4-5 (1) 一页大小 4KB 则页内偏移需 12 位 剩余 52 位用于存储索引 若使用单级页表 则共有 $2^{52}\cdot 8 = 2^{55}$ 字节 $= 2^{15}$ TB

(2) 索引位下降为 36 位 共 $2^{36}\cdot 8 = 512$ GB

(3) 多级页表可以随着进程占用内存的增大而相应增多该进程页表数目，而不是直接分配大量内存给页表.

4-6. 如果高位索引低位标签由于内存地址是连续分配的可能导致多个地址指向同一索引引起大量冲突

4-7. 因为虚拟地址的页内偏移与实际物理地址相同，也就是与组索引十块内偏移相同。在实际访存中可以直接由虚拟地址得到索引位，从缓存中得到对应位置块的标签，并与从页表中得到的物理地址对比，提高了访存速度.

4-8. 1) $\overline{T} = 97\% \times 1 + 3\% \times 110 = 4.27$ 个周期

2) 缓存命中率 $\frac{64KB}{1GB} = \frac{2^{16}}{2^{30}} = 2^{-14}$ 缺失率 $1-2^{-14}$

平均周期 $110\times(1-2^{-14}) + 1\times 2^{-14} \approx 110$ 个

3) 局部性原理说明 cpu 访问寄存器时，总趋于聚集在一片连续区域中，在 0 中由于随机访存指令均匀分布在整个主存中，导致缓存命中率低，违背了局部性原理，访存时间大大提高

4) 设平均命中率为 $k\%$，则 $k\% + (1-k\%)110 \le 105$ $k \ge 4.58$ 即命中率大于 4.58%

3.

9. 根据给出的不同缓存配置，补全下表中缺失的字段。

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

10. 1) A 平均：$t_A = 0.22 + 100p_1$

B 平均：$t_B = 0.52 + 100p_2$ $t_A \le t_B$ $p_1 < p_2 + 0.3\%$

2) A 平均 $t_A = 0.22 + 0.22kp_1$

B 平均 $t_B = 0.52 + 0.52kp_2$ $t_A \le t_B$ $p_1 < \frac{26}{11}p_2 + \frac{15}{11}k = 2.36p_2 + 1.36k$

11. ①直接映射：索引位有 4 bit，即最后一个16进制位：

0 0 0 1
0 1 0 1

替换 0x1001 ← 0x1021

0 0 0 1
0 1 0 1
0 1 0 1
0 1 0 1
0 1 0 1

0x1005 ← 0x1045 ← 0x1305 ← 0x2ee5 ← 0xffd5

共5次

②2路组联，索引位有3位，每路2个

替换：0x1005 ← 0x1305 ← 0xffd5 共3次

0x1045 ← 0x2ee5

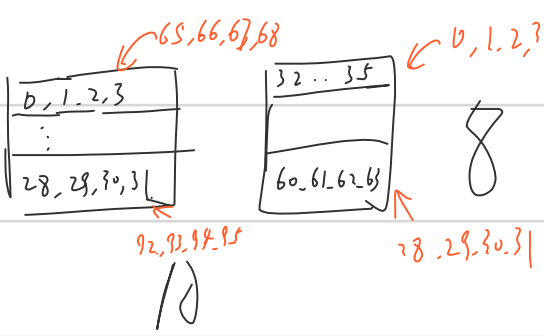
③4路组联：索引位2位，每路4个，但所有访存指向同一索引

故发生 7-4=3次

④8路组联，索引1位，每路8个，无替换

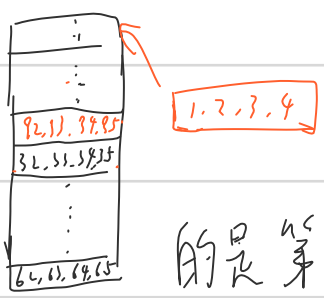
12. 一个 array[i] 大小为4字节，地址为2位

缓存共16块，每块16字节，块内偏移4位，相当于存4个连续数组值



对A而言，2路每路8块，每访问64次数据刷一次新，由于遵从LRU，从第64个元素开始缓存将原第1个元素的缓存块替换。遍历完一遍之后，重新访问arry[0]时，第1组的两路均不匹配，于是遵从LRU将arry[32]…一路从头替换。

故总缺失率为 25%



对B而言，遍历完一遍之后，回到 arry[0]时，仍然无匹配，但替换的是第一组的块，而未替换块 arry[32]…，故只有一半缓存被重复替代

总缺失率为 $\frac{24+16\times 99次}{96\times 100}=5.75\%$

13. 相邻两次访存之间如果地址连续，可以提高缓存利用率，故改为

```
for (int j=0 ; j<128 ;++j){
    for(int i=0 ; i<64 ;i++)
        A[j][i] = A[j][2*i+1];
}
```

14. (1) 共有 $2^{12}/2^5 = 128$ 块 每块 8 个数据

优化前，由于相邻访存 A[j][i] 与 A[j+1][i] 不是相邻的，不在一个块内；而由于组数为128，相邻2个j之间地址相差 64/8=8 个块地址，故第一次j遍历，将128组中的16组重复替换8次，第二次循环时因全被替换过，故又全部缺失，故每次缓存均缺失。共 128×64=8192 次

优化后，相邻访存地址连续，在同一个块内，而j循环同样每次循环都会对新的一块缓存赋值，故每8次访问缺1次，共 128×64/8=1024 次。

(2) 优化前：每次j循环对一个新缓存块访存，将128个块用完，每8次j循环更新一次，共缺失 $\frac{64}{8}\times 128 = 1024$ 次

优化后：仍然是将每个缓存块都占满，不同在于对一个缓存块更新后会连续访问8次，然后继续更新下一个缓存块。总共缺失仍是 $\frac{64\times 128}{8} = 1024$ 次

(3) 优化前：每8次j循环需要128个块缓存，为了这些块不重复，至少要共 $\frac{64}{8}\times 128 = 1024$ 块，容量为 1024×32B=32KB

优化后：由于连续访问，只用1个块即可满足，故共32B

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	✗	✓	✗ 被output换了	✓	✗	✗	✗	✗
行 2	✗	✗	✓	✓	✗	✗	✗	✗
行 3	✗	✓	✗	✓	✗	✗	✗	✗

共有 32B/16B = 2 块，每块4个数

共命中 8 次

16. (1) 512/(16×2) = 16 组，每块存4个数

每访问4次缺失一次，命中率 75%

(2) 不可以，因为缺失与块是否有被替代无关，缓存扩大在块大小不变下只是避免了块被替代的情况

(3) 可以，因为连续地址的访问，块大小越大，同一块存储的数越多，利用率越高，缺失率越小