第4季

TLB管理

本节课主要内容

- 本章主要内容
  - 为什么需要TLB
  - TLB工作原理
  - TLB别名和同名问题
  - ASID
  - TLB管理指令
  - 案例分析

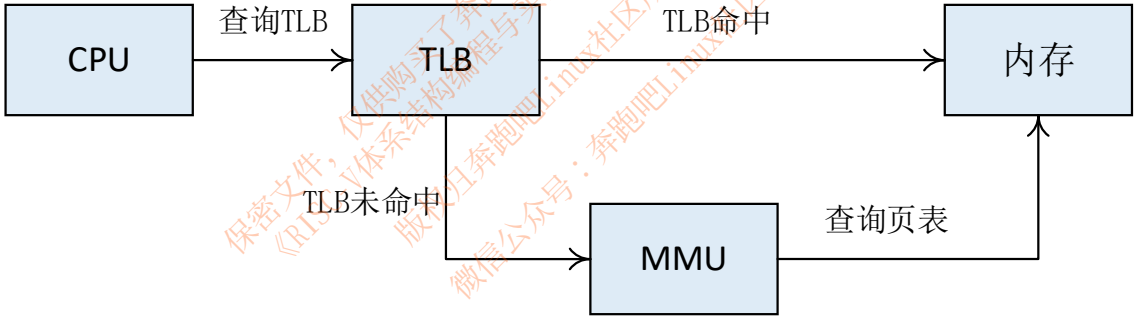
技术手册：

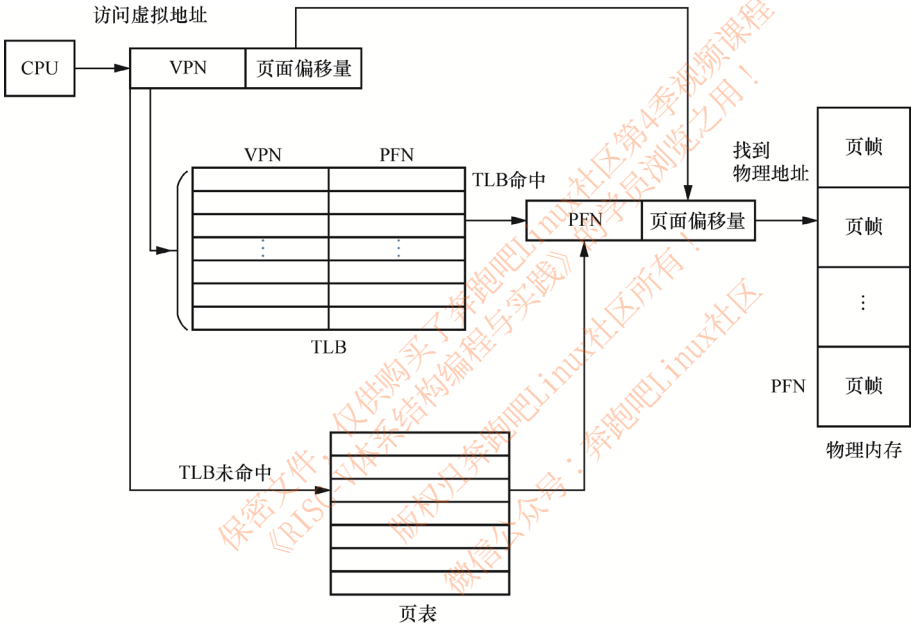
1. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203
2. SiFive U74-MC Core Complex Manual, 21G2.01.00

本节课主要讲解书上第13章内容

为什么需要TLB?

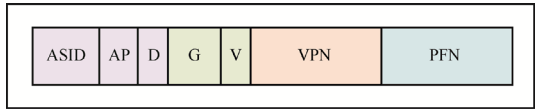
- MMU查询页表很慢，多级页表需要多次访问内存
- 把MMU的地址转换结果缓存到一个缓冲区中，这个缓冲区叫作TLB（Translation Lookaside Buffer）





RISC-V架构中TLB表项结构

- RISC-V体系结构手册中没有约定TLB项的结构



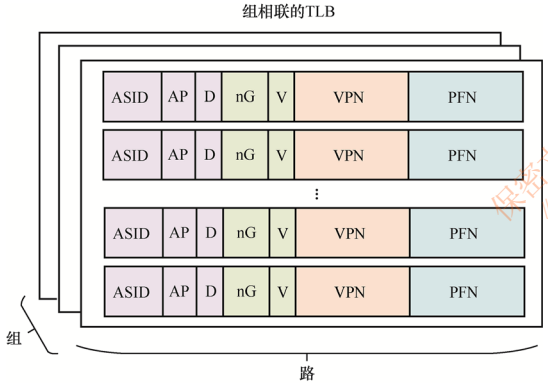
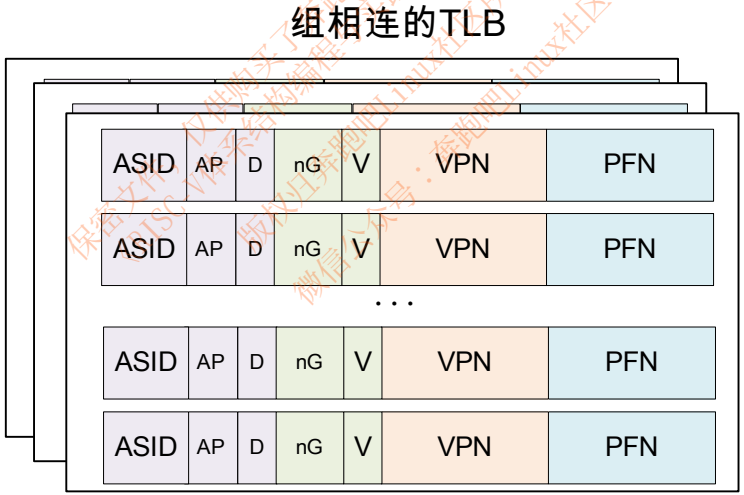
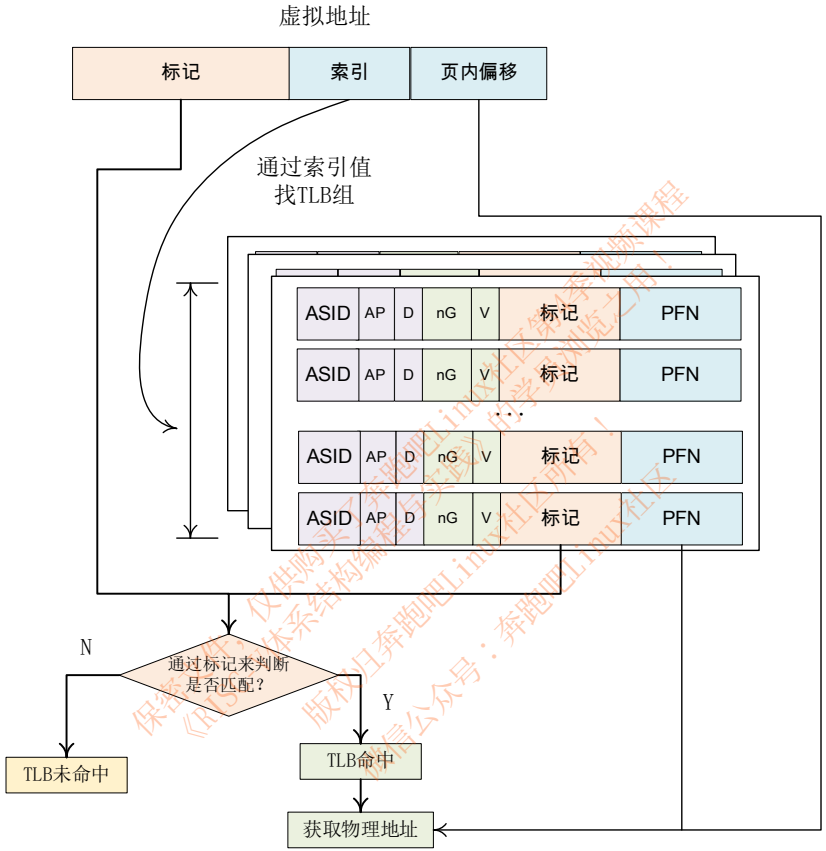


表 13.1 TLB 项的相关属性

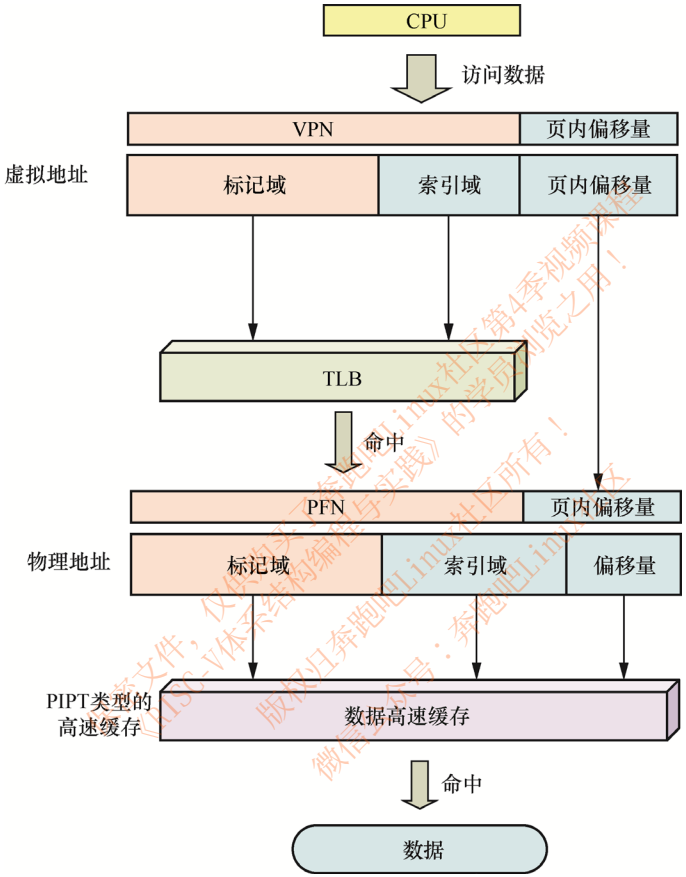
属性	描述
VPN	虚拟页帧号
PFN	物理页帧号
V	有效位
G	表示是否是全局 TLB 或者进程特有的 TLB
D	脏位
AP	访问权限
ASID	进程地址空间 ID

- TLB也支持全相连，直接映射，组相连等三种映射方式。
- SiFive U74采用两级TLB设计。类似2级cache的设计思路。
  - ✓ L1 instruction and data TLBs.
    - 40-entry fully-associative L1 I-TLB
    - 30-entry fully-associative L1 D-TLB
  - ✓ L2 unified TLB.
    - 512 direct-mapped TLB entries
- 平头哥C910也是采用两级TLB
  - L2 TLB采用4路组相联结构



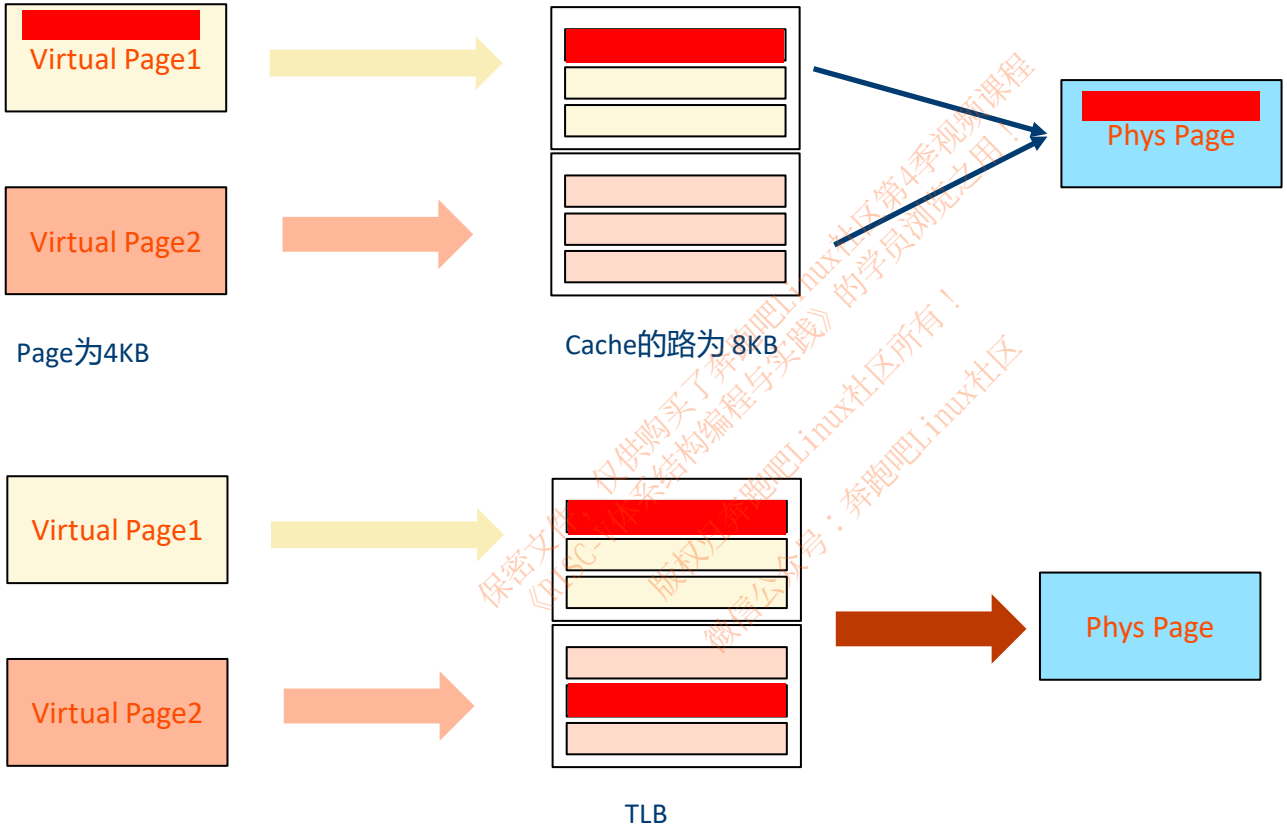


采用组相联TLB的查询过程



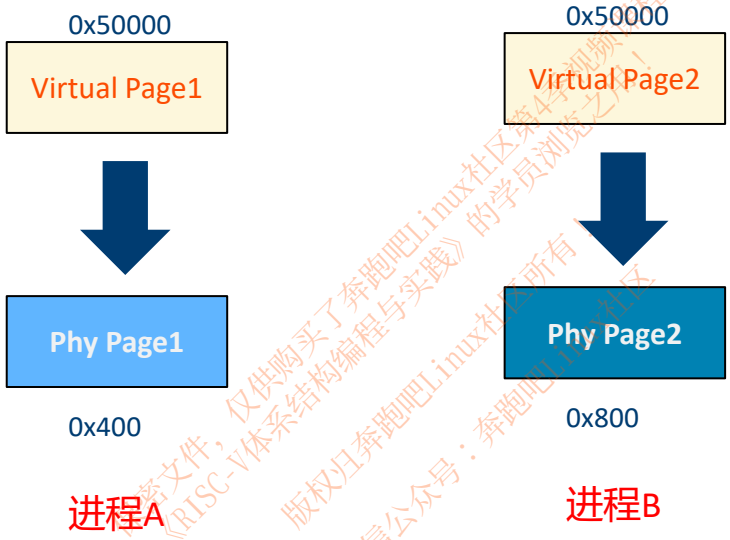
TLB与高速缓存

TLB的重名（别名）问题



虽然VP1和VP2在TLB里缓存了两个不同的TLB entry，但是TLB entry里的PFN都是指向Phys Page，所以，不会产生重名问题

TLB的同名问题

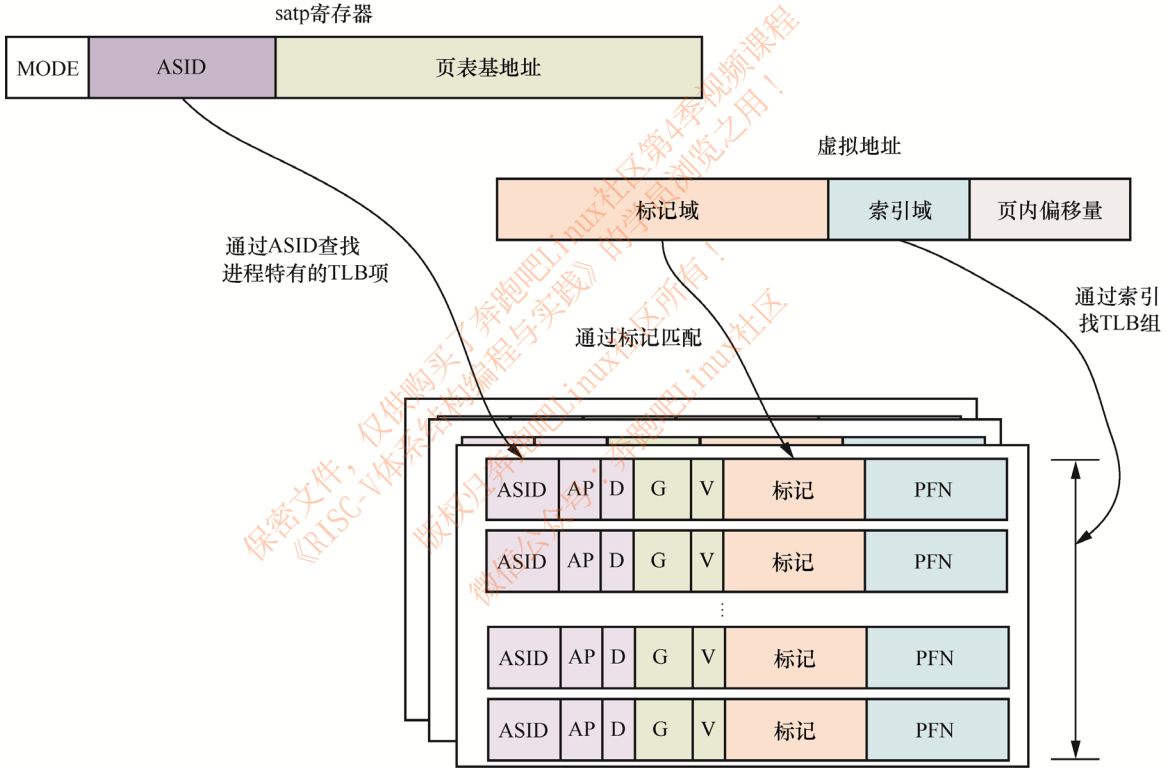


ASID（Address Space Identifier）

- 全局类型的TLB：内核空间是所有进程共享的空间。
- 进程独有类型的TLB：用户地址空间是每个进程独立的地址空间。

- ASID机制用来 实现进程独有类型的TLB。
- RISC-V的ASID存储在satp寄存器中，一共16位宽，最多支持65 536个ASID

ASID怎么用?



TLB维护指令

- RISC-V架构提供了一条TLB维护指令SFENCE.VMA，融合了内存屏障与刷新TLB的指令
  - ✓ 内存屏障：保证在屏障之前的存储操作与屏障之后的读写操作的执行次序。这里主要指的是对虚拟内存管理中的相关数据的读写操作，例如，对页表的读写操作等。
  - ✓ 刷新TLB：刷新本地处理器上与地址转换相关的高速缓存，如TLB等
- SFENCE.VMA指令的作用范围仅限于本地处理器

```
SFENCE.VMA rs1 rs2
```

rs1：用来指定虚拟地址
rs2：用来指定ASID

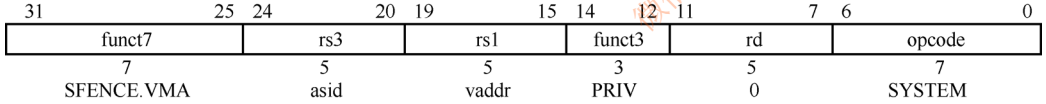


表 13.2 SFENCE.VMA 指令的参数

参数组合	描述
rs1=x0 和 rs2=x0	rs1=x0 表示对页表相关数据结构的所有读和写操作进行排序，这针对所有地址空间。 rs2=x0 表示会使所有地址空间中与地址转换相关的高速缓存（如 TLB 等）失效
rs1=x0 和 rs2!=x0	rs1=x0 表示对页表相关数据结构的所有读和写操作进行排序，这特指 ASID 为 rs2 的进程的地址空间。 rs2 表示进程的 ASID，这条指令会使与该进程对应的地址转换相关的高速缓存（如 TLB 等）失效
rs1!=x0 和 rs2=x0	rs1 表示虚拟地址，这条指令会对这个虚拟地址对应的页表的相关数据结构的所有读和写操作进行排序，这里仅仅针对该虚拟地址对应的所有地址空间。 rs2=x0 表示会使与所有地址空间的地址转换相关的高速缓存（如 TLB 等）失效
rs1!=x0 和 rs2!=x0	rs1 表示虚拟地址，这条指令会对这个虚拟地址对应的页表相关数据结构的所有读和写操作进行排序，这特指 ASID 为 rs2 的进程的地址空间。 rs2 表示进程的 ASID，这条指令会使与该进程中 rs1 指定的虚拟地址所对应的地址转换相关的高速缓存（如 TLB 等）失效

例子

例子13-1：刷新进程p在本地处理器中全部的TLB，其中参数asid为进程p的ASID

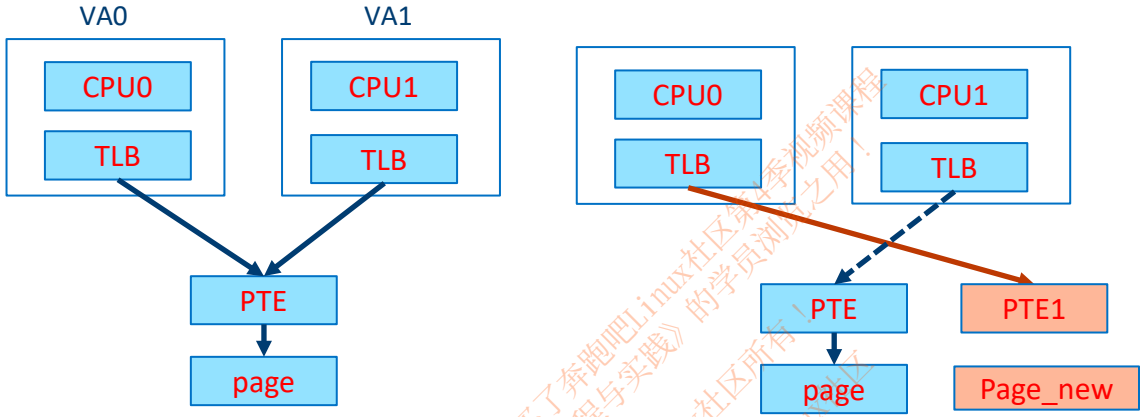
```
void local_flush_tlb_all_asid(unsigned long asid)
{
  __asm__ __volatile__ ("sfence.vma x0, %0"
            :
            : "r" (asid)
            : "memory");
}
```

例子13-2：刷新进程p在本地处理器中一个地址范围的TLB，其中参数asid为进程p的ASID

```
void local_flush_tlb_range_asid(unsigned long start, unsigned long size, unsigned long
asid)
{
  unsigned long i;

  for (i = 0; i < size; i += PAGE_SIZE) {
      __asm__ __volatile__("sfence.vma %0, %1"
                :
                : "r"(start + i), "r"(asid)
                : "memory");
  }
}
```

TLB广播（TLB shootdown）

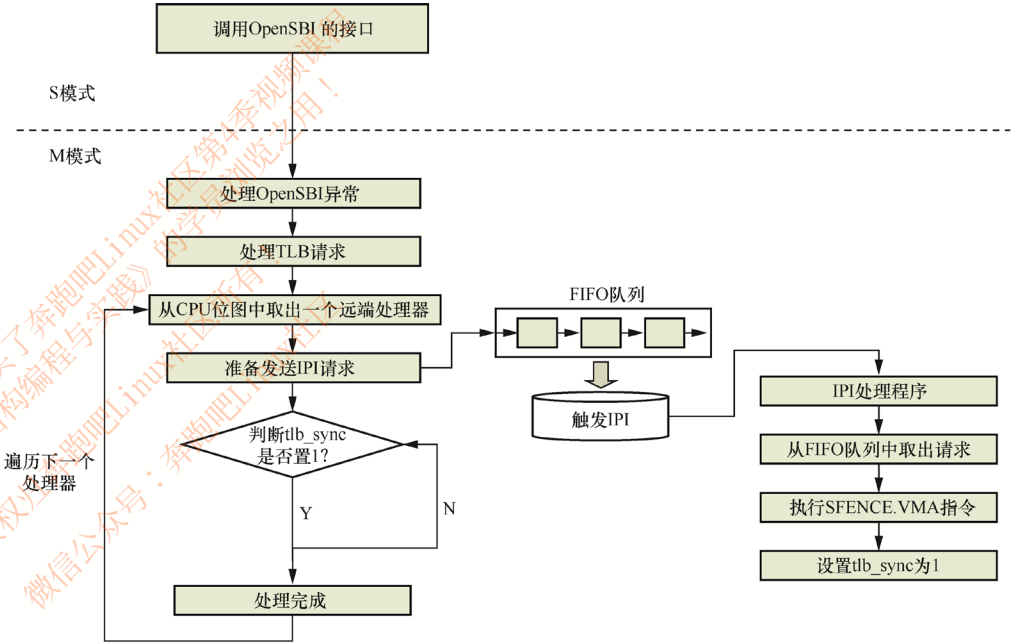


- SFENCE.VMA指令只作用于本地处理器，如果在多处理器系统中刷新TLB，则需要使用TLB广播。
- TLB shootdown：A request to another processor to remove a newly invalid TLB entry

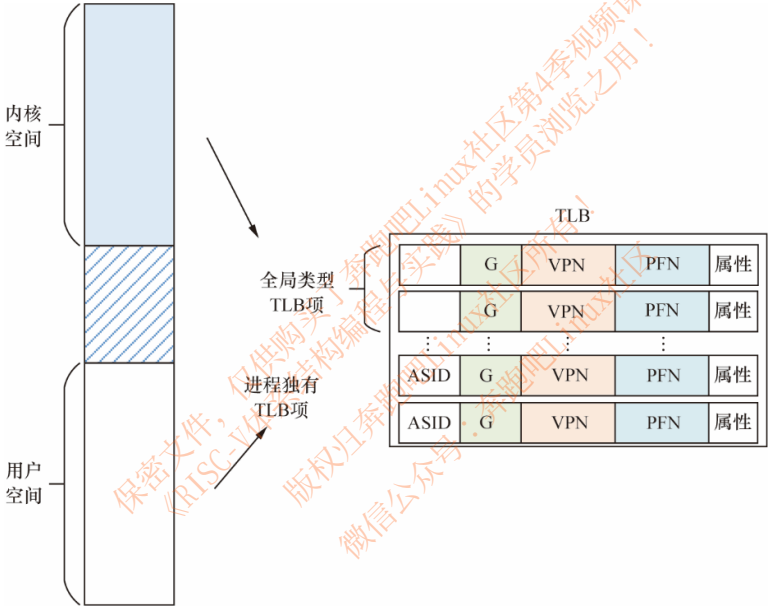
- 有些处理器体系结构在芯片内部实现TLB广播协议，如ARM的DVM事务（Distributed Virtual Memory Transaction）协议
- RISC-V架构需要使用软件触发IPI才能完成TLB广播

OpenSBI提供了两个刷新进程TLB的接口。
SBI_EXT_RFENCE_REMOTE_SFENCE_VMA：用来刷新全部进程的TLB。
SBI_EXT_RFENCE_REMOTE_SFENCE_VMA_ASID：用来刷新指定进程的TLB。

在RISC-V体系结构中实现TLB广播的步骤如下。
（1）在本地处理器中执行SFENCE.VMA指令。
（2）依次向系统中其他处理器触发IPI。
（3）其他处理器在IPI处理函数中执行SFENCE.VMA指令。
（4）其他处理器发送信号给本地处理器，告知IPI处理已经完成。

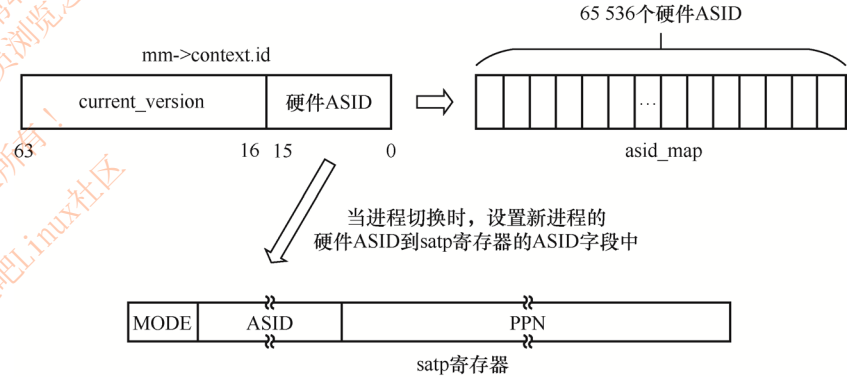


案例分析1：TLB在Linux内核中的应用



案例分析2：ASID在Linux内核中的应用

- 硬件ASID通过位图来分配和管理。
- 新创建的进程，使用位图机制来分配一个空闲的ASID，把这个ASID填充到satp寄存器里。
- 当系统中ASID加起来超过硬件最大值时，会发生溢出，需要冲刷全部TLB，然后重新分配ASID
- 当切换进程的时候，需要把进程持有的硬件 ASID 写入satp寄存器里。
- 在Linux内核里，进程切换出去之后会把ASID存储在mm数据结构的context字段里面。当进程再切换回来的时候，把ASID设置到satp寄存器里。



案例分析3：Linux内核中的TLB维护操作

表 13.3　Linux 内核中管理 TLB 的接口函数

接口函数	描述
flush_tlb_all()	使所有处理器上的整个 TLB（包括内核地址空间和用户地址空间的 TLB）失效
flush_tlb_mm(mm)	使一个进程中整个用户地址空间的 TLB 失效
flush_tlb_range(vma, start, end)	使进程地址空间的某个虚拟地址区间（从 start 到 end）对应的 TLB 失效
flush_tlb_kernel_range(start, end)	使内核地址空间的某个虚拟地址区间（从 start 到 end）对应的 TLB 失效
flush_tlb_page(vma, addr)	使虚拟地址（addr）所映射页面的 TLB 页表项失效
local_flush_tlb_all()	使本地 CPU 对应的整个 TLB 失效

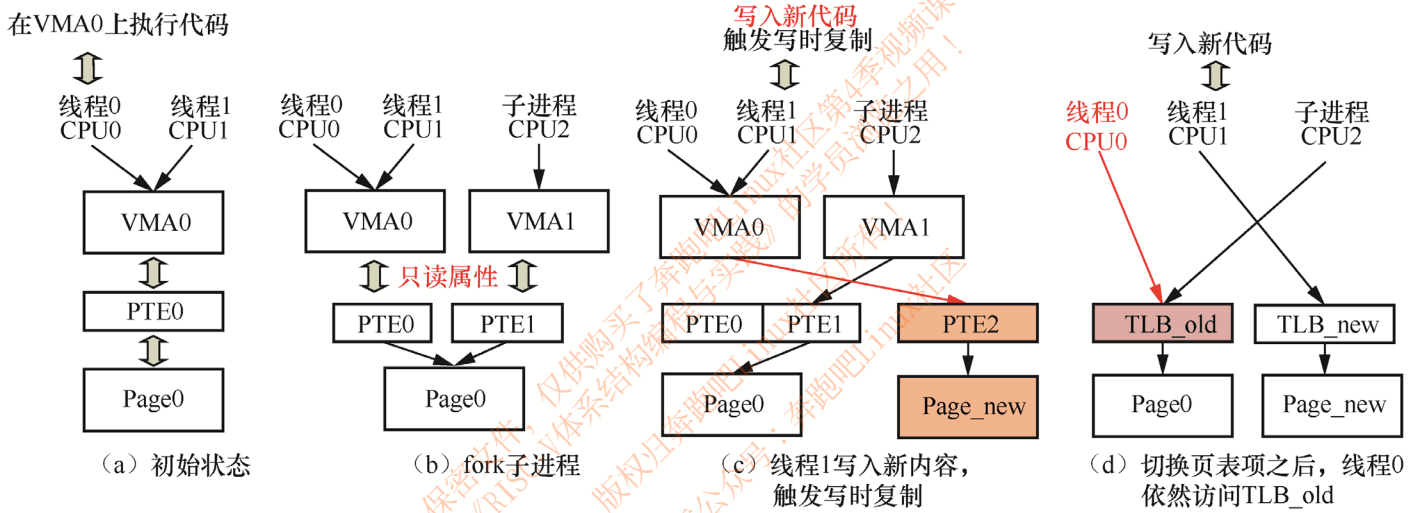
案例分析4：BBM机制

- 在多核系统中，多个虚拟地址可以同时映射到同一个物理地址，出现为同一个物理地址创建了多个TLB项的情况，而更改其中一个页表项会破坏缓存一致性以及内存访问时序等，从而导致系统出问题。

- BBM（Break-Before-Make，先断开后更新）

- BBM机制工作流程：
  - ✓ 使用一个失效的页表项来替换旧的页表项，执行一条内存屏蔽指令。
  - ✓ 执行SFENCE.VMA指令来刷新对应的TLB。发送IPI中断到其他CPU上，让其他CPU也刷新TLB。
  - ✓ 写入新的页表项，执行内存屏障指令，保证写入操作被其他CPU观察者看到。

假设主进程有两个线程：线程0和线程1，线程0运行在CPU0上，线程1运行在CPU1上，它们共同访问一个虚拟地址。这个VMA映射到Page0上。



解决方案：
根据BBM机制，C对应的步骤需要分解成如下几个步骤。
（1）在切换页表项之前，CPU1把旧的页表项内容清除掉。
（2）刷新对应的TLB，发送广播到其他CPU上。
（3）设置新的页表项（PTE2）。
（4）对于线程1来说，VMA的虚拟地址映射到Page_new之后才能往Page_new中写入新代码。

小结：RISC-V架构规范与TLB

- RISC-V架构没有约定TLB表项的内容。
- RISC-V架构没有约定TLB映射方式：直接映射，全相联，组相联，多级TLB等
- RISC-V架构没有约定TLB refill是硬件实现还是软件实现
- RSIC-V架构约定了刷新TLB的指令：sfence.vma
- RISC-V架构中TLB广播通过IPI中断来实现
- RISC-V架构的ASID机制，16位的ASID