**作业（高速缓存、虚拟内存）**

陈文迪 519021910071

1. **直接映射：**

为了将内存数据块放置到高速缓存中，可以将内存地址分几部分看待：



Offset bits是块内偏移， Index bits用于标记该内存数据块在高速缓存中的组号（set number）, 剩下的tag bits是该数据块的标记。

如果高速缓存采用直接映射（direct mapped）采用写直达（write through）的更新策略，那么高速缓存中每一行包含的内容为：cache data block, tag, valid bit， 不需要包含dirty bit.

**根据已知信息，请填写下表中的空格部分：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Address size(bits)内存地址的长度，寻址到byte | Cache size | Block Size  数据块大小 | tag bits  tag 位数 | Index bits  Index 位数 | Offset bits  块内偏移位数 | Bits per row  每一行的总位数 |
| 16 | 4KB | 4B | 4 | 10 | 2 | 32+4+1 |
| 32 | 32KB | 16B | 17 | 11 | 4 | 128+17+1=146 |
| 32 | 64KB | 16B | 16 | 12 | 4 | 128+16+1=145 |
| 64 | 2048KB | 128B | 43 | 14 | 7 | 1068 |

**2、组相联映射：**

假设某计算机的主存地址空间大小为64MB，采用字节编址方式。其cache数据区容量为4KB，采用4路组相联映射方式、LRU替换和回写（write back）策略，块大小为64B。请问：

**（1）主存地址字段如何划分？要求说明每个字段的含义、位数和在主存地址中的位置。**

由于cache数据区容量为4KB，因此共有4KB/64B=64行，而每组有4行，因此共有16组。这表明主存地址需要用4位来确定组号，用6位来确定块内地址。而主存地址空间大小为64MB=B。故主存地址字段可以这样划分：高16位作为tag，中间4位作为组号，低6位作为块内地址。

**（2）该cache的总容量（不仅包括数据区容量）有多少位？**

4路组相联的每一行需要增加两个状态位来记录组内的访问顺序。因此，对于每一行，需要16个tag标记位，1位有效位，1位修改为和2位LRU位，因此cache的总容量有位。

**3、代码分析与高速缓存的性能：**

一个二路组相联映射的高速缓存（ 2-way associative cache ）容量为128 bytes ，每个高速缓存块大小为32 字节（32 bytes per block）. long long型数据的长度为 8个字节（8 bytes）.假定table数组的起始地址为 0x0. **以下代码的高速缓存失效率(miss rate)为多少？**

**int i;**

**int j;**

**long long table[4][8];**

**for (j = 0; j < 8; j++)**

**for (i = 0; i < 4; i++)**

**{ table[i][j] = i + j; }**

**问题解答：**

一个缓存块能存储连续的4个long long 型数据。该二路组相联cache共有2组，每组有两行。第0组在主存中的起始地址为000000000，001000000，010000000和011000000，恰好对应table[0][0]，table[1][0]，table[2][0]和table[3][0]的地址；第1组在主存中的起始地址为000100000，001100000，010100000和011100000，恰好对应table[0][4]，table[1][4]，table[2][4]和table[3][4]。因此，当采用上述代码访问数组时，每次访问的下一个数据要么位于同一组但还未放入缓存的块中，要么位于另一组但也不在缓存的块中，因此miss rate将达到100%。

**4、平均存储器访问时间（Average Memory Access Time：AMAT）**

AMAT 是内存访问的平均（expected）时间 , 可以用以下公式来估算：

AMAT = hit\_time + miss\_rate × miss\_penalty

* Hit\_time: cache hit 时，访问cache 所花的时间
* Miss\_rate: 高速缓存的失效率
* miss penalty：当发生cache miss时，需要花的额外的访存时间 ，所以一次 cache miss 需要花费 (hit\_time + miss\_penalty) 的时间

**假设高速缓存系统的属性如下，求AMAT是多少？**

a) L1$ hits in 1 cycle (local miss rate 25%)

b) L2$ hits in 10 cycles (local miss rate 40%)

c) L3$ hits in 50 cycles (global miss rate 6%)

d) Main memory hits in 100 cycles (always hits)

Global miss rate和 Local miss rate 的定义请参考如下描述：

Global miss rate：

– the fraction of references that miss some level of a multilevel cache

– misses in this cache divided by the total number of memory accesses generated by the CPU

Local miss rate – the fraction of references to one level of a cache that miss

Local Miss rate L2$ = L2$ Misses / L1$ Misses

= L2$ Misses / total\_L2\_accesses • L2$ local miss rate

**问题解答：**

AMAT=1 cycle + 25% × 10 cycles + 25%×40%×50 cycles + 6% × 100 cycles = 14.5 cycles.

**5、虚拟存储器（Virtual Memory**）

程序中使用的内存地址是虚拟地址，一个虚拟地址（VA）可以看作两部分：虚页号、页内偏移（page offset），如下图中标示：

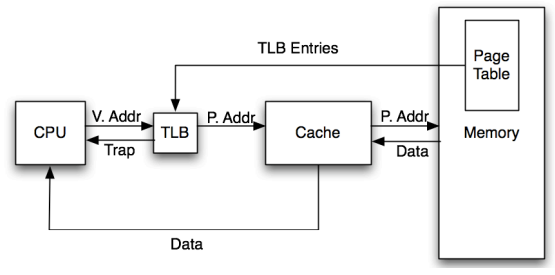


但事实上无论是数据还是指令都是存放在物理内存上的。一个物理地址（PA）也可以看作两部分：物理页号、页内偏移。如下图中标示：



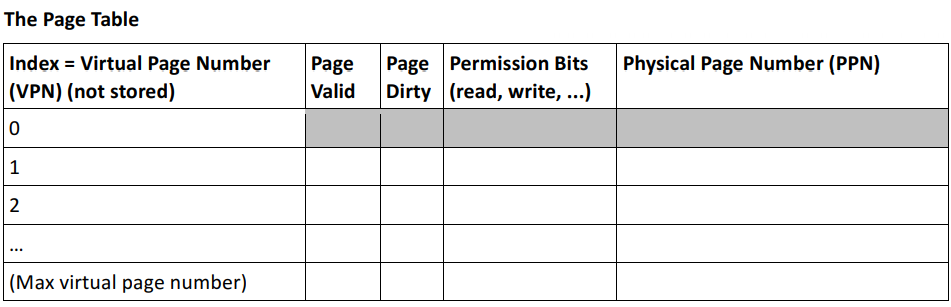
如果页大小是4KB=4096 bytes，那么page offset 就是12位。

将 VA 转换为 PA 会使用到快表（TLB） 和页表（Page Table）. 下图示意了TLB和页表（ｐ在内存访问时所处的位置。



每一个进程都有一个页表，页表存储在内存中，操作系统通过设置一个专用寄存器的值，告诉硬件页表在内存中的起始地址。每当切换进程时，操作系统会将要执行的进程的页表起始地址转载到这个专用寄存器中。

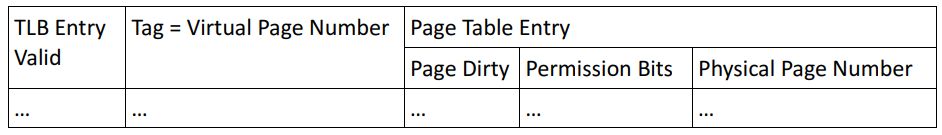
页表的结构一般如下：



每一个页表项（ page table entry）除了记录虚页号（VPN）和物理页号（PPN）的映射关系之外，还设置了有效位、脏位和权限位：

* “page valid” 有效位：用于标记该虚页是否在内存中；
* “page dirty” 脏位：操作系统需要知道，是否将该内存页更新到磁盘上；
* “permission bits”权限位：用于限制对该页进行某种操作。

快表（TLB）是页表的缓存（cache）, 假设TLB如果采用全相联映射（fully associative）机制，它的结构如下：



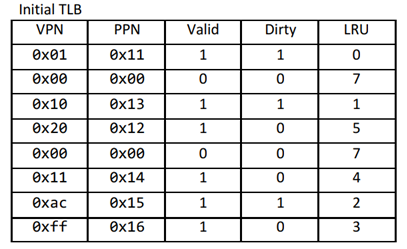
回答问题：

1. **如果页表地址寄存器中装入了新的值，TLB会发生什么操作？**

TLB应该将全部条目的valid位设置为0，表示失效。

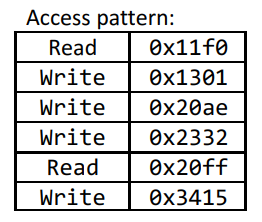
1. 某个处理器内存地址长度为16位，页大小为256 byte，TLB采用全相联映射，总共有8个TLB 表项， 并采用LRU替换机制（LRU位有3位，可以表示8个TLB表项的访问次序。如果LRU位的值为0，表示该页最近刚刚被访问）。

假设当前进程初始时TLB的内容如下，并假设该进程访问的所有页既可以读也可以写。



假设现在空闲的物理页是：0x17, 0x18, 0x19；

如果接下来，用标记出的访问模式（读或者写）对以下内存地址进行访问：



**请画出完成以上访问后，TLB的final state。**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| VPN | PPN | Valid | Dirty | LRU |
| 0x01 | 0x11 | 1 | 1 | 5 |
| 0x13 | 0x17 | 1 | 1 | 3 |
| 0x10 | 0x13 | 1 | 1 | 6 |
| 0x20 | 0x12 | 1 | 1 | 1 |
| 0x23 | 0x18 | 1 | 1 | 2 |
| 0x11 | 0x14 | 1 | 0 | 4 |
| 0xac | 0x15 | 1 | 1 | 7 |
| 0x34 | 0x19 | 1 | 1 | 0 |

