4-5

1) 页数 $= \frac{2^{65}}{4KB} = 2^{53}$

总空间 = 页数 · 页表条目

$= 2^{53} \cdot 8$

$= 2^{56}$ B

2) 总空间 $= 2^{56-(64-48)} = 2^{40}$ B

3) 多级页表通过分级映射，将大页表分解为多层小页表，从而节省空间

4-6 若用高位作为索引，不同数据可能会映射到同一个组中，从而导致冲突。而如果使用中间位作为标签，那么可能会出现不同数据具有相同标签，从而导致混淆。

4-7 这样可以使缓存系统更有效利用硬件，因为在与虚拟内存交互时，缓存系统的地址映射将更为简单，这也使缓存管理更加容易，因为可以继承页表的地址转换机制。

4-8 (1) $\bar{t} = 1 \times 97\% + 110 \times 3\%$

$= 0.97 + 0.33$

$= 1.3$ 周期

(2) 命中率：$\eta = \frac{64KB}{1GB}$

$\bar{t} = 110 \times \eta + (1-\eta) \times 1$

$= 109.99$ 周期

(3) 只有有较好的局部性，处理器访存性能才会提升，若完全随机访问，只会降低访存性能

(4) 设命中率为 η

$\bar{t} = 110 \times (1-\eta) + \eta \leq 105$

即 $110 - 109\eta \leq 105$

$\eta \geq \frac{5}{109} \approx 4.6\%$

超过 4.6% 的命中率即可

4-9

编号	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	标签位数 Bit	偏移位数 Bit
1	32	4	64	2	32	5	21	6
2	32	4	64	8	8	3	23	6
3	32	4	64	全相联	1	0	26	6
4	32	16	64	1	256	8	18	6
5	32	16	128	2	64	6	19	7
6	32	64	64	4	256	8	18	6
7	32	64	64	16	64	6	20	6
8	32	64	128	16	32	5	20	7

4-10

1) $\bar{t_1} = 0.22 \times (1-p_1) + 100.22 \times p_1$

$\bar{t_2} = 0.52 \times (1-p_2) + 100.52 \times p_2$

若 $\bar{t_1} < \bar{t_2}$

则 $0.22 + 100p_1 < 0.52 + 100p_2$

即 $100p_1 < 0.3 + 100p_2$

即 $p_1 < p_2 + \frac{3}{1000}$

2) $\bar{t_1} = 0.22 \times (1-p_1) + k \cdot 0.22 \times p_1$

$\bar{t_2} = 0.52 \times (1-p_2) + k \cdot 0.52 \cdot p_2$

仍 $\bar{t_1} < \bar{t_2}$

$$0.22 + (k-1)\cdot 0.22\cdot p_1 < 0.52 + (k-1)\cdot 0.12\cdot p_2$$

$$\therefore \quad p_1 < \frac{0.3}{(k-1)\cdot 0.22} + \frac{26}{11} p_2$$

4-11

块地址(Hex)	块地址(bin)	直接索引	2路索引	4路索引	8路索引
0x 1001	0001 0000 0000 0001	0001	001	01	1
0x 1005	0001 0000 0000 0101	0101	101	01	1
0x 1021	0001 0000 0010 0001	0001 ✓	001	01	1
0x 1045	0001 0000 0100 0101	0101 ✓	101	01	1
0x 1305	0001 0011 0000 0101	0101 ✓	101 ✓	01 ✓	1
0x 2ee5	0010 1110 1110 0101	0101 ✓	101 ✓	01 ✓	1
0x ff05	1111 1111 0000 0101	0101 ✓	101 ✓	01 ✓	1

直接索引：5次替换

2路：3次；4路：3次；8路：0次

4-12：A：8个组　　B：16个组.　　一个块能存下4个整数

共访问了9600次

A：$\bar{i}=0$　miss 24

$\bar{i}>0$　24/cycle

miss率：$\frac{24\times 100}{96\times 100} = \frac{1}{4} = 25\%$

65~68 → 0~4
69→72→5~8
73→76→9~12
77→80→13~16
81→84→17~20
85→88→21~24
89→92→25~28
93→96→29~32

33~36
37~40
41~44
45~48
49~52
53~56
57~60
61~64

B：

0~4 ← 65~68
5~8 ← 69~72
9~12 ← 73~76
13~16 ← 77~80
17~20 ← 81~84
21~24 ← 85~88
25~28 ← 89~92
29~32 ← 93→96
33~36
37~40
41~44
45~48
49~52
53~56
57~60
61~64

4-13

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

4-14 1) 共128组，一个块能存8个整数

优化前：每次循环都缺失，共8192次

优化后：每8次缺失一次，共1024次

2) 优化前：首轮j从0~128全部缺失； i=1~3，j不缺失

i=4缺失 ... 共2048次

优化后：1024次

3) 优化前至少 4kB

优化后

4-15

	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	×	×	×	miss	√	√	√
行 1	×	×	×	×	×	√	√	√
行 2	×	×	×	×	×	√	√	√
行 3	×	×	×	×	×	√	√	√

4-16 1) 32个块，每块4个元素

i=0 miss ; i=1,2,3 hit ----

未命中率

2) 提高：缓存大小越大，命中率越高，因为可存储的块数越多

3) 降低：块数减少，命中率下降