存储

存储层级

存储等级

类别	大小	速度	描述
registers	小	快	
L1-cache(On-chip)			
L2-cache(SRAM)			以寄存器为单位的RAM
Main memory(DRAM)			以电容为单位的RAM
Disk,Tape...	大	慢	

磁盘访问

$$access\ time = seek(\text{寻道时间}) + \frac{rotation\ time(\text{旋转一圈时间})}{2} + data\ transfer + queuing\ delay$$

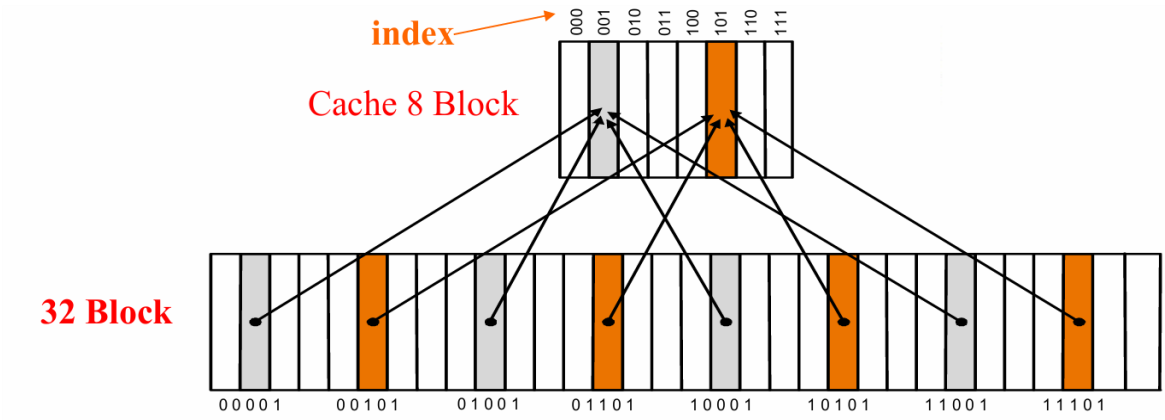
计算题

算 CPI，但是会给出指令正常 CPI，取址指令 L1-cache miss rate，L2-cache miss rate，分支预测错误率，因为指令内存和数据内存会造成结构冒险，算最后 CPI

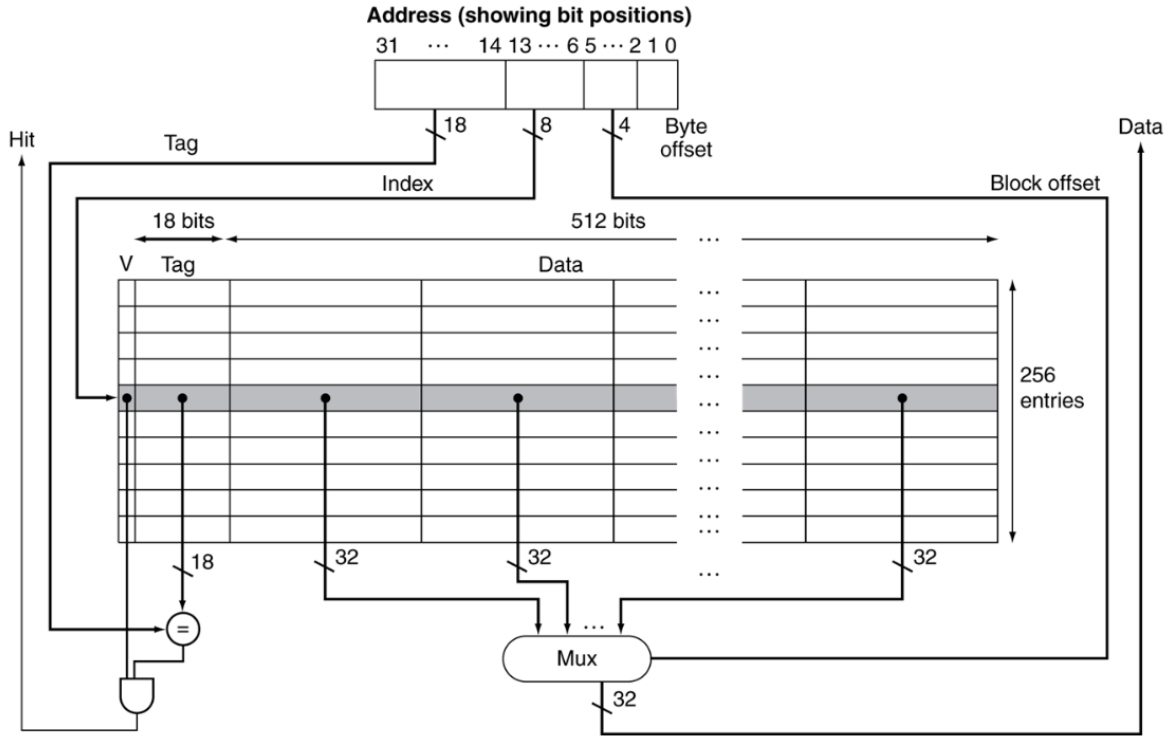
cache

direct mapped cache

将一定量内存块映射到一个 cache 块中



每个块需要记录 valid, dirty bit,Tag,data



计算

对于一个地址，我们将之分为：

Tag，index，offset

这个地址常常标注了每个 byte/word，当然考试的时候会具体给地址到底标注了 half word 还是 word 还是 bit，没有标注建议问监考老师。

其中，

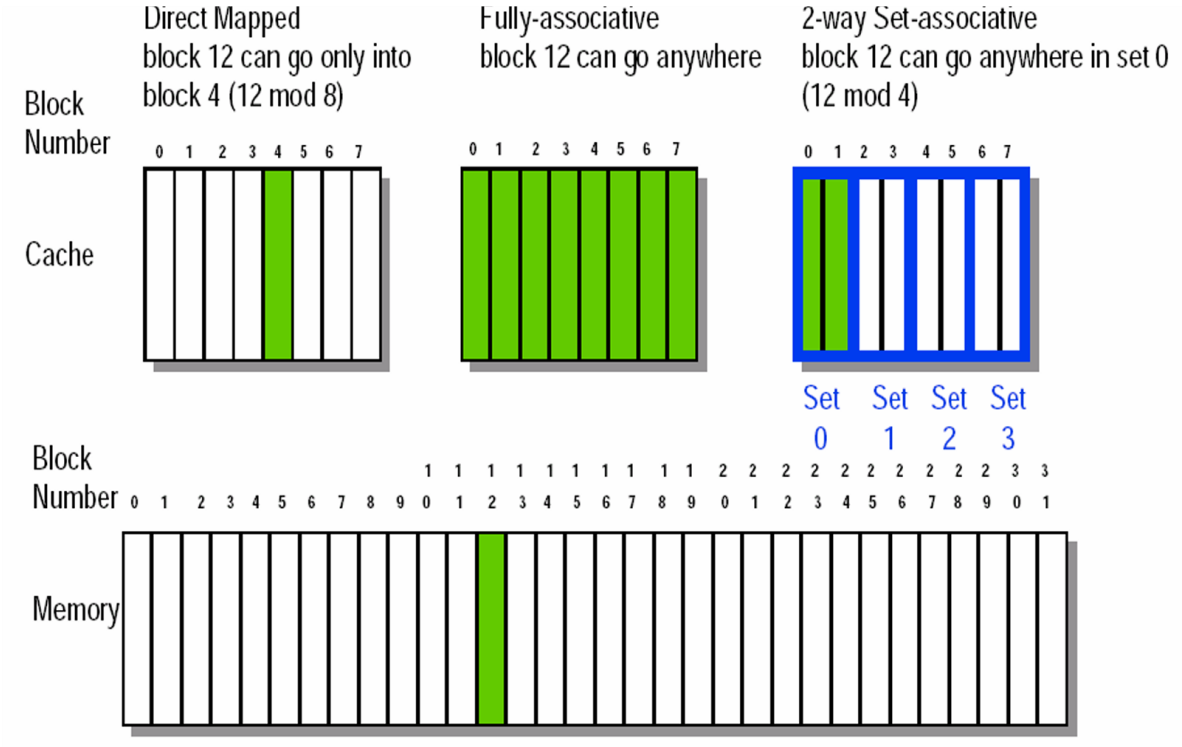
$$offset_{\text{位数}} = \log \frac{\text{块大小}}{\text{最小表示单元}}$$

$$index_{\text{位数}} = \log \frac{cache_{\text{大小}}}{\text{块大小} \times \text{组相联数}}$$

$$tag_{\text{位数}} = \text{地址位数} - index_{\text{位数}} - offset_{\text{位数}}$$

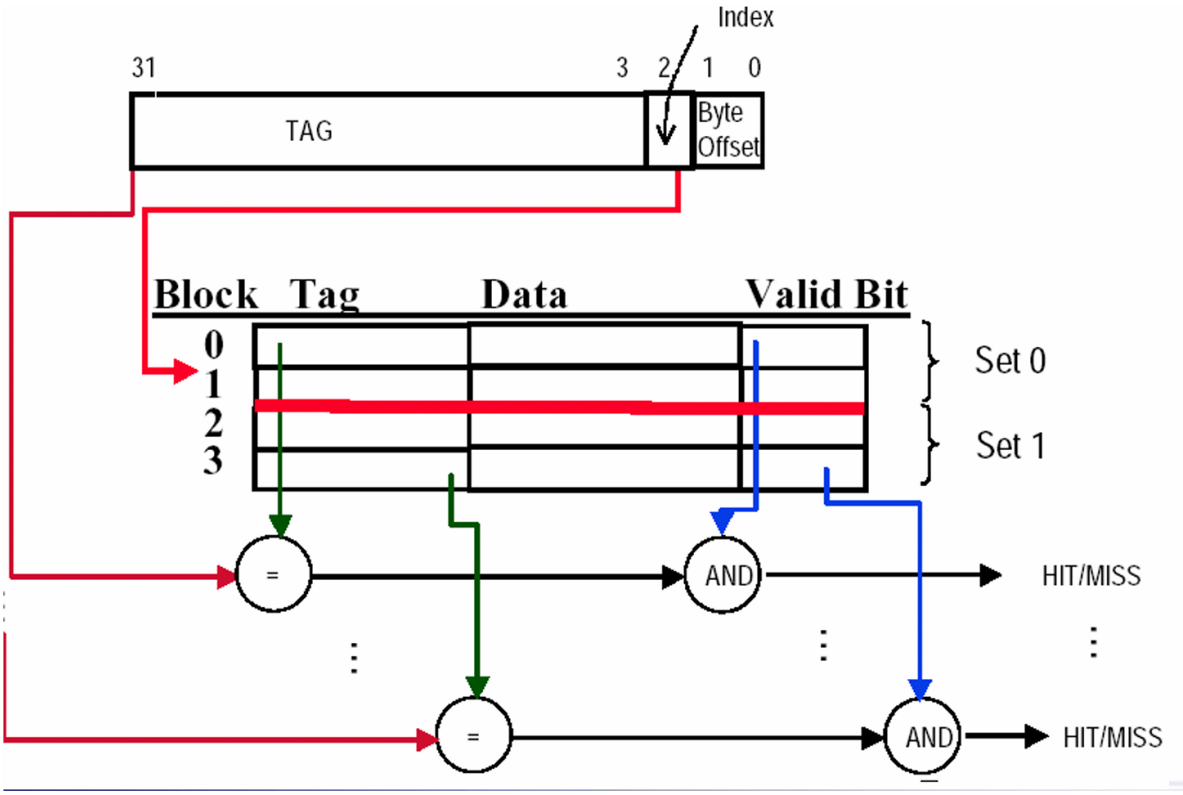
$$cache_{\text{总大小}} = cahce_{\text{数据大小}} + \text{块数} \times (tag_{\text{位数}} + (\text{具体参考题目有没有} v, d, LRU))$$

cache的优化

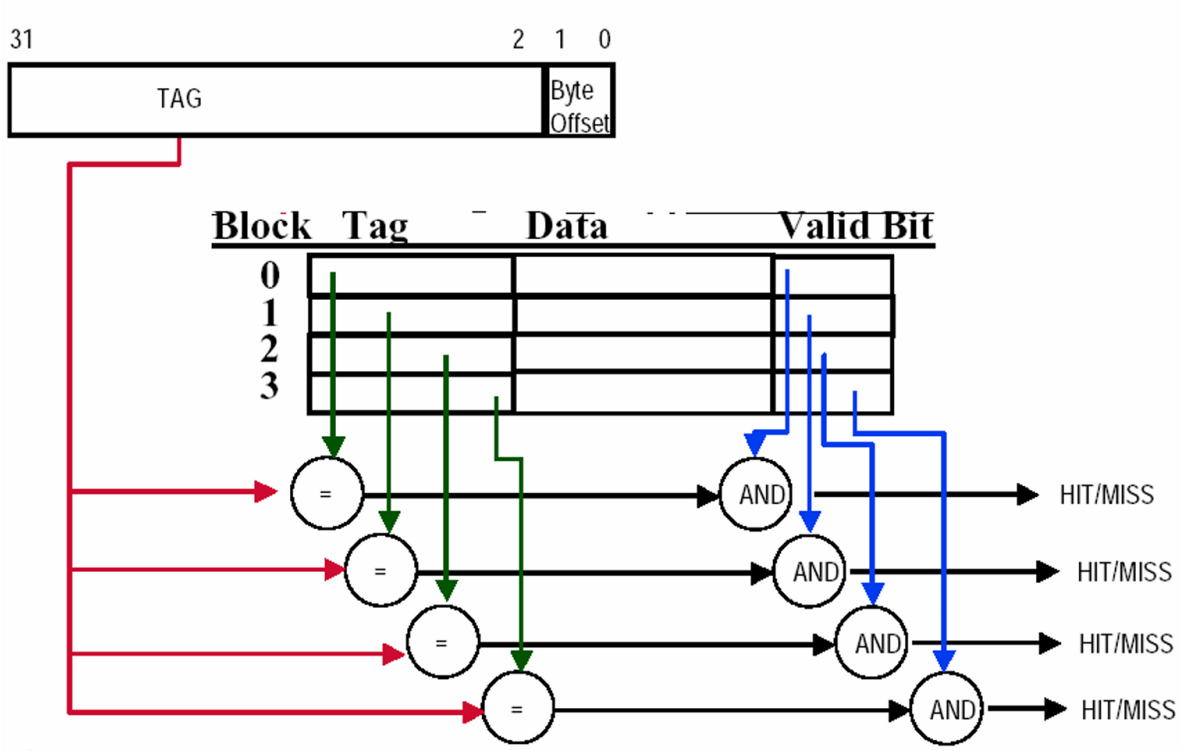


组相联

多路的cache命中和访问情况



全相联



替换

替换哪一块。通常有三种策略：

- **Random replacement**，随机挑选一个幸运 block 覆盖掉（需要一个随机数生成器）
- **LRU, Least Recently Used**，选择上一次使用时间距离现在最远的那个 block 覆盖掉（需要一些额外的位用来记录信息，具体实现没有讲）
- **FIFO, First In First Out**，选择进入时间最早的 block 覆盖掉（同样需要一些额外的位记录信息，同样没讲具体实现）

优化效果

> 这玩意其实带点背书，理解的话就是其他不变单优化一项

1. 对于 compulsory miss, 也成为 cold start miss，即第一次访存造成的延误，可以使用 increase block size 解决，但是这种方法增加了 miss penalty，也增加了因为 dirty 造成的 miss rate
2. 对于 capacity miss，即舍弃的块又被使用（A replaced block is later accessed again），我们认为原因是 finite cache size，使用 increase cache size 解决，但是会造成 increase access time 的问题
3. 对于 conflict miss，又称 aka collision miss 因为组数不够造成问题（competition for entries in a set），可以采用 increase associativity 解决，但是会造成 increase access time 的问题，我们认为 conflict miss would not occur in a fully associative cache of the same total size

cache 访问

直接问定义

###

cache的命中和访问过程情况,算命中率,看操作完后cache长啥样

过程

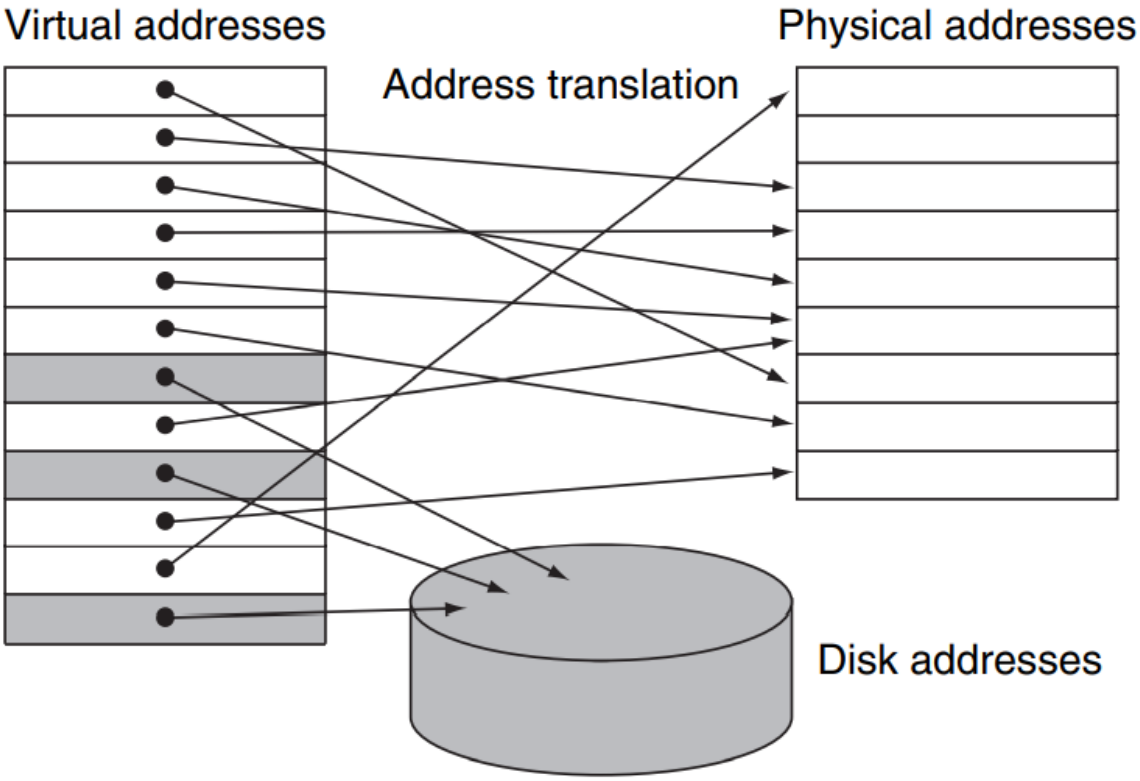
- Read
  - Hit
    - 直接从 cache 里读就好了
  - Miss
    - Data cache miss
      - 从 memory 里把对应的 block 拿到 cache，然后读取对应的内容。
    - Instruction cache miss
      - 暂停 CPU 运行，从 memory 里把对应的 block 拿到 cache，从第一个 step 开始重新运行当前这条指令。
- Write
  - Hit 有两种可以选的方式：
    - write-through

      即每次写数据时，既写在 cache，也写在 main memory。这样的好处是 cache 和 main memory 总是一致的，但是这样很慢。
      - 一个改进是引入一个 **write buffer**，即当需要写 main memory 的时候不是立即去写，而是加入到这个队列中，找机会写进去；此时 CPU 就可以继续运行了。当然，当 write buffer 满了的时候，也需要暂停处理器来做写入 main memory 的工作，直到 buffer 中有空闲的 entry。因此，如果 main memory 的写入速率低于 CPU 产生写操作的速率，多大的缓冲都无济于事。
    - **write-back**，只将修改的内容写在 cache 里，等到这个 block 要被覆盖掉的时候将其写回内存。这种情况需要一个额外的 **dirty bit** 来记录这个 cache block 是否被更改过，从而直到被覆盖前是否需要被写回内存。由于对同一个 block 通常会有多次写入，因此这种方式消耗的总带宽是更小的。
  - Miss 同样有两种方式：
    - **write allocate**，即像 read miss 一样先把 block 拿到 cache 里再写入。
    - **write around** (or **no write allocate**)，考虑到既然本来就要去一次 main memory，不如直接在里面写了，就不再拿到 cache 里了。
    - write-back 只能使用 write allocate；一般来说，write-through 使用 write around，其原因是明显的。

虚拟地址

虚拟地址

实际上的 main memory（我们称之为 **物理内存, physical memory**）中的地址称为 **物理地址, physical addresses**；而我们给每一个程序内部使用到的内存另外编一套地址，称为 **虚拟地址, virtual addresses**；虚拟内存技术负责了这两个地址之间的转换 (**address translation**，我们稍后再讨论转换的方式)：



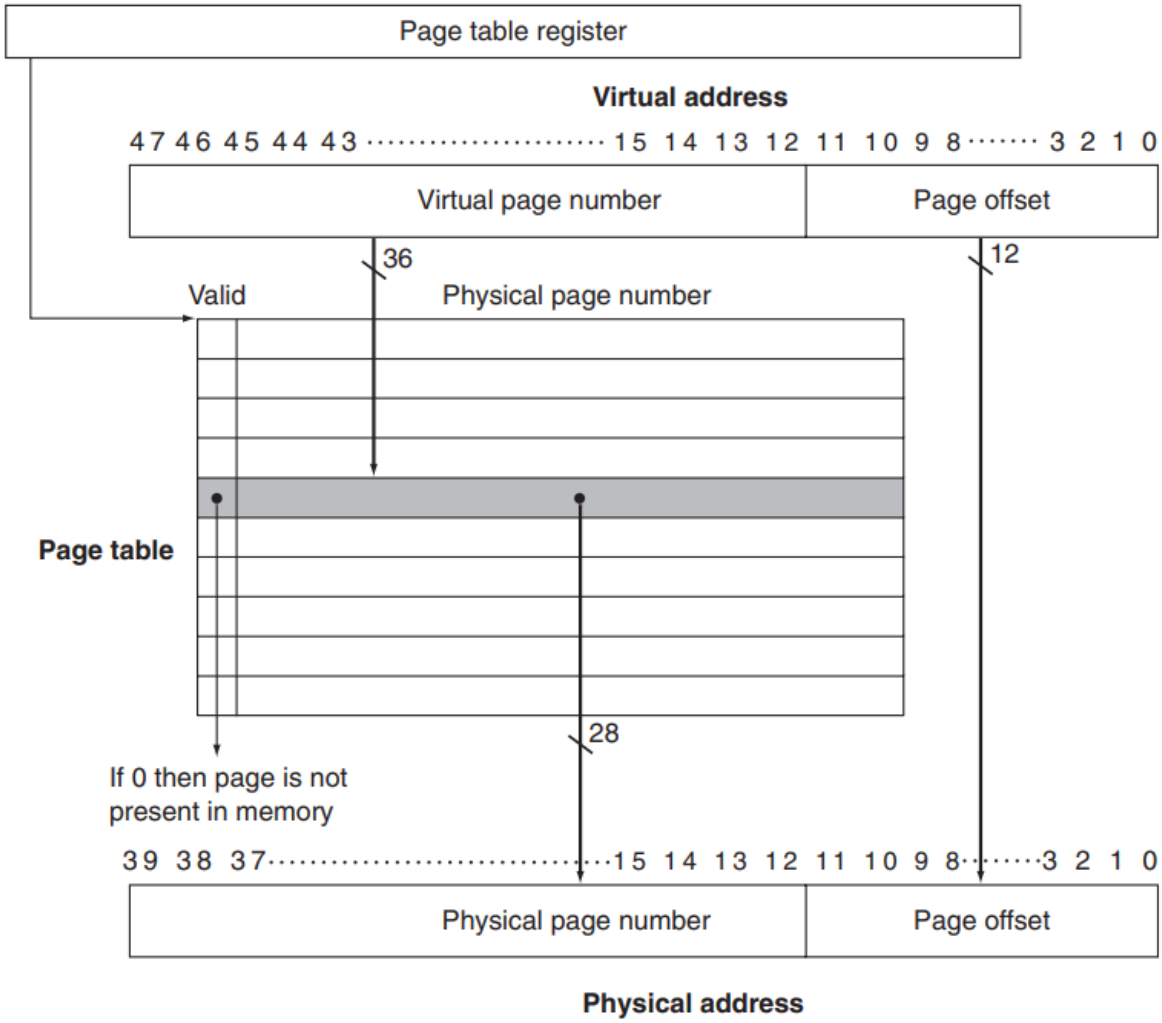
虚拟存储的技术和 cache 的原理是一样的，但是一些术语的名字并不相同。对应于 cache 中的 block / line，虚拟存储的内存单元称为 **page**，当我们要访问的 page 不在主存中而是在磁盘里，也就是 miss，我们称之为一次 **page fault**。

physical memory 的存放并没有分组的概念，即用 cache 的术语来说，main memory 是 fully-associative 的。page fault 的开销是非常大的，因此比较低的 page fault 的概率相对于额外的查询来说是非常划算的。同样，由于读写磁盘是非常慢的，write through 的策略并不合适，因此在 virtual memory 的技术中，我们采取 write back 的方式。

页表

page table 这种结构，它被存放在 main memory 中，每个程序（实际上是进程，但是写课本的人好像现在并不想引入这个概念）都有一个自己的 page table；同时硬件上有一个 **page table register** 保存当前进程这个页表的地址。

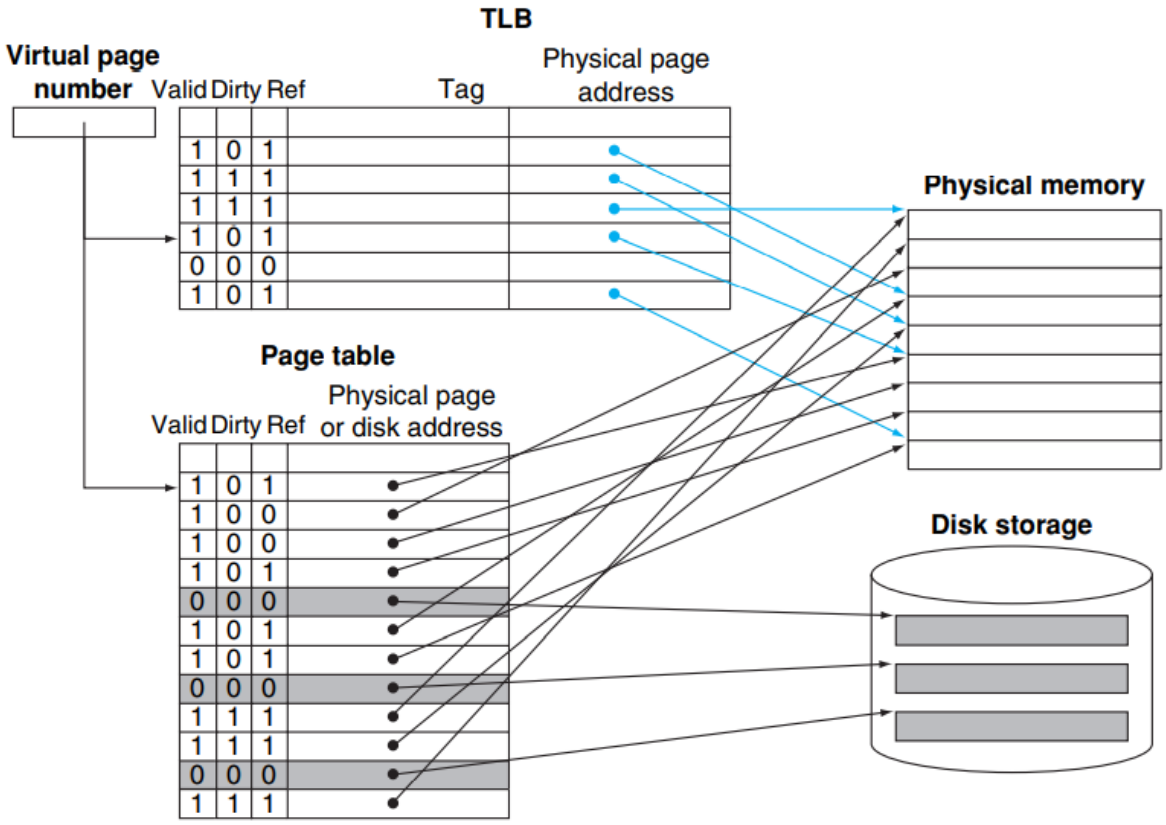
每个 entry 中包含了一个 valid bit 和 physical page number。如果 valid bit = 1，那么转换完成；否则触发了 page fault，handle 之后再进行转换。



TLB

结构

一个专用的高速查找硬件 cache，这里称它为 **translation look-aside buffer (TLB)**。它实际上就是 page table 的专用 cache（它真的是 cache；page table 并不是 cache，只是像 cache），其 associativity 的设计可以根据实际情况决定。



当 TLB miss 的时候，处理器去 page table 查找对应的项；如果发现对应项是 valid 的，那么就把他拿到 TLB 里（此时被替换掉的 TLB entry 的 dirty bit 如果是 1，也要写会 page table）；否则就会触发一个 page fault，然后在做上述的事。

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

FIGURE 5.33 The possible combinations of events in the TLB, virtual memory system, and cache. Three of these combinations are impossible, and one is possible (TLB hit, page table hit, cache miss) but never detected.

大小计算

TLB的大小以及TLB对应内存的大小

$$表项数量 = \frac{2^{虚拟地址位数}}{页表大小}$$

$$page\ table_{大小} = 表项数量 \times 表项大小$$

多核情况

即上课讲的 FSM