作业（高速缓存、虚拟内存）

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：

Tag bits	Index bits	Offset bits

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	146
32	64KB	16B	16	12	4	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。**

块大小为 64B，说明块内偏移量为 6 位。

Cache 的大小为 4KB，而块大小为 64B，故 Cache 中有 4KB/64B=64 个块。由于采取 4 路组相联映射方式，可知每组中有 16 个块，也就是说组号需要 4 位。

主存地址空间大小为 64MB，也就是 26 位。低 6 位是块内偏移量，中间 4 位是组号，则高 16 位为 Tag 位。

（2） **该 cache 的总容量（不仅包括数据区容量）有多少位？**

Cache 的每一行包括 16 位 Tag 位，1 位有效/无效位，2 位 LRU 位，1 位 dirty 位和共 64B 的数据位，而一共有 64 行，因此一共有 64×(16+1+2+1+64×8)=34048 位=4256B。

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; }
```

块大小为 32 字节，说明主存地址的低 5 位是块内偏移位；cache 大小为 128 字节，说明 cache 内一共包含了 4 个块，而使用二路组相联映射，说明一组里有两个块，组号有一位。

由于 table 数组为 long long 型，一个数据的长度为 8 个字节，故 4 个数据正好填满低 5 位，8 个数据正好填满低 6 位。也就是说，table[i][j]和 table[i+1][j]的低六位是完全相同的。因此，访问 table[0][0]时，第一次 cache miss 并将 table[0][0-7]放入 cache 的第 0 组第 0 块；访问 table[1][0]时，第二次 cache miss，由于组号完全相同，因此将 table[0][0-7]放入 cache 的第 0 组第 1 块；而访问 table[2][0]时，第三次 cache miss，由于组号完全相同，且该组所有的块都已经满了，所以将 table[2][0-7]放入 cache 的第 0 组第 0 块而替换掉 table[0][0-7]；访问 table[3][0]时，第三次 cache miss，由于组号完全相同，且该组所有的块都已经满了，所以将 table[3][0-7]放入 cache 的第 0 组第 0 块而替换掉 table[1][0-7]。

之后访问 table[0][1]，之前虽然 table[0][0-7]被放入过 cache，但是它被 table[2][0-7]替换过了，因此仍然是 cache miss，再一次用 table[0][0-7]放入 cache 的第 0 组第 0 块而替换掉 table[2][0-7]；依次进行下去，始终发生 cache miss。直到访问到 table[0][4]，组号变为 1，但是其实还是一样的情况，由于之后访问的数据组号也都是 1，所以之后仍然不可能命中。

因此 cache miss rate 是 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

即 75%的概率 L1$ hits；剩下 25% L1$ misses 时，有 40%的概率 L2$ misses，也就是说 25%×40%=10%的概率 L2$ hits；L3$的 global miss rate 6%，说明在剩下的 15%中，9%的概率 L3$ hits，6%的概率 main memory hits。

所以 AMAT=75%×1 cycle + 10 cycles×10% + 50 cycles×9% + 100 cycles×6% = 12.25 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：

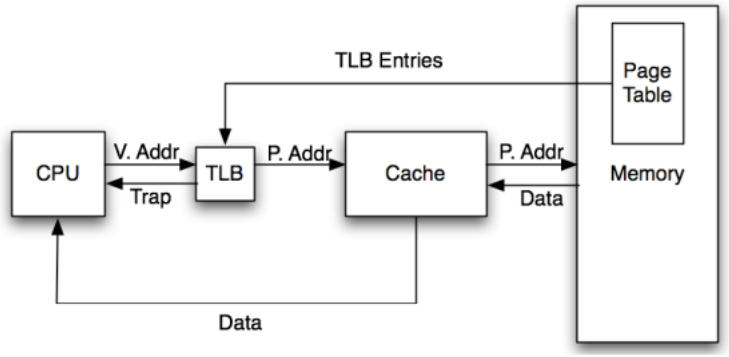
Virtual Page Number	Page Offset

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：

Physical Page Number	Page Offset

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：

The Page Table

Index = Virtual Page Number (VPN) (not stored)	Page Valid	Page Dirty	Permission Bits (read, write, ...)	Physical Page Number (PPN)
0				
1				
2				
...				
(Max virtual page number)				

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：

TLB Entry Valid	Tag = Virtual Page Number	Page Table Entry		
		Page Dirty	Permission Bits	Physical Page Number
...	...	...	...	...

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？

将 TLB 中的所有值全部置零

2） 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。

Initial TLB

VPN	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	0
0x00	0x00	0	0	7
0x10	0x13	1	1	1
0x20	0x12	1	0	5
0x00	0x00	0	0	7
0x11	0x14	1	0	4
0xac	0x15	1	1	2
0xff	0x16	1	0	3

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：

Access pattern:

Read	0x11f0
Write	0x1301
Write	0x20ae
Write	0x2332
Read	0x20ff
Write	0x3415

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