分页: TLB

分页在取指、取数会有额外的一次内存访问,系统变慢

如何加速虚拟地址转换,避免额外的一次内存访问

TLB(translation-lookaside buffer),地址转换缓存,硬件缓存

TBL的基本算法

一个大体框架, 说明硬件如何处理虚拟地址转换.
假定使用线性页表和硬件管理的 TLB

首先从虚拟地址中提取页号(VPN),然后检查 TLB 是否有该 VPN的转换映

如果有, 则 TLB 命中, 接下来从相关的 TLB 项中取出页帧号(PFN),
与虚拟地址中的偏移组成物理地址,并访问内存(假定保护检查通过)

TLB 未命中, 则硬件访问页表来寻找转换映射.假设该虚拟地址映射有效,
而且有访问权限,用该转换映射更新 TLB,最后再重新尝试指令.

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)    // TLB Hit
   if (CanAccess(TlbEntry.ProtectBits) == True)
      Offset = VirtualAddress & OFFSET_MASK
      PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
      AccessMemory(PhysAddr)
   else
      RaiseException(PROTECTION_FAULT)
else    // TLB Miss
   PTEAddr = PTBR + (VPN * sizeof(PTE))
   PTE = AccessMemory(PTEAddr)
   if (PTE.Valid == False)
      RaiseException(SEGMENTATION_FAULT)
   else if (CanAccess(PTE.ProtectionBits) == False)
      RaiseException(PROTRCTION_FAULT)
   else
      TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
      RetryInstruction()
```

TLB 和其他缓存相似,在一般情况下,如果转换映射在缓存中命中,只会增加很少的开销
如果 TLB未命中,就会带来很大的分页开销.必须访问页表来查找转换映射,导致一次额外的内存引用.
如果这经常发生,程序的运行就会显著变慢.相对于大多数 CPU 指令,内存访问开销很大,TLB 未命中导致更多内存访问

示例:访问数组

一个简单的循环,访问数组中的每个元素.
简单起见,假设循环产生的内存访问只针对数组,忽略 i 和 sum

```
int sum = 0;
for (i = 0; i < 10; i++)
{
   sum += a[i];
}
```

10次访问操作中TLB的表现为:[a0]未命中、命中、命中、
[a3]未命中、命中、命中、命中、[a7]未命中、命中、命中

TLB 命中率70%,在每次访问新的一页时,会导致未命中.
如果增加页的大小,则数组被分割成的页数更少,TLB命中率会提高

得益于空间局部性,数组元素被紧密存放在相邻几页中.
TLB还是提高了性能,只有对页中第一个元素的访问才会导致TLB未命中

由于时间局部性,即在短时间内程序再次访问数组,此时TLB已缓存所有的数据,命中率会很高.
TLB的成功依赖于空间和时间局部性

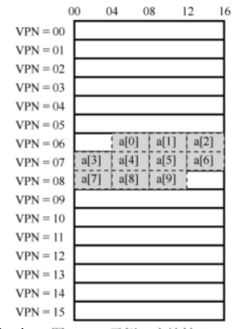


图 19.2 示例：小地址空间中的一个数组

缓存是计算机系统中最基本的性能改进技术之一. 硬件缓存, 无论是(Cache)指令,数据, 还是(TLB)地址转换,
其背后的思想是利用指令和数据引用的局部性. 局部性通常有两种.
时间局部性: 指最近访问过的指令或者数据可能很快会再次访问,比如循环中的循环变量和指令(比较, ++)
空间局部性: 指程序访问内存地址x时,可能很快会访问临近x的内存,比如遍历某个数据,访问一个接一个的元素.
但是缓存不能无限大,装下所有的数据. 因为存在基本的定律,想要快速地访问就必须小,
光速和其他物理限制,大的缓存注定慢,所以只能用小而快的缓存.

来处理TLB未命中

复杂指令集计算机: 硬件全权处理 TLB未命中, 比如x86.

硬件通过页表基址寄存器,知道页表在内存中的确切位置,以及页表的确切格式

发生未命中时,硬件会“遍历”页表,找到正确的页表项,
取出想要的转换映射,用它更新 TLB,并重试该指令

精简指令集计算机:软件操作系统管理 TLB

TLB 未命中时,硬件抛出一个异常, 这会暂停当前的指令流,将特权级提升至内核模式,跳转至陷阱处理程序(trap handler)

陷阱处理程序是操作系统的一段代码,用于处理 TLB未命中.查找页表中的转换映射,
然后用特别的“特权”指令更新 TLB,并从陷阱返回.硬件会重试该指令(导致 TLB命中)

TLB miss 的从陷阱返回指令不同于系统调用的从陷阱返回,返回后不是执行下一条指令,硬件必须从导致陷阱的当前指令继续执行,
然后命中TLB. 根据陷阱或异常的原因，系统在陷入内核时必须保存不同的程序计数器，以便将来能够正确地继续执行

TLB miss的异常处理代码需要小心避免引起 TLB miss的无限递归.
解决方案比如可以把 TLB 未命中陷阱处理程序直接放到物理内存中[没有映射过(unmapped),不用经过地址转换];
或者在 TLB中保留一些项,记录永久有效的地址转换,并将其中一些永久地址转换槽块留给处理代码本身,
这些被监听的(wired)地址转换总是会命中 TLB

软件管理主要优势是灵活性:操作系统可以用任意数据结构来实现页表,不需要改变硬件;
另一个优势是简单性,从TLB 控制流中可以看出(图19.3第11行对比图19.1的11～19 行),
硬件不需要对未命中做太多工作,它只抛出异常,操作系统的未命中处理程序会负责剩下的工作

TLB 的内容

硬件上, TLB 是全相联的(fully-associative)缓存

一条地址映射可能存在 TLB 中的任意位置,
硬件会并行地查找 TLB

一条 TLB 项的内容可能是: VPN | PFN | 其他位, VPN 和 PFN 同时存在于TLB中

“其他位"通常有一个有效(valid)位, 保护(protection)位,脏位(dirty),地址空间标识符(address-space identifier)

TLB 的有效位指出 TLB项是不是有效的地址映射.例如系统启动时,所有的 TLB项通常被初始化为无效状态,
因为还没有地址映射被缓存在这里.一旦启用虚拟内存,程序开始运行,访问自己的虚拟地址,TLB 就会慢慢地被填满

如果一个页表项(PTE)被标记为无效,就意味着该页并没有被进程申请使用,正常运行的程序不应该访问该地址.
当程序试图访问这样的页时,就会陷入操作系统,操作系统会杀掉该进程

上下文切换对 TLB的处理

TLB 中包含的虚拟到物理的地址映射只对当前进程有效,对其他进程没有意义.
在发生进程切换时,必须注意确保即将运行的进程不要误读了之前进程的地址映射

例子, 进程P1正在运行, TLB中缓存了P1的10号虚拟页映射到了100号物理帧.
还有一个进程P2,P2 的 10 号虚拟页映射到 170 号物理帧. 操作系统不久后决定进行一次上下文切换, 运行P2.
问题很明显,VPN10 被映射成了 PFN100(P1)和 PFN170(P2),硬件分不清哪个项属于哪个进程

关键问题：进程切换时如何管理 TLB 的内容?
上一个进程在 TLB 中的地址映射对于即将运行的进程是无意义的

一种方法是在上下文切换时,简单地清空 TLB

软件管理TLB,可以在发生上下文切换时,通过一条显式(特权)指令来完成

硬件管理TLB,则可以在页表基址寄存器内容发生变化时清空TLB.
注意,在上下文切换时,操作系统必须改变页表基址寄存器(PTBR)的值

清空TLB,每次进程运行,当它访问数据和代码页时,都会触发TLB未命
中.如果操作系统频繁地切换进程,这种开销会很高

为了减少开销,一些系统增加了硬件支持,实现跨上下文切换的TLB 共享
比如在TLB中添加了一个地址空间标识符(Address Space Identifier,ASID).
可以把 ASID看作是进程标识符(PID),但通常比PID 位数少(ASID 一般是8位)

有了ASID,TLB 可以同时缓存不同进程的地址空间映射.
硬件也需要知道当前是哪个进程正在运行,以便进行地址转换.
操作系统在上下文切换时,必须将某个特权寄存器设置为当前进程的 ASID

表 19.2 添加 ASID 字段后的 TLB

VPN	PFN	有效位	保护位	ASID
10	100	1	rwx	1
—	—	0	—	—
10	170	1	rwx	2
—	—	0	—	—

还有一种情况,TLB 中某两项非常相似,属于两个不同进程的两项,
将两个不同的 VPN 指向了相同的物理页.
如果两个进程共享同一物理页(例如代码段的页),就可能出现这种情况

表 19.3 包含相似两项的 TLB

VPN	PFN	有效位	保护位	ASID
10	101	1	r-x	1
—	—	0	—	—
50	101	1	r-x	2
—	—	0	—	—

实际系统的 TLB表项

MIPS R4000, 软件管理TLB,支持 32位的地址空间,页大小 4KB, 虚拟地址中,预期会有20位的VPN和12位的偏移量.
但实际上TLB中只有 19位的VPN.因为用户地址只占地址空间的一半(剩下的留给内核).
VPN转换成最大24位的物理帧号(PFN),因此可以支持最多有 64GB物理内存(2^24 4KB内存页)

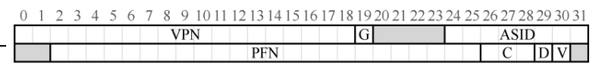


图 19.4 MIPS 的 TLB 项

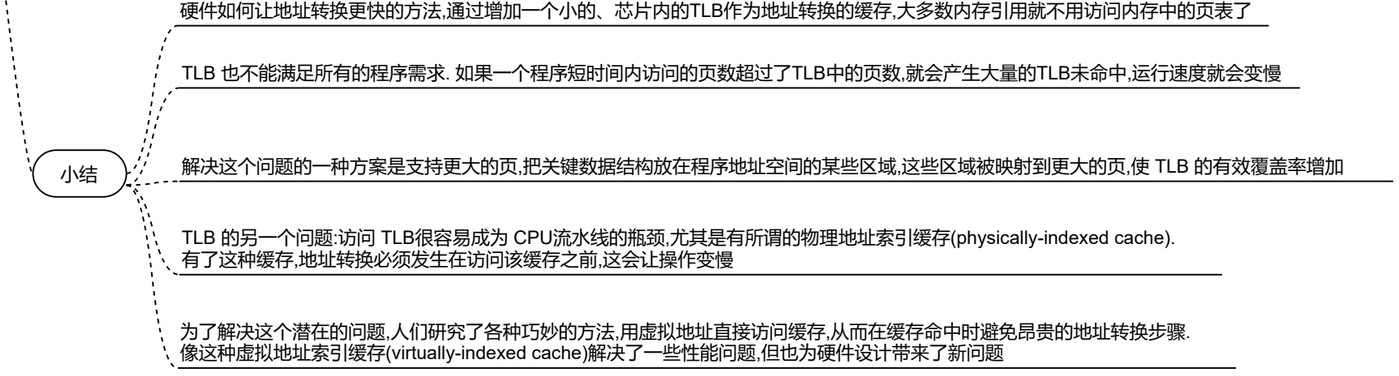
全局位: G, 指示这个页是不是所有进程全局共享的, 比如代码. 如果为1 就会忽略 ASID

一致性位(Coherence,C),决定硬件如何缓存该页;
脏位(D), 表示该页是否被写入新数据;
有效位(V),告诉硬件该项的地址映射是否有效

页掩码(page mask)字段,没在图中展示,用来支持不同的页大小

MIPS的TLB通常有32项或64项,大多数给用户进程使用,有一小部分留给操作系统.
操作系统可以设置一个被监听的寄存器,告诉硬件需要为自己预留多少TLB槽.
这些保留的转换映射用于关键时候它要使用的代码和数据,例如TLB 未命中处理程序,这些时候TLB未命中可能会导致问题

MIPS的TLB是软件管理的,所以系统需要提供一些更新TLB的指令.
TLBP:查找指定的转换映射是否在TLB中;TLBR:将TLB中的内容读取到指定寄存器中;
TLBWI:替换指定的TLB项;TLBWR:用来随机替换一个TLB项.


小结
硬件如何让地址转换更快的方法,通过增加一个小的、芯片内的TLB作为地址转换的缓存,大多数内存引用就不用访问内存中的页表了
TLB 也不能满足所有的程序需求. 如果一个程序短时间内访问的页数超过了TLB中的页数,就会产生大量的TLB未命中,运行速度就会变慢
解决这个问题的一种方案是支持更大的页,把关键数据结构放在程序地址空间的某些区域,这些区域被映射到更大的页,使 TLB 的有效覆盖率增加
TLB 的另一个问题:访问 TLB很容易成为 CPU流水线的瓶颈,尤其是有所谓的物理地址索引缓存(physically-indexed cache).
有了这种缓存,地址转换必须发生在访问该缓存之前,这会让操作变慢
为了解决这个潜在的问题,人们研究了各种巧妙的方法,用虚拟地址直接访问缓存,从而在缓存命中时避免昂贵的地址转换步骤.
像这种虚拟地址索引缓存(virtually-indexed cache)解决了一些性能问题,但也为硬件设计带来了新问题