6.

如果将组索引作为高位，在一个组连接的缓存中，有可能会导致数据数据存放在同一组的不同位置，对于其他的组没有利用，对于一个空间局部性好的程序，其会从连续的空间读取数据，考虑索引位只有 1 位的极端情况，高位索引时，每次读取数据都会将前一次的数据覆盖，这也就导致缓存只保存着一个块大小的数据，在一般情况下，对于组相连得到缓存，高位索引会无法利用其余组的空间。

改用中间位做索引，同一组中的块不再是连续的，这样可以保证缓存中的所有组都能被有效的利用。

7.

在真实缓存设计中，让地址的组索引位数和块偏移位数之和同虚拟内存系统页偏移位数相同的主要原因是为了优化缓存的命中率。

缓存的命中率是指缓存中存储的数据与请求的数据匹配的概率。为了提高缓存的命中率，需要将缓存设计成可以根据请求的数据地址自适应地选择合适的缓存行。虚拟内存系统将页面（page）的大小设置为一个固定的大小，通常是 4KB 或者 2MB，并且每个页面都被映射到物理内存中的一个地址。因此，虚拟内存系统中的页偏移位数就是页面的地址偏移量。
如果缓存的组索引位数和块偏移位数之和与虚拟内存系统页偏移位数相同，那么缓存的设计可以更加自适应。具体地说，当请求的地址是一个页面时，缓存可以将页表（page table）映射到物理内存中的页框（page frame）的位置，并根据页表中找到的页框索引（page frame index）来选择缓存行。如果请求的地址是一个页面，但是缓存的组索引位数和块偏移位数之和与虚拟内存系统页偏移位数不同，那么缓存就需要通过页表（page table）来查找正确的缓存行。这个过程需要更多的时间，从而降低了缓存的命中率。

因此，让地址的组索引位数和块偏移位数之和同虚拟内存系统页偏移位数相同可以提高缓存的命中率，从而提高系统的性能和响应速度。

8.

(1)延时=0.97*1+0.03*110=4.27

(2)延时=(1/16384)*1+(16383/16384)*110=109.9933472

(3)局部性原理通常表现在两方面：时间局部性，程序中的某条指令一旦执行，不久后该指令可能再次执行；某数据被访问过，不久后该数据可能再次被访问。就是即将访问信息很可能就是目前正在使用的信息。空间局部性，一旦程序访问了某个存储单元，不久后，其附近的存储单元也将被访问，就是即将访问信息和现在正在使用的信息在空间上相邻或者临近。
如果一个程序中的数据不具有局部性，程序读取数据是随机的，这会造成缓存命中率的大幅降低，也就造成缓存失效，相当于处理器还是需要访问速度慢的内存或是磁盘，对于具有局部性的程序，其大部分时间都在执行少数的代码，访问的数据也在一个较小的地址空间，因此建立一个高速缓存来存放这些最容易被处理器访问的数据，此时缓存命中率会很高，也就更多次的读取速度快的缓存，从而加速程序

(4)解方程：x+(1–x)*110=105，得 x=0.04587，也就是缓存命中率高于 4.587%

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

10.

(1)满足(1–p1)*0.22+100p1<(1–p2)*0.52+100p2，即 99.78*p1<0.3+99.48*p2

(2)满足(1–p1)*0.22+0.22k*p1<(1–p2)*0.52+0.52k*p2，即 0.22*(k–1)*p1<0.3+0.52*(k–1)p2

11.

首先将块地址转为 10 进制，按顺序分别为 4097，4101，4129，4165，4869，12005，65285，对于 16 取余分别得到 1,5,1,5,5,5,5，由于数据缓存初始为空，不论直接映射或是任意路组相联，第一次访问缓存不同的块时均不会发生替换，对于直接映射，在第一次访问后缓存的块每一次访问相同的块都会发生替换，因此 0x1021,0x1045,0x1305,0x2ee5,0xff05 均会发生替换，替换次数是 5，对于 2 路组相联，前两次访问对 16 取余相同的块时不会发生块替换，之后的会发生块替换，因此最后的 0x1305,0x2ee5,0xff05 会发生块替换，替换次数是 3，对于 4 路组相联前四次访问对 16 取余相同的块时不会发生块替换，因此只有最后的 0xff05 会发生块替换，替换次数是 1，对于 8 路组相联前八次访问对 16 取余相同的块时不会发生块替换，因此不会发生块替换

12.

缓存 A 有 8 个块，2 路组相联，缓存 B 有 16 个块，直接映射，任意一种情况，数据除了低 8 位，均为 tag，由于数据为整型数据，占四个字节，因此对于整型数组，下标+1 对应地址+4，当 tag 位相同时，对应了 4*16=64 个不同数组元素，96/64=1.5

对于直接映射组相联缓存，也就是缓存 B，块内偏移为 4 位，只有 0–31 号数组元素和 64–95 号数组元素会发生 index 相同但是 tag 不同的情况也就是缓存失效，因此对于 32–63 号数组元素，在初始缓存为空的条件下，只有 j=0 的循环中每 4 次访问缓存都会发生 1 次缓存失效，其次数为 32/4=8 次，但是对于 0–31 号数组元素和 64–95 号数组元素，在每次循环中，每 4 次访问缓存都会发生 1 次缓存失效，因此总缓存缺失次数为 (32/4)+((32/4)+(32/4))*100=1608 次，缓存失效率为 0.1675

对于 2 路组相联缓存，也就是缓存 A，块内偏移为 4 位，index 为 3 位，在任何情况下在连续四次访问缓存中，都会发生一次缓存失效，所以缓存失效次数为 9600*(1/4)=2400 次，缓存失效率为 0.25

13.

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

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

14.

(1)因为缓存大小为 4KB，即 2^12 字节，块大小为 2^5 字节，因此，快个数为 2^7 个，index 为 7 位，地址位数为除了低 12 位的所有位，对于优化后的程序，在任意循环中，每连续 8 次访问缓存中都会发生一次缓存失效，因此失效次数为(64*128)*(1/8)=1024，对于优化前的程序，每次访问缓存都会发生缓存失效，因此失效次数为(64*128)=8192 次

(2) 对于优化后的程序，在任意循环中，每连续 8 次访问缓存中都会发生一次缓存失效，因此失效次数为(64*128)*(1/8)=1024，对于优化前的程序，每 2 次访问缓存都会发生 1 次缓存失效，因此失效次数为(64*128)/2=4096 次

(3)对于优化后的程序，index 最小为 3 位，即缓存大小为 2^8 字节，对于优化前的程序，需要 10 位 index，即缓存大小为 2^15 字节

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	hit	hit	hit	miss	hit	hit	hit
行 1	miss	hit	hit	hit	miss	hit	hit	hit
行 2	miss	hit	hit	hit	miss	hit	hit	hit
行 3	miss	hit	hit	hit	miss	hit	hit	hit

16.

(1)块大小为 16 字节，块个数为 16 个，2 路组相联，因此 index 为 4 位，块内偏移量为 4 位，每 4 次访问缓存发生一次访存失效，因此缓存命中率为 75%

(2)因为缓存总大小增加，而块大小增加，因此快个数变多，index 增加，但是块偏移量不变，依旧占据数据的低四位，在本程序中，如果 index 增加一位，达到 5 位，这样偏移量位数和 index 位数总和为 9 和刚好能保证 128 个整形数组元素全部被取到，也就相比优化前的缓存，能减少一般的缓存失效次数，因此，增加缓存大小可以增加缓存命中率

(3)因为缓存的块大小变大而缓存大小不变，因此缓存的块内偏移量增加，而块个数减少，index 位数减少，若块大小变大 1 位，每 8 次访问缓存发生一次访存失效，因此访存失效率减少，因此增大缓存的块大小可以增加缓存命中率