6. 使用高位作为标签可以提供足够的位数来唯一标识缓存行，可以支持更大的地址范围，而中间位作为标签则可能提供的位数较少，无法唯一标识缓存行，增加了冲突和缓存失效的风险。

7. ① 简化地址转换：位数相同则可以直接使用页偏移的位数，而无需进行额外的地址转换，提高了缓存系统的效率

② 硬件支持：可以更好地利用硬件支持，例如处理器可以直接使用虚拟地址的页偏移来缓存查找，而无需额外的逻辑操作

8. (1) $\overline{T} = 1 \cdot (1-3\%) + 110 \cdot 3\% = 4.27$ 周期

(2) $1GB \gg 64KB$，故每次访问都会导致缓存缺失，$\overline{T} = 110$ 周期

(3) (2)中每次随机访问，导致无法利用局部性原理进行缓存命中，每次访问都会导致缓存缺失，导致访问延时较高

(4) 命中率 η：$1 \cdot \eta + 110(1-\eta) \leq 105$

$\therefore \eta \geq 94.59\%$ 即可

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	标签位数	偏移位数
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

偏移位数 $= \log_2$(块大小)

组数量 = 缓存大小 / 块大小 / 相联度.

组索引位数 $= \log_2$(组数量)

标签位数 = 地址位数 − 偏移位数 − 组索引位数.

10. (1) $T_A = 0.22(1-P_1) + 100 \cdot P_1$, $T_B = 0.52(1-P_2) + 100 \cdot P_2$ A优于B 则 $T_A < T_B$

即 $99.78P_1 < 99.48P_2 + 0.30$

(2) $T_A = 0.22(1-P_1) + 0.22K P_1$, $T_B = 0.52(1-P_2) + 0.52K P_2$ $T_A < T_B$

即 $22(K-1)P_1 + 22 < 52(K-1)P_2 + 52$

$(K-1)(11P_1 - 26P_2) < 15$

11. 块地址即没有偏移量，因为共有16个块，故只需要最多4位索引，即考察最后4位即可

0001 直接映射：5次块替换

0101

0001 2路组相联：考虑后3位：3次块替换

0101

0101 4路组相联：考虑后2位：3次块替换

0101

0101 8路组相联：考虑后1位：0次块替换

001 001

101 101

01 01 01 01

1 1 1 1 1 1 1

12. LRU: least-recently used 将最近最不常用的替换出去

256/16 = 16 = 2^4：4位索引，16个组

~~i: 0~95 即 0000000 ~ 101 1111~~

缓存B：4位索引，~~tag: 000~101，共计6种 tag，每次改变 tag 就会发生1次 miss~~，tag为0或1

4位数据。

32位：array[] 的地址

从 0, 4, 8, ..., 380

即 0000, 0100, ... 10111100

索引从 0000 ~ 1111

索引 1000 ~ 1111 的 tag 只有0，而索引 0000 ~ 0111 的 tag 为 0和1交替

01000000 ~ 01111100

128 ~ 252

miss: 00000000, 00001000, ..., 00111000

10000000, 10001000, ..., 10111000

共计 miss 7×2 = 14个

10111100

index 偏移

缺失率为 $\frac{14}{96} \times 100\% = 14.6\%$

缓存A：2路 3位索引，2位tag，4位数据 索引：000~111，tag：00、01、10

考虑索引 000：	路1	路2		
①	01_000	10_000	此时输入 00_000	miss 1次
②	00_000	10_000	输入 01_000	miss 1次
③	00_000	01_000	输入 10_000	miss 1次
④	10_000	01_000		

同一个i下

按 LRU 替换

总计 miss 3×8 = 24次

缺失率 $\frac{24}{96} \times 100\% = 25\%$

13. for (int j=0; j<128; ++j) {

for (int i=0; i<64; ++i) {

A[j][i] = A[j][i] + 1;

}

} // 这样可以连续访问 A[p][q] 和 A[p][q+1] 对应的连续的地址

14. (1) 索引数 = $\frac{4KB}{32Byte}$ = 128, 7位索引, 5位偏移

优化前: j: 64个地址连续 j: 128组不同的连续地址

而块大小只有32个字节, 说明同一个j中的索引会有差异, 即出现强制失效.

缺块次数: $128 \times \frac{64}{32} = 256$ 次

优化后: $128 \times \frac{64}{32} = 256$ 次

(2) FIFO: 先进先出 总计 1个组 128路 不需要索引

优化前: 恰j=128 对应 128路, i=0: 全miss i=1 全不miss ... i=32: 全miss ... i=63 全不miss

缺块次数: 128×2=256次

优化后: 仍是 256次

(3) 8KB

15. 二维数组 行间地址不连续, 列间地址连续 数据: 4位, 索引: 1位

input [0][0] ~ input [0][3] 地址: 0000 0000 、 0000 0100 、 0000 1000 、 0000 1100 .

索引 其它

故除了第一个miss都hit

	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

output 是固定列数, 改变行数

index

output [0][0] ~ output [3][0]: 地址: 0100 0000 、 0101 0000 、 0110 0000 、 0111 0000

注意 tag发生了改变, 故全部 miss

16. (1) 索引: $512 \div 2 \div 16 = 16 = 2^4$, 4位索引, 4位数据偏移. 两路组且用LRU, 故分别在 input[0][] 和 input[1][]

input[0][0] ~ input[0][127]: 0000 0000 ~ 0111 1111

在索引达到 0000、0001 .... 0111 时发生 miss, 共计8次

∴ 命中率 $= 1 - \frac{8}{128} = 93.75\%$

(2) 不会, 因为不改变块大小的前提下增加总大小, 只会增加索引的位数,

索引仍会在 0000 ~ 0111 发生 miss, 故不改变命中率

(3) 可以, 因为增加块大小会将索引向左平移, 例如将块大小翻3倍, 索引为 0000 ~ 0011

只会发生4次miss, 提高了命中率.