4.6 若使用高位作为组索引，则会有更连续的一段较长的空间被分在同一组索引中，即增加了竞争的可能，并导致缓存命中率降低；同时使用中间位作索引，索引位数可灵活增减以适应多路缓存不同的组数量，若置高位为索引则索引位数不能过大

4.7 这样处理页地址直接与标签对应，能够更快进行地址命中判断，省去地址分割与重新比较的逻辑；标签与物理地址的映射关系以及虚拟页地址与物理地址的映射关系可以统一起来进行管理，节省了页表占用的空间

4.8 (1) 访问延时 $t_i = 97\% \times T + 3\% \times 110T = 4.27T$

(2) 命中率为 $\frac{64kB}{1GB} = 6.1035 \times 10^{-5}$，访问延时为 $\frac{64kB}{1GB} \times T + \frac{1GB-64kB}{1GB} \times 110T$

$= 109.99T$

(3) 空间局部性体现在 Cache 能容纳数据大小的局部性，时间局部性体现在 Cache 保存的一段数据为最近经常使用的数据，刷新缓存需要带来比直接访问主存更大的代价。由于本程序访问数据完全随机，因此 Cache 的局限性反而降低了访存速度

(4) 设平均命中率为 m，则 $m \times T + (1-m) \times 110T < 105T \Rightarrow m > \frac{5}{109} = 4.59\%$

4.9

地址位数 Bit	缓存大小 KB	块大小 Byte	相联度	组数量	组索引位数 Bit	标记位数 Bit	偏移位数 Bit
32	4	64	2	32	5	21	6
32	4	64	8	8	3	23	6
32	4	64	全相联	1	0	26	6
32	16	64	1	256	8	18	6
32	16	128	2	128	7	18	7
32	64	64	4	256	8	18	6
32	64	64	16	64	6	20	6
32	64	128	16	32	5	20	7

4.10 (1) $0.22p_1 + 100.22(1-p_1) < 0.52p_2 + 100.52(1-p_2)$，即 $p_1 > p_2 + 0.3\%$

此时系统A平均访问时间低于系统B

(2) $0.22p_1 + 0.22k(1-p_1) < 0.52p_2 + 0.52k(1-p_2)$，即 $0.22p_1 > \frac{0.3k}{1-k} + 0.52p_2$

4.11 (1) 直接映射，索引位数为4，索引依次为 0001，0101，0001，0101，0101，0101，0101，发生5次替换

(2) 2路组相联，索引位数为3，索引依次为 001，101，001，101，101，101，101，发生3次替换

(3) 4路组相联，索引位数为2，索引均为01，发生3次替换

(4) 8路组相联，共访问7次内存，故不发生替换

4.12 程序依次往 0~383 地址写数据，缓存共16块，块内偏移4字节，

除去块内偏移后的地址为 0B ~ 1011 1B

(1) 直接映射，索引共 4 位，首次循环缺失率 25%，此后循环缺失率 16.7%

运行程序的缺失率为 $\frac{1}{N}25\% + \frac{N-1}{N}16.7\% = 16.75\%$

(2) 两路组相联，索引共 3 位，首次循环与此后循环缺失率均为 25%，因为除首次循环的 array[0] ~ [15] 位访问为强制不命中外，其它不命中均为冲突不命中，

运行程序的缺失率为 25%

4.13
```
for (int j=0; j<128; ++j) {
    for (int i=0; i<64; ++i) {
        A[j][i] = A[j][i] + 1;
    }
}
```

4.14 (1) 块内偏移 5 位，块总数 128 个，优化前地址访问顺序为每隔 64 个数字进行访问，索引共 7 位，

a[0][0] ~ a[15][0] 为强制不命中，a[16][0] ~ a[127][0] 均为冲突不命中，

a[0][1] 也为冲突不命中（与 a[112][0] 冲突），依此下去，均不命中，

缓存缺失次数为 128 × 64 = 8192

优化后为逐地址访存，由于每块可存储 8 个 int 数，则缺失次数为 $\frac{128\times64}{8} = 1024$

(2) 全相联缓存无索引位，优化前地址访问 a[0][0] ~ a[127][0] 恰好装满缓存的 128 块，访问 a[0][1] 时命中，并一直命中至 a[127][7]，此后 a[0][8] ~ a[127][8] 再次更新缓存，缺失次数为 $\frac{128\times64}{8} = 1024$

优化后的代码从 a[0][0] ~ a[0][7] 访问第 1 块，a[0][8] ~ a[0][15] 访问第 2 块，直到 a[15][56] ~ a[15][63] 访问最后一块，缺失率始终为 $\frac{1}{8}$，缺失数为 1024

(2) 优化前需满足 a[0][0] ~ a[127][0] 均强制不命中，需要 128×8 = 1024 个缓存块，缓存容量为 1024×32B = 32KB，

优化后单缓存块也能满足需要，则最低缓存容量为 32B

4.15 input 数据地址范围为 0x00 ~ 0x3B，output 数据地址范围为 0x40 ~ 0x7B

缓存共 2 块 16B

	input 数组				output 数组			
	列0	列1	列2	列3	列0	列1	列2	列3
行0	miss	hit	hit	hit	miss	miss	miss	miss
行1	miss	hit	hit	hit	miss	miss	miss	miss
行2	miss	hit	hit	hit	miss	miss	miss	miss
行3	miss	hit	hit	hit	miss	miss	miss	miss

4.16 (1) 两路组相联，每路各 16 组，input[0][0] 和 input[0][8] 按 LRU 替换策略恰好被存入不同路缓存，每次访问缓存，input[0][4i] 和 input[0][4i] 不命中，此后三次访问均命中，则命中率为 75%

(2) 增加总大小并不会改善命中率，因为无论增加多少缓存块，对单个缓存块的连续 4 次访问一定是第一次不命中，后三次命中，即命中率仍为 75%

(3) 增加块大小可以改善命中率，因为此时有更多数据被载入同一块中，对单个缓存块可连续访问更多次，而仍然只有第一次不会命中