点异常

- 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?

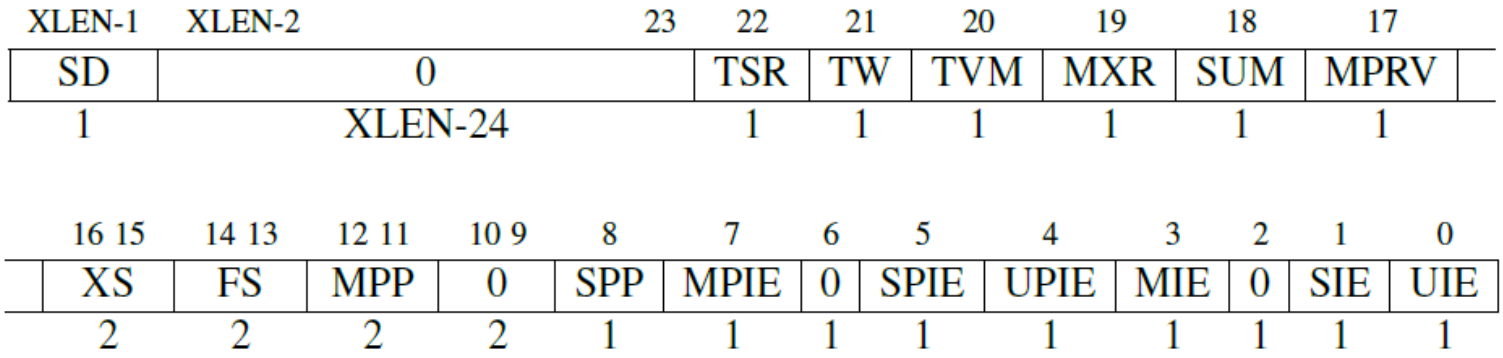
中断源

- 软件中断：通过向内存映射寄存器写入数据来触发，用户一个hart中断另外一个hart（处理器间中断）
- 时钟中断：实时技术器mtime大于hart的时间比较寄存器mtimecmp，会触发时钟中断
- 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

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
  - 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
  - mip(Machine Interrupt Pending)：正准备处理的中断
  - mtval(Machine Trap Value)：trap的附加信息：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

mstatus控制状态寄存器



- 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

M模式异常处理（硬件部分）

- 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
- 发生异常之前的权限模式保留在mstatus的MPP域中，再把权限模式更改为M

时钟中断处理代码

```
1   # save registers
2   csrrw a0, mscratch, a0  # save a0; set a0 = &temp storage
3   sw a1, 0(a0)            # save a1
4   sw a2, 4(a0)            # save a2
5   sw a3, 8(a0)            # save a3
6   sw a4, 12(a0)           # save a4
7
8   # decode interrupt cause
9   csrr a1, mcause         # read exception cause
10  bgez a1, exception      # branch if not an interrupt
11  andi a1, a1, 0x3f       # isolate interrupt cause
12  li a2, 7                # a2 = timer interrupt cause
13  bne a1, a2, otherInt    # branch if not a timer interrupt
14
15  # handle timer interrupt by incrementing time comparator
16  la a1, mtimecmp         # a1 = &time comparator
17  lw a2, 0(a1)            # load lower 32 bits of comparator
18  lw a3, 4(a1)            # load upper 32 bits of comparator
19  addi a4, a2, 1000       # increment lower bits by 1000 cycles
20  sltu a2, a4, a2         # generate carry-out
21  add a3, a3, a2          # increment upper bits
22  sw a3, 4(a1)            # store upper 32 bits
23  sw a4, 0(a1)            # store lower 32 bits
24
25  # restore registers and return
26  lw a4, 12(a0)           # restore a4
27  lw a3, 8(a0)            # restore a3
28  lw a2, 4(a0)            # restore a2
29  lw a1, 0(a0)            # restore a1
30  csrrw a0, mscratch, a0  # restore a0; mscratch = &temp storage
31  mret                    # return from handler
```

mie[7]对应于M模式中的时钟中断。控制状态寄存器mip具有相同的布局，并且它指示当前待处理的中断。

如果 mstatus.MIE = 1，mie[7] = 1，且 mip[7] = 1，则可以处理机器的时钟中断。

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和mstatus这些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

wfi指令（Wait For Interrupt）

- wfi通知处理器目前没有任何有用的工作
- 应该进入低功耗模式，知道任何使能有效的中断等待处理，即mie&mip != 0
- 可以有多种实现方式
  - 在中断待处理前都停止时钟
  - 指令也可以当做nop来执行
  - （通常在循环内使用）

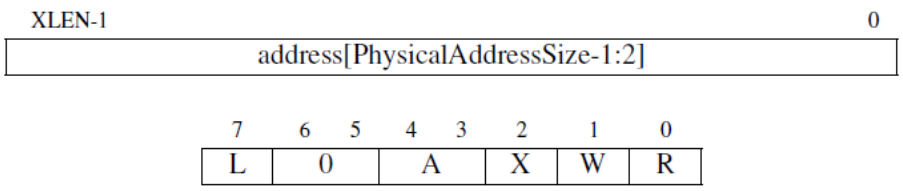
M模式中的用户模式和进程隔离

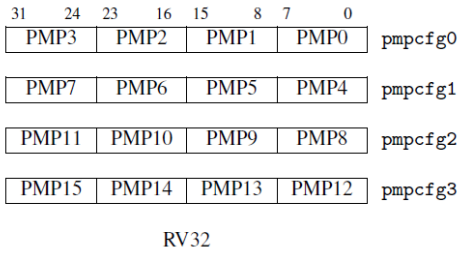
- 通过设置用户模式（不可信的代码）来阻止用户执行特权指令（例如mret）和访问控制状态寄存器（如mstatus）
- 用户模式拒绝执行这些指令，会产生非法指令异常
- mstatus.MPP设置为U，然后执行mret指令，软件可以从M模式进入U模式
- 如果在U模式下发生异常，则把控制权移交给M模式

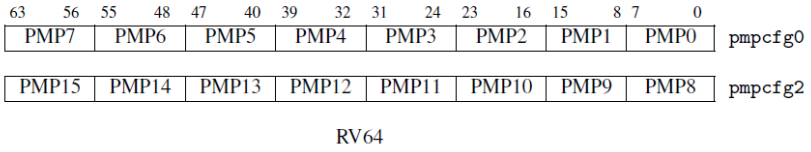
Encoding	Name	Abbreviation
00	User	U
01	Supervisor	S
11	Machine	M

用户代码的内存限制

- 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







M模式内存管理的限制

- 优点：以相对较低的成本提供了内存保护（可以被认为是一种简化的段保护机制）
- 缺点：
  - 仅支持固定数量的内存区域，无法扩展来支持复杂的应用程序
  - 这些区域必须在物理存储中连续，系统可能产生碎片化的问题
  - 不能有效去支持辅助内存的使用

S模式与页内存管理

- 更加现代化的内存管理方式：基于页面的虚拟内存
- 新的模式：监管者模式（Supervisor Mode）
- 虽然是一个可选的模式，但是为了支持现代的操作系统，例如Linux，FreeBSD或者Windows，这个模式是必不可少的
- 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

有S模式的异常处理

- 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。

异常委托寄存器（1）

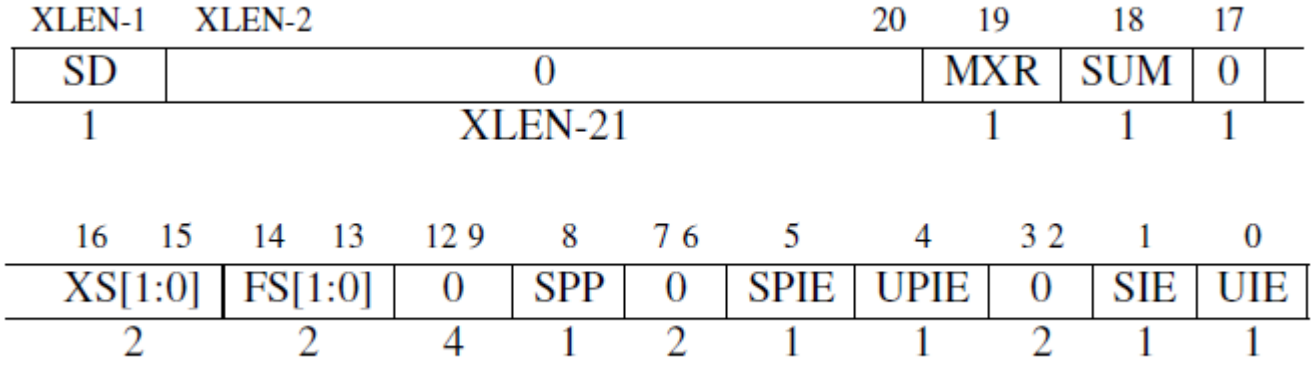
- mideleg (Machine Interrupt Delegation）CSR控制将哪些中断委托给S模式处理
- 与mip和mie一样，mideleg中的每个为对应一个异常。
  - 例如mideleg[5]对应于S模式的时钟中断，如果把它置位，S模式的时钟中断将会移交S模式的异常处理器程序，而不是M模式的异常处理程序
  - 委托给S模式的任何中断都可以被S模式的软件屏蔽。sie(Supervisor Interrupt Enable)和sip（Supervisor Interrupt Pending）CSR是S模式的控制状态寄存器，是mie和mip的子集。这两个寄存器和M模式下有相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

异常委托寄存器（2）

- M模式还可以通过medeleg CSR将同步异常委托给S模式。
  - 例如medeleg[15]会吧store page fault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
  - 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模式，或者在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

S模式的sstatusCSR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



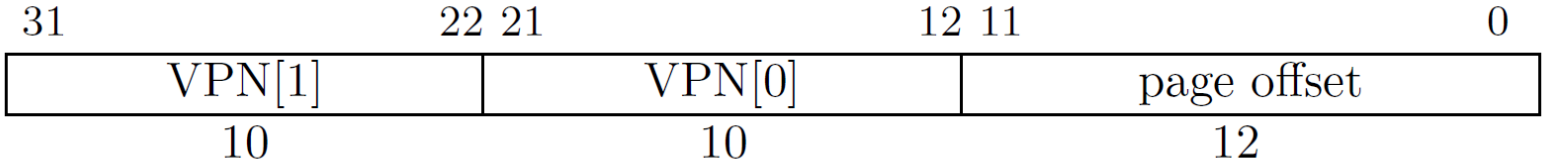
S模式的异常处理

- hart接受了异常，并需要委派给S模式，那么硬件会原子性的经历下面的状态转换
  - 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其它特定异常的信息字
  - 把sstatus CSR中的SIE置零，屏蔽中断，且SIE之前的值被保存在SPIE中
  - 发生例外时的权限模式被保存在sstatus的SPP域，然后设置当前模式为S模式

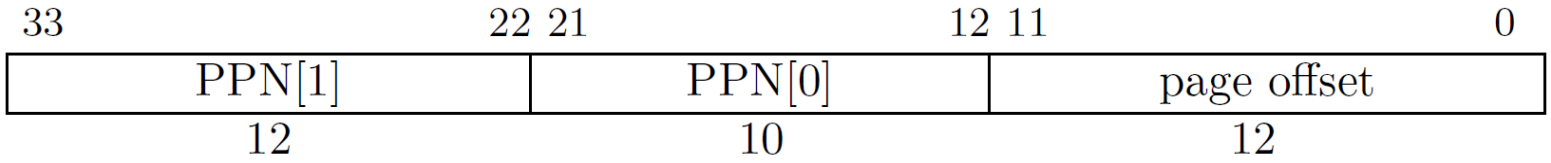
基于页面的虚拟内存

- 分页命名模式：SvX，其中X是以位为单位的虚拟地址长度
- 内存划分为固定大小的页面进行地址转换和对内存内容的保护（页面大小通常为2KB，也有大页面粒度）
- 启用分页的时候，大多数地址（包括load和store的有效地址和PC中的地址）都是虚拟地址
- 要访问物理内存，虚拟地址必须被转换为真正的物理地址
- 通过页表的结构来进行转换
- 权限位指示那些权限模式和通过哪种类型的访问可以操作这个页
- 访问未被映射的页或者访问权限不足会导致页面错误异常（page fault exception）

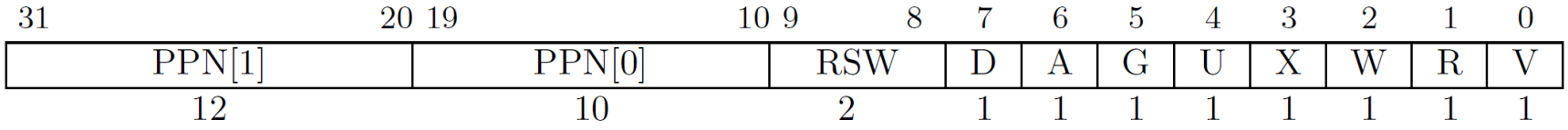
Sv32的虚拟地址与物理地址



Sv32虚拟地址

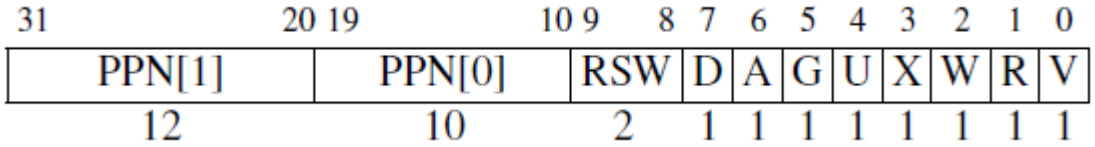


Sv32物理地址



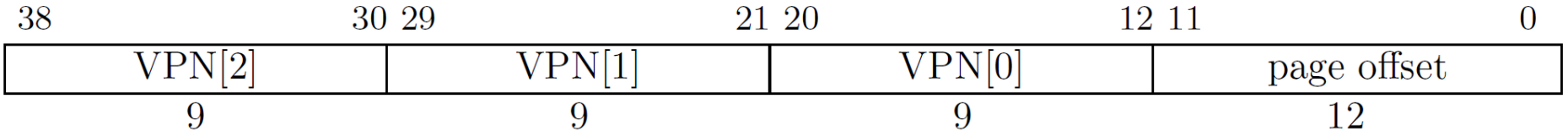
Sv32页表项

RV32 Sv32页表项（PTE: page table entry）

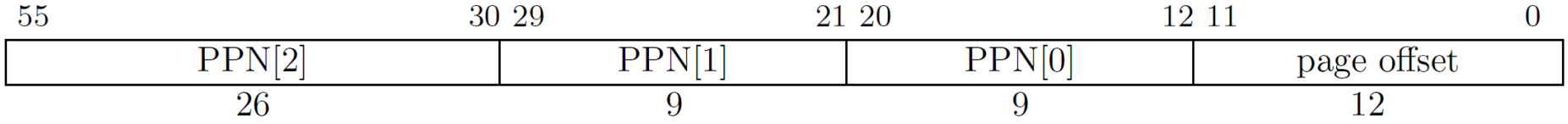


- V：有效位
- R,W,X：读，写，执行位
- U：0代表U模式不能访问，但是S模式可以；1代表U模式可以访问，S模式可以
- G：Global是否对所有地址空间有效
- A：Access，是否被访问过
- D：Dirty，是否被修改过
- RSW：操作系统使用，被硬件忽略
- PPN：物理页号，这是物理地址的一部分。如果这个页表项是一个叶子结点，那么PPN是转换后物理地址的一部分。否则PPN给出的是下一级页表的地址

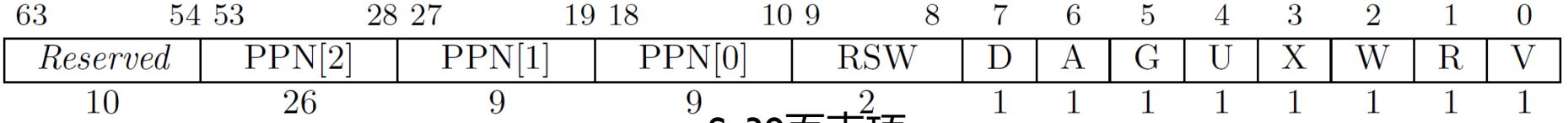
Sv39的虚拟地址与物理地址



Sv39虚拟地址



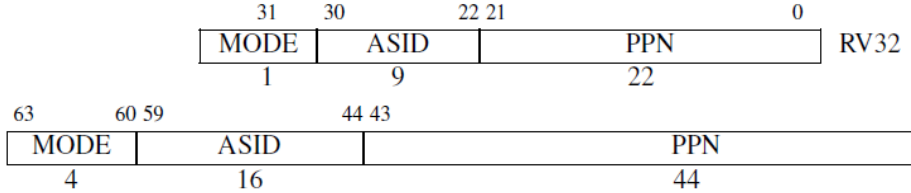
Sv39物理地址



Sv39页表项

satp寄存器

- satp（Supervisor Address Translation and Protection，监管者地址转换和保护） S 模式控制状态寄存器控制了分页系统
- ASID：Address Space Identifier，地址空间标识符（可选），用以降低上下文切换开销



PPN指向根页表的物理地址

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

RV64		
Value	Name	Description
0	Bare	No translation or protection.
8	Sv39	Page-based 39-bit virtual addressing.
9	Sv48	Page-based 48-bit virtual addressing.

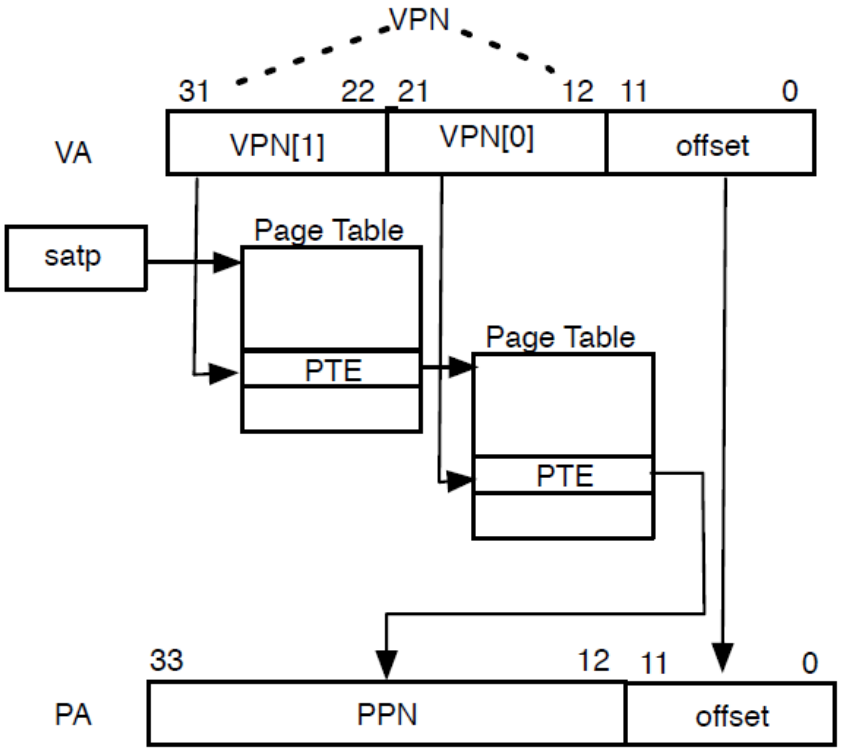
MODE域编码

satp寄存器初始化

- M模式的程序在第一次进入S模式之前会把0写入satp，以禁用分页
- 然后S模式的程序在初始化页表以后会再次进行satp寄存器的写操作

虚拟地址到物理地址的转换

- satp.PPN 给出了一级页表的基址，VA[31:22]给出了一级页号，因此处理器会读取位于地址(satp.PPN × 4096 + VA[31: 22] × 4)的页表项。
- 该 PTE 包含二级页表的基址，VA[21:12]给出了二级页号，因此处理器读取位于地址(PTE.PPN × 4096 + VA[21: 12] × 4)的叶节点页表项。
- 叶节点页表项的 PPN 字段和页内偏移（原始虚址的最低 12 个有效位）组成了最终结果：物理地址就是(LeafPTE. PPN × 4096 + VA[11: 0])



虚拟地址到物理地址的转换

1. Let a be `satp`.$ppn \times$ PAGESIZE, and let $i =$ LEVELS $- 1$.
2. Let pte be the value of the PTE at address $a + va.vpn[i] \times$ PTESIZE.
3. If $pte.v = 0$, or if $pte.r = 0$ and $pte.w = 1$, stop and raise a page-fault exception.
4. Otherwise, the PTE is valid. If $pte.r = 1$ or $pte.x = 1$, go to step 5. Otherwise, this PTE is a pointer to the next level of the page table. Let $i = i - 1$. If $i < 0$, stop and raise a page-fault exception. Otherwise, let $a = pte.ppn \times$ PAGESIZE and go to step 2.
5. A leaf PTE has been found. Determine if the requested memory access is allowed by the $pte.r$, $pte.w$, $pte.x$, and $pte.u$ bits, given the current privilege mode and the value of the SUM and MXR fields of the `mstatus` register. If not, stop and raise a page-fault exception.
6. If $i > 0$ and $pa.ppn[i - 1 : 0] \neq 0$, this is a misaligned superpage; stop and raise a page-fault exception.
7. If $pte.a = 0$, or if the memory access is a store and $pte.d = 0$, then either:
   - Raise a page-fault exception, or:
   - Set $pte.a$ to 1 and, if the memory access is a store, also set $pte.d$ to 1.
8. The translation is successful. The translated physical address is given as follows:
   - $pa.pgoff = va.pgoff$.
   - If $i > 0$, then this is a superpage translation and $pa.ppn[i - 1 : 0] = va.vpn[i - 1 : 0]$.
   - $pa.ppn[\text{LEVELS} - 1 : i] = pte.ppn[\text{LEVELS} - 1 : i]$.

虚址到物理地址转换的完整算法。va 是输入的虚拟地址，pa 是输出的物理地址。PAGESIZE 是常数 2^{12}。在 Sv32 中，LEVELS = 2 且 PTESIZE = 4；而在 Sv39 中，LEVELS = 3 且PTESIZE = 8

TLB

- 如果取指，load，store要访问多次页表，将会大大降低性能
- 所有的现代处理器都使用地址转换缓存（TLB：Translation Lookaside Buffer）来减少这种开销
- 如果操作系统修改了页表，TLB就会变得不可用
- sfence.vma通知处理器，软件可能已经修改了页表，处理器可以刷新TLB
  - rs1指示了那个虚拟地址对应的转换被修改了
  - rs2指示被修改页表的地址空间标识符（一般相当于进程）ASID
  - 如果两者都是x0，整个TLB会被刷新

与中断相关的寄存器（1）

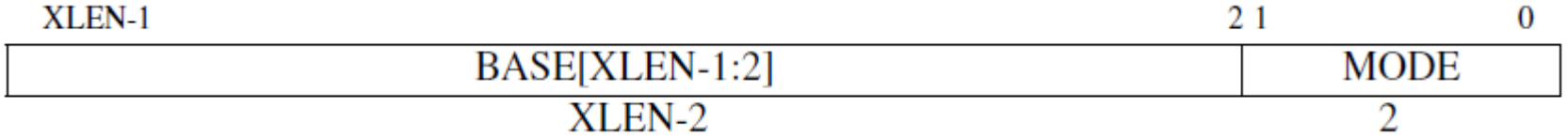
XLEN-1 12	11	10	9	8	7	6	5	4	3	2	1	0
WIRI	MEIP	WIRI	SEIP	UEIP	MTIP	WIRI	STIP	UTIP	MSIP	WIRI	SSIP	USIP
WPRI	MEIE	WPRI	SEIE	UEIE	MTIE	WPRI	STIE	UTIE	MSIE	WPRI	SSIE	USIE
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

机器模式中断寄存器。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。

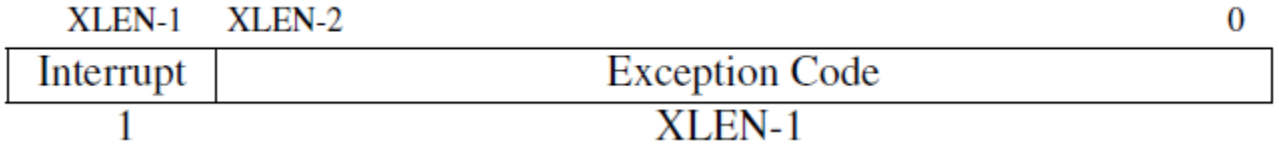
XLEN-1 10	9	8	7 6	5	4	3 2	1	0
WIRI	SEIP	UEIP	WIRI	STIP	UTIP	WIRI	SSIP	USIP
WPRI	SEIE	UEIE	WPRI	STIE	UTIE	WPRI	SSIE	USIE
XLEN-10	1	1	2	1	1	2	1	1

监管者中断寄存器。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。

与中断相关的寄存器（2）



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$(\boldsymbol{BASE} + (\boldsymbol{4} \times \boldsymbol{cause}))$。



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。

与中断相关的指令

- 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。

操作CSR（1）

- 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。
- 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。

操作CSR（2）

- csrr dst, csr（CSR Read）
  - 仅读取一个 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。

总结

- 遵循着RISC-V一贯的设计思路，RISC-V的系统体系结构也同样使用了模块化的设计
- 所有的标准的RISC-V处理器都会实现M模式，这个模式使用尽可能少的硬件来实现对于异常和中断的支持，实现用户程序和操作系统的隔离，M模式适合需要一定保护的嵌入式操作系统
- 在M模式之上RISC-V也定义了S模式，打开页式内存管理，支持虚拟内存的内存管理方式
- 系统编程的两个大的主题：异常和中断处理，内存管理

谢谢