第五章 开发存储器

1. 介绍存储器与其开发

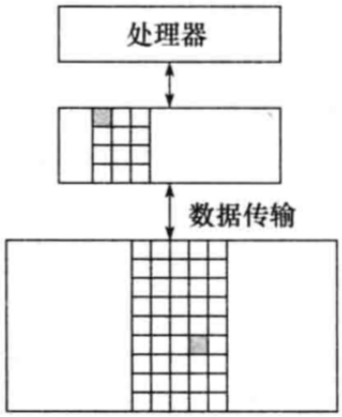
(1) 为什么不都用同样高速的存储器，而要进行复杂的分类呢？因为数据的重要程度不一样，有的是放在手边马上就要用到的数据，有的是要存储起来的数据，距离cpu的核心的距离也不相同。

(2) 通过什么原则来确定一个数据是否将要被访问：局部性原理

时间局部性、空间局部性。利用局部性原理，我们可以把计算机存储器组织称为存储器层次结构。存储器的层次结构由不同速度和容量的多级存储器构成，但是数据每次只能在相邻的两个层次之间进行复制。

时间局部性：某个数据项在被访问之后可能很快被再次访问的特性。
空间局部性：某个数据项在被访问之后，与其地址相近的数据项可能很快被访问的特性。

> 块(block)或行(line)，表示两级层次结构中存储信息交换的最小单元。



块(或者行)在存储器层次结构中传输示意图

如果处理器寻找的数据放在高层存储器的某个块中，就是*命中*，没找到就是*缺失*，如果发生了缺失的话，从低级一些的存储器中获取被寻找的数据。命中率可衡量存储器层次结构是否高性能。缺失代价(miss penalty)是将对应的块从低层存储器替换到高层存储器中以及信息块传送给存储器的时间之和。

我们知道，距离cpu越远，存储器的容量越大，同样读取的速度会越慢（参考书上图5-3，金字塔结构）cpu首先在最近的存储器中寻找数据，如果没有明珠，就需要访问容量大但是速度慢的低层存储器层次，这样效率显然降低了。

2. 存储器技术

现在存储器结构中的主要的四种技术如下表所示，本章节就是讨论这些技术。

存储器技术	典型访问时间（ns）	2012 年每 GiB 的价格（美元）
SRAM	0.5 ~ 2.5	500 ~ 1 000
DRAM	50 ~ 70	10 ~ 20
Flash	5 000 ~ 50 000	0.75 ~ 1.00
磁盘	5 000 000 ~ 20 000 000	0.05 ~ 0.10

(1) SRAM

SRAM组织成存储阵列结构，有一个读写端口。所谓静态存储，Static意味着对任何数据的访问时间都是固定的。SRAM不需要刷新，并且访问时间与周期时间非常接近。

(2) DRAM

SRAM是掉电易失(加电数据可以保持)，但是DRAM使用了电容来保存电荷的数据， 不过也不稳定，需要（周期性）刷新来保存数据。刷新就是读出内容然后写回。

> Q：如何做到对保存的数据进行读取或者写入?
> A：使用一个晶体管对它进行访问。（这也是为什么SRAM密度没有DRAM大，因为DRAM每存1位只用一个晶体管）

DRAM采用了一种两级译码结构，可以通过在一个读周期后紧跟一个写周期的方式一次刷新一行。（一行单元共用一个字线）

DRAM 增加了时钟，因此被称为同步DRAM，优势在于使用时钟对存储器和处理器保持了同步。

(3) 闪存

是可擦除的可编程只读存储器。EEPROM允许使用称为PROM编程器的硬件将数据写入设备中。在PROM被编程后，它就只能专用那些数据，并且不能被再编程这种记忆体用作永久存放程式之用。通常会用于电子游戏机、或电子词典这类可翻译语言的产品之上。

损耗均衡技术：将写操作从已经写入很多次的块中映射到写入次数比较少的块中，从而使得写操作尽量**分散**。

(4) 磁盘存储器

磁道：位于磁盘表面的数万个同心圆环中的任意一个圆环称为一个磁道。
扇区：构成磁盘上磁道的基本单位，是磁盘上数据读写的最小单位。

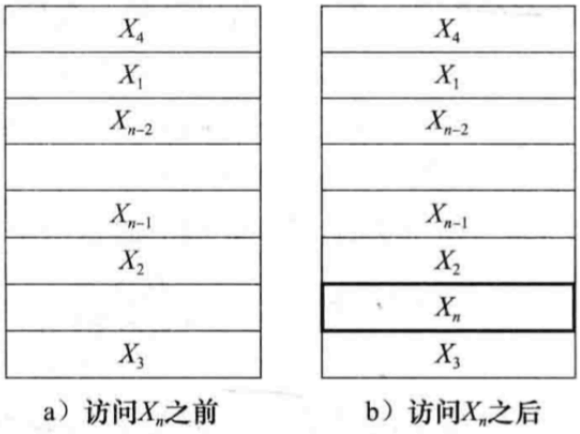
3. Cache技术

3.1 Cache基本原理

”Cache操作的最小单位是块!“

——做错了很多题的人

cache是一个存放等待使用的事物的安全场所，它把最近所访问过的数据项都存放起来，是处理器和主存之间的一个特殊层次。如下图所示，cache其实就是用来保存最近访问过的数据（让cpu方便查找）。当处理器索要访问的数据项X_N不在cache中的时候，*就导致了一次缺失，然后X_n被从主存里调入到cache。*



a）访问X_n之前　　b）访问X_n之后

那么如何找到数据项是否在cache中呢？这就好像是寻址，通过一定的映射结构来找到字的地址。常用的方法有**直接映射**，在直接映射中，每一个存储器的地址仅仅对应到cache中的一个位置。这种映射的计算方式如下：

$$（块地址）mod（cache中的块数）$$

如果 cache 中的块数是 2 的幂，取模的计算就很简单，只需要取地址的低 $\log_2$（块中的 cache 容量）位。因此，一个 8 块的 cache 可以使用块地址中最低的三位（$8 = 2^3$）。例如，图 5-8 中，直接映射的 cache 块大小为 8 个字，存储器地址 $1_{10}(00001_2)$ 到 $29_{10}(11101_2)$ 被映射到 cache 中 $1_{10}(001_2)$ 到 $5_{10}(101_2)$ 的位置。

上面这一段话截取自课本，老师说这是*2的幂的性质*，我目前还没想到如何证明，这里先强行记住吧（）

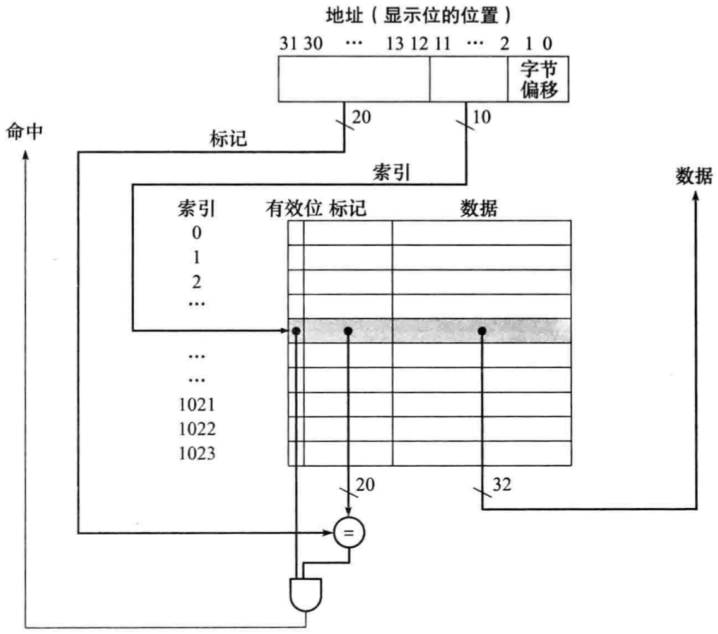
仅有地址是不够的，我们根据求模的性质可以知道，有可能两个地址对应求得的结果是相同的，那么如何来确定cache中的数据是我们所请求的呢？——**加入一组tag**。tag就是用来标记包含的地址信息，从而来判断是否为所求，它只需包含地址的**高位**，我们知道，在取模计算的时候，高位对结果是没有任何影响的，这也是为什么有可能不同的 地址定位到了相同的cache。

cache中是有可能没有包含有效信息的，比如，当一个处理器启动时，cache中没有数据，标记域的值也就没有意义了。因此我们使用一个有效位**valid**来表示是否为有效地址。

cache的访问具有时间局部性：总是替换掉最近较早访问的字。

3.1.2 地址模式

对于一个主存中的地址，它被划分为两个部分：标记域和cache索引，前者可以帮助与cache中的信息做比较，后者用来选择块，后者一般就是低$log_2(块)$位。我们接下来来分析一组主存地址：



如这张图所示，地址的低位用来选择由数据字和标记组成的一个cache块位置。cache中的标记就是与主存地址中的高位进行比较从而来判断是否符合请求的地址。因此，主存地址的[11:2]这十位用来判断块的地址（至于为什么是10呢？下面会详细介绍，这个cache的容量有关），[1:0]这两位是偏移量，那么剩下的（32-10-2）位就用来做高位比较了。当通过主存的地址寻找cache中数据时，如果标记相同、块地址正确且有效位为1，那么就是一次HIT，否则就是MISS。

注意cache是按照**字**访问的，因此:

- 32 位地址。
- 直接映射 cache。
- cache 大小为 2^n 个块，因此 n 位被用来索引。
- 块大小为 2^m 个字（2^{m+2} 字节），因此 m 位用来查找块中的字，两位是字节偏移信息。

标记域的大小为

$$32 - (n + m + 2)$$

直接映射的 cache 总位数为

$$2^n \times (\text{块大小} + \text{标记域大小} + \text{有效位域大小})$$

由于块大小为 2^m 个字（2^{m+5} 位），同时我们需要 1 位有效位，因此这样一个 cache 的位数是

$$2^n \times (2^m \times 32 + (32 - n - m - 2) + 1) = 2^n \times (2^m \times 32 + 31 - n - m)$$

cache块中的索引以及标记唯一确定了cache块中存放的内容在主存中的地址。

Cache不可避免地会产生一定的缺失（miss），对于这种情况，我们其实可以有很多改进的方法。比如，大一些的Cache块可以更好地利用空间局部性从而降低缺失率，但是这样的比较大的块查询的速度会相对慢一些，缺失的成本也会增加。由书上图5-11可以看出，当块特别大的时候缺失率反而会上升。这个后面会专门讨论如何取舍。

3. 2 cache缺失处理

cache缺失顾名思义就是由于数据不在cache中而导致被需要的数据不能被满足。针对这种情况，控制单元会从主存或者较低一级的cache中读取数据， 并且更新有效位到Y。所以Cache这样的更新方式使得它总是会替换掉较早访问的字。

当cache缺失的时候，他会冻结所有的寄存器中的内容，整个处理器阻塞，等到主存的读操作完成以后，才开始正常执行读指令的任务。

1）把程序计数器（PC）的原始值（当前 PC－4）送到存储器中。
2）通知主存执行一次读操作，并等待主存访问完成。
3）写 cache 项，将从主存取回的数据写入 cache 中存放数据的部分，并将地址的高位（从 ALU 中得到）写入标记域，设置有效位。
4）重启指令执行第一步，重新取指，这次该指令在 cache 中。

注意，访问内存的时候可能需要多个周期，只要有了多个周期，就要把pc不由自主加上的4都给减掉，不然读的就不是同一条指令了！

> Instruction cache miss: restart instruction fetch
> Data cache miss: complete data access

3.3 写操作处理

为了保持cache和主存中的内容一致（时刻一致，不会因为主存中的内容变化而cache中没变化而造成inconsistent）最简单的方法是将数据同时写入主存和cache中（write through）写直达的方法每次都要把数据写到内存里，耗费的时间太长。

> 对于写直达机制，有两种情况引起阻塞：
>
> 1. 写缺失，要求在继续执行写操作之前取回数据块
> 2. 写缓冲区阻塞

$$\text{写操作阻塞的时钟周期数} = [(\text{写的次数} / \text{程序数}) \times \text{写缺失率} \times \text{写缺失代价}] + \text{写缓冲区阻塞}$$

写缓冲（write buffer）：当写入主存的操作完成以后，再把缓存中的数据项写进到主存。如果缓存区内容满了，那么挂起，处理数据，等到有空位置了才能继续。

写回（write back）：当发生写操作的时候，新的值只被写入到cache中，然后当修改过的cache块被替换的时候才需要写到主存中。

4. 改进Cache的性能

4.1 改进方案评价

1. cpu速度加快：miss penalty更加大
2. 降低了基本的CPI：greater proportion of time spent on memory stalls
3. 加快时钟周期：Memory stalls account for more cpu cycles （阻塞/挂起的绝对时间是不变的）

> 如何评价cpu处理的速度很快，但是存储系统很慢?
>
> 因为存储器阻塞花费的时间会更多，然后根据Amdahl定律就可以知道，cpu速度快只能改变他那一部分的速度。

4.2 Associative Cache ~ 灵活放置块

之前讲的块放入cache的方法是比较直接的，因为一个块指定了cache中的一个确定的位置（直接映射）还有其他方法，比如全相联（fully associative），意味着块可以和cache中的任何一个位置进行关联。HOWEVER，这样的比较方法在硬件上的开销实在是大，因此它只适用于块数比较少的cache。

介于这两个极端之间，还有组相联（set associative） 这种方法中可以被放置的位置数量是固定的。我们有n路组相联，n通常为2，4，6，8。组相联和直接映射的储存块计算位置方式不同。

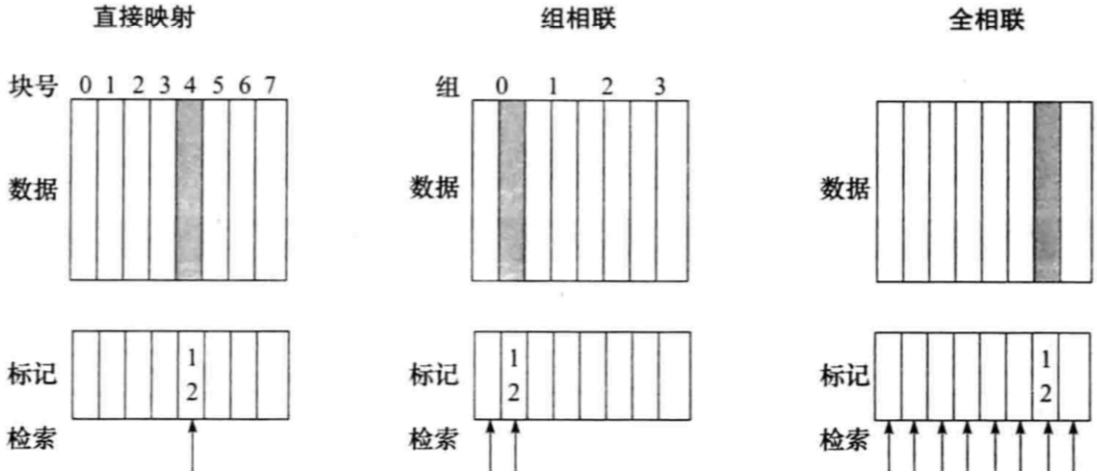
回想直接映射的 cache，一个存储块的位置是这样给出的：

（块号）mod（cache 中的块数）

而在组相联 cache 中，包含存储块的组是这样给出的：

（块号）mod（cache 中的组数）

注意在组相联中，块可以被放在组中的任何一个位置，因此组中的所有块的标记都要被检索。下面这张图就很生动地展示了各种映射方式的检索开销：



从课上讲的几个实例可以看出，随着组相联时组内数量增加，miss rate降低了。由此看来，组相联的确可以比较好地解决缺失的问题。但是新的问题出现了，我们可以看见，相联度从4到8，miss rate只下降了0.2个百分比。所以全相联不一定能够显著降低。块可以映射的位置越多，硬件的消耗和开销就越大，比如全相联就比较耗。

注意：当缺失的时候，我们需要一个统一的替换规则，对于组相联，一般是替换掉最近最少使用的块中内容。这叫LRU替换算法，但是当相连度提高时，执行就变得有些苦难了。

4.2.2 替换问题

最常用的方法时LRU法，即最近最少使用，被替换掉的事醉酒没有使用的那一块。

LRU 替换算法的实现是通过跟踪每一块的相对使用情况。对于一个两路组相联 cache，跟踪组中两个数据项的使用情况可以这样实现：在每组中单独保留一位，通过设置该位指出哪一项被访问过。当相联度提高时，LRU 的执行就变得困难些；在 5.8 节中，我们将会讨论另一种替换机制。

4.3 使用多级cache结构

参考老师上课的例题。全概率公式~使用。一般的，多级cache 中的一级cache相比起普通的单级cache一般很小，二级cache一般比他大。**一级cache更关注命中时间，二级cache更关注缺失率。**

如果题目里说的缺失概率是一个全局的概率，就直接乘以就可以了。这个题目真的有点坑。

5. 虚拟存储器

虚拟机是什么？：

系统虚拟机让用户觉得自己在使用包括操作系统的副本在内的整个计算机。一台运行多个虚拟机的计算机可以支持多个不同的操作系统。在一个传统的平台上，一个单独的操作系统拥有所有的硬件资源，但是通过使用虚拟机，多个操作系统共享硬件资源。

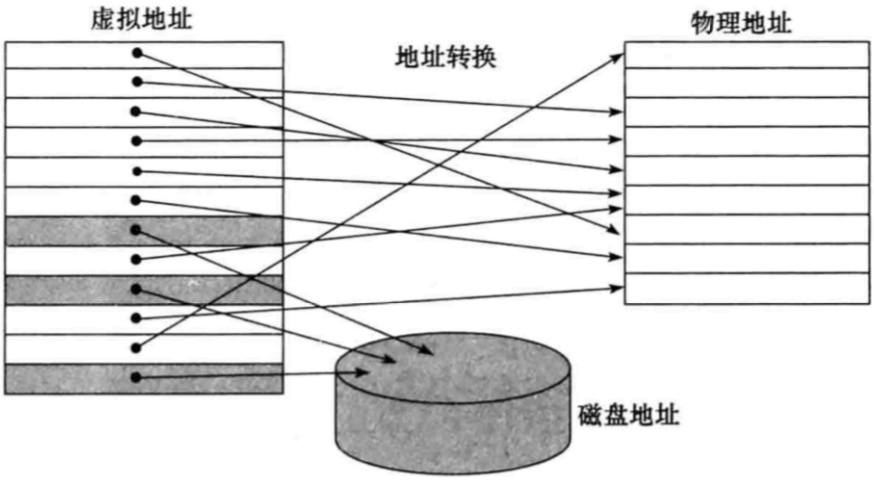
5.1 虚拟存储器 Virtual Memory

Virtual Memory: Use main memory as a cache for secondary (disk) storage. 这也是一种缓存技术。每一个程序都在虚拟存储器中有一段空间，放置它最近要用到的代码、数据。这样其他的程序就不会访问、修改这个程序的数据，分隔开了。

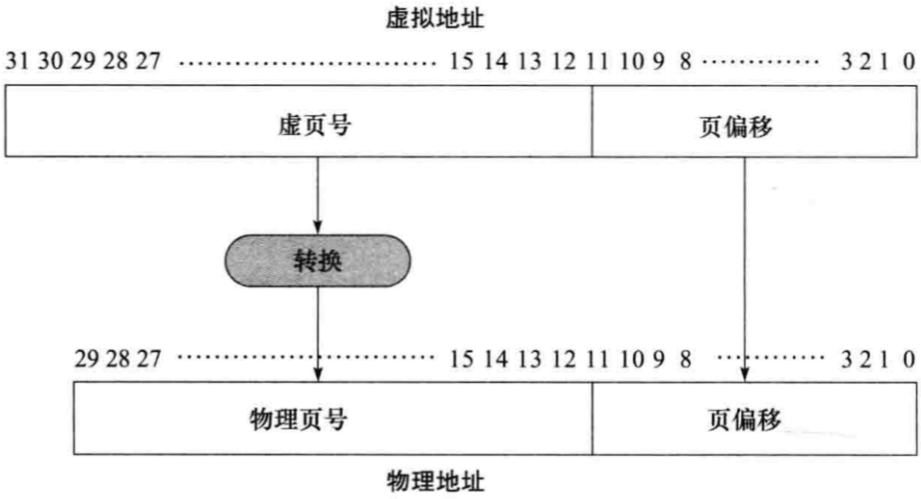
> 术语：在VM中，块被称为页(page)，访问缺失被称为缺页(page fault)。
>
> 虚拟地址：虚拟空间的地址，当需要访问主存的时候，需要通过地址映射转换成物理地址。

In a virtual memory, programs share main memory but each gets a private virtual address space to avoid comflicts. the space stores the code and data that the program recently uses. This strategy could protect from other programs' visit.

虚拟存储器可以管理由主存和辅助存储器组成的两级存储器层次结构。他的工作原理和cache几乎一样，但是术语和一些具体的细节又有所不同，后面要提到。



从虚拟地址，到物理地址，其实就是把虚页号转换成物理页号，然后页偏移直接对应下来就可以访问主存。如下图所示：这个对应比起cache就简单一些了。



5.2 Page table 和 Page Table Register

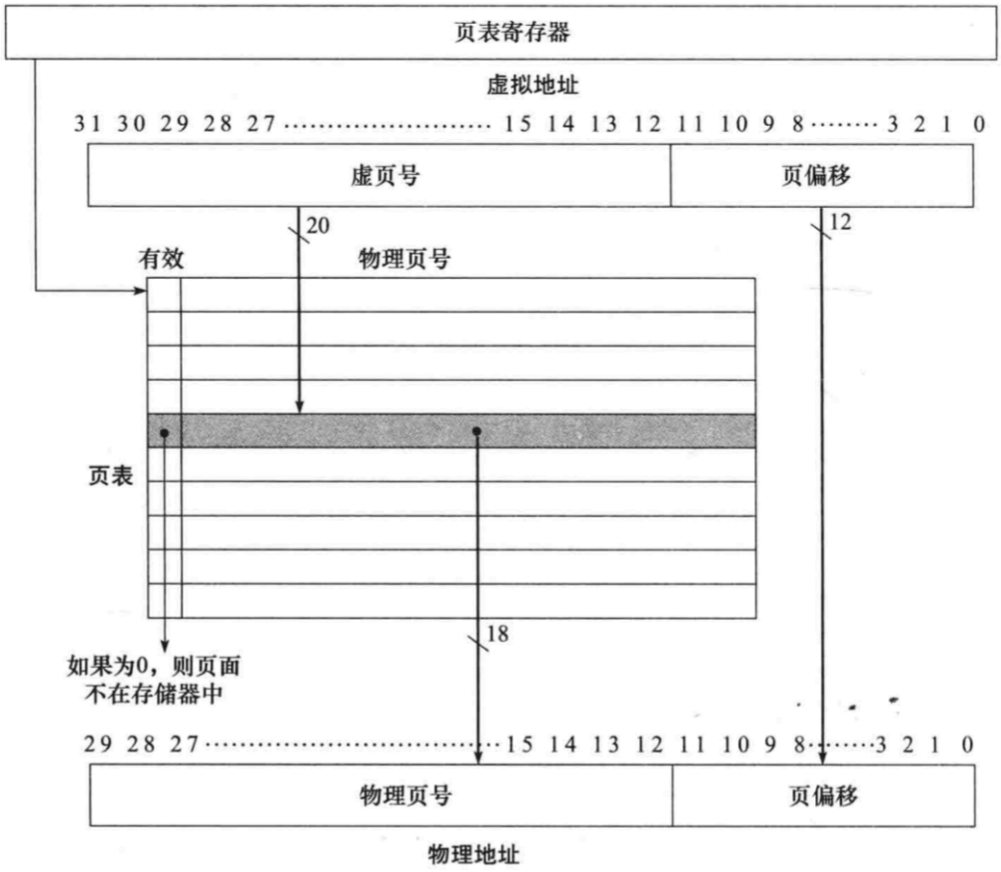
由于缺页的代价高得惊人，设计人员通过对页的放置进行优化从而降低缺页频率。如果允许一个虚拟页映射到任何一个物理页，那么当缺页发生时，操作系统可以选择任意一个页进行替换。例如，操作系统可以使用复杂的算法和复杂的数据结构来跟踪页的使用情况以选择在较长一段时间内不会被用到的页。使用更先进更灵活的替换策略降低了缺页率，也使全相联方式下页的放置变得更简单。

页表：页表被放在主存中，是用虚拟地址中的页号作为索引，是一个保存着虚拟地址和物理地址之间转换关系的表。

> page table registers in CPU points to page table in physical memory.

如果在主存中有内容：页表中的对应项设置有效位为1，并且直接关联一个物理地址存放到PT中；如果主存中没有内容，发生缺页，PTE会在低级的disk中查找。可以发现，页表中包含了每个可能的虚拟页的映射，因此是不需要标记位的！

页表寄存器：可以指出页表在存储器中的位置，是一个指向页表首页的寄存器。



5.3 缺页故障 Page Fault Penalty

当一个缺页故障发生的时候，首先，操作系统获得控制权，page必须要从disk磁盘中获取，这样的一个过程要耗费"millions of clock cycles",并且需要有操作系统实现。缺页故障对cpu的影响如果大了，之前所讲的cache技术的优势也就没有了，因此需要减小出现的比率。

swap space：操作系统在创建进程的时候在闪存或者磁盘上为进程中所有的页创建空间，也记录了每个虚拟页在磁盘上的存放位置。它为进程的全部虚拟地址空间预留了磁盘空间。

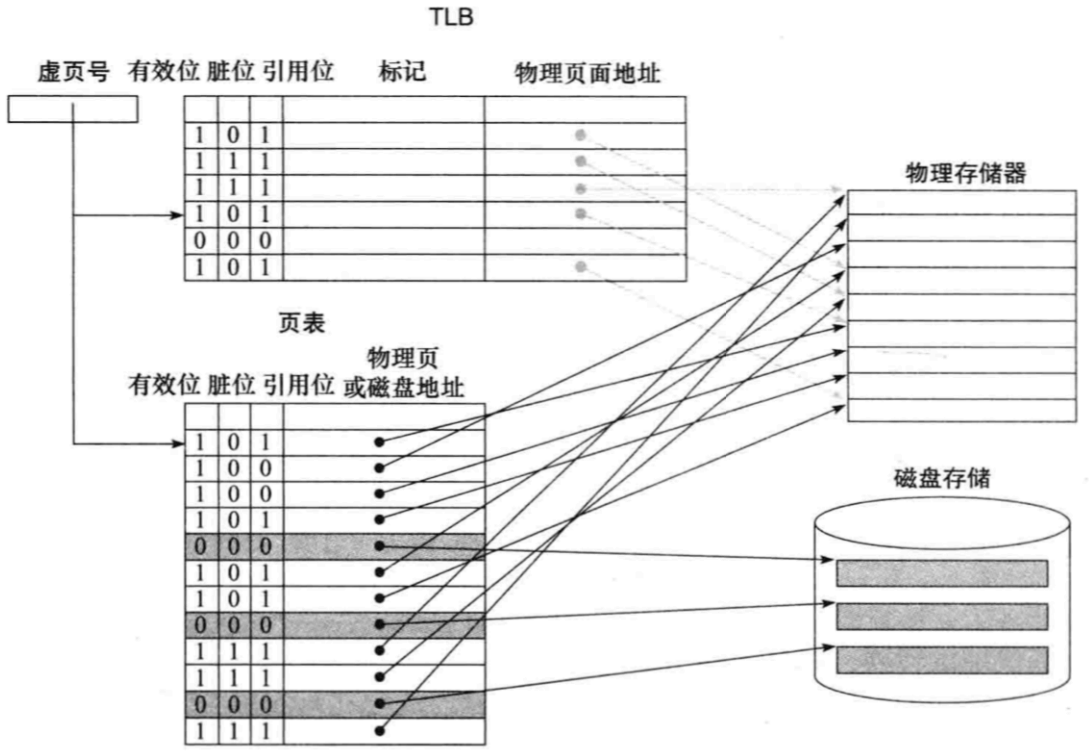
> dirty bit：脏位，当页中的任何字被写时，就将dirty置有效。脏位决定了是否需要把当前的page中内容写入到磁盘，这样的决定可以减少在写磁盘上的时间消耗。

5.4 TLB translation- Lookaside Buffer

对页表的访问其实开始还是有点大的，因为页表是被放在主存里，程序访问主存需要两次，一次获得物理地址，一次获得数据。TLB又被称作快表，它用来记录最近使用地址的映射信息的高速缓存，从而可以避免每次都要访问页表。（我感觉就是在PT和Disk之间又加了一级缓存cache）

如图 5-29 所示，TLB 的每个标记项存放虚页号的一部分，每个数据项中存放了物理页号。由于我们每次访问的是 TLB 而不是页表，TLB 需要包括其他状态位，如脏位和引用位。

每次访问，我们都要在 TLB 中查找虚页号。如果命中，物理页号就用来形成地址，相应的引用位被置位。如果处理器执行的是写操作，脏位同样要被置位。如果 TLB 发生缺失，我们必须判断是发生缺页还是仅仅是一次 TLB 缺失。如果该页在主存中，那么 TLB 缺失只是一次转换缺失。在这种情况下，处理器可以通过将页表中的变换装载到 TLB 中并且重新访问来进行缺失处理。如果该页不在主存中，TLB 缺失就是一次真的缺页。在这种情况下，处理器调用操作系统的异常处理。由于 TLB 中的项比主存中的页数少得多，发生 TLB 缺失会比缺页频繁得多。



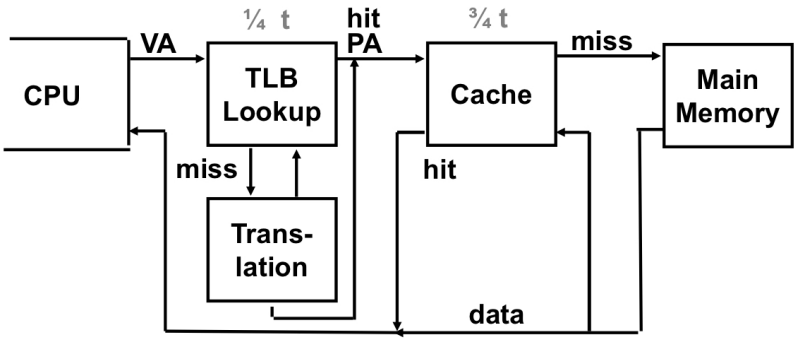
首先明确几个知识点：

- 缓存之于内存等价于快表之于页表
- 快表与页表存储的是虚拟地址和物理地址之间的转换（按页存储）
- 程序执行用的是虚拟地址（每个进程一个单独的编址空间存储在进程PCB中），但是访问内存或者访问缓存用的就是物理地址了。

PT、TLB工作流程：

1. 处理器发出一个读写访存请求，将该请求的虚拟地址解析成虚拟页号、页内偏移两部分。虚拟页号用来访问 TLB 以及内存中的页表。页内偏移用来访问渗透缓存。
2. 虚拟页号被解析成 TLB tag 比较地址和 TLB 索引两部分。通过 TLB 索引 定位到某一个或某几个（组相联）TLB 单元后，将 TLB tag 比较地址和定位到的 TLB 单元的 TLB tag 比较：若相同，则将该 TLB 单元中存储的 TLB 数据取出，该 TLB 数据即是此虚拟地址对应的缓存 tag 比较地址；否则，表明 TLB 缺失，使用 虚拟页号访问内存中的页表，从页表中找到该虚拟地址对应的缓存 tag 比较地址。
3. 将(1)中的页内偏移解析成缓存索引和块内偏移两部分。缓存索引被三级渗透缓存通用，处理器依次访问各级缓存。通过缓存索引定位到某一个或某几个（组相联）缓存单元后，将(2)中得到的缓存 tag 比较地址和定位到的缓存单元的 tag 比较：若相同，则表示缓存命中，通过块内偏移在缓存

单元中得到处理器需要的数 据；否则，表明该级渗透缓存缺失，继续访问下级缓存。若所有渗透缓存都缺失， 则将(2)中 tag 比较地址和(1)中的页内偏移组合成物理地址，使用物理地址访问内存，得到处理器需要的数据。



小测验答案

5.1 1 和 4。(3 是错误的，因为每个计算机的存储器层次结构的开销是不同的，但是在 2013 年开销最高的通常是 DRAM。)

5.3 1 和 4。更低的缺失代价可以允许使用更小的 cache 块，因为没有更多的延迟；而更高的存储带宽通常导致更大的块，因为缺失代价只是稍微大了一些。

5.4 1。

5.7 1－a，2－c，3－b，4－d。

5.8 2。(大容量的块和预取都能降低强制缺失，因此 1 是错误的。)

ref： https://blog.csdn.net/qq_41154905/article/details/105813019