

5.

- 1) 64 位虚拟地址空间意味着地址空间中的每个地址都可以由 64 位二进制数表示。如果一个页表项占用 8 字节空间，那么一个页表可以存储  $2^{64} / 2^{12} = 2^{52}$  个页表项（每个页表项对应一个 4KB 的页）。因此，一个单级页表系统需要  $2^{52} * 8$  字节的空间来存储页表。
- 2) 在一个单级页表系统中，每个页表项对应一个 4KB 的页，因此需要占用 8 字节的空间。如果只使用 48 位的虚拟地址空间，那么一个页表可以存储  $2^{48} / 2^{12} = 2^{36}$  个页表项。因此，一个单级页表系统需要  $2^{36} * 8$  字节的空间来存储页表。
- 3) 通过使用多级页表，系统可以灵活地分配内存空间，只为实际使用的部分分配页表项，而不是为整个地址空间分配页表项。这样可以有效降低虚拟内存系统的存储开销，因为只有实际使用的内存部分才需要占用页表项的存储空间，而未使用的部分可以被省略。

6.

组索引用于确定数据在缓存中的组（也称为集合）的位置，而标签用于唯一标识数据。使用中间位作为组索引可以提高缓存的映射能力，即可以将更多的数据映射到不同的组中。这有助于减少缓存冲突和提高缓存的命中率。

7.

保持相同的组索引位数可以确保虚拟内存系统的页面与缓存系统的组之间存在一种固定的映射关系。这样可以简化地址转换的逻辑，提高缓存系统的访问效率。

8.

- 1) 平均延时= $1 * 0.97 + 110 * 0.03 = 4.27$
- 2) 命中的概率为  $64KB / 1GB = 1 / 16384$   
平均延时= $1 / 16384 * 1 + 16383 / 16384 * 110 = 109.99$
- 3) 局部性原理是指程序在一段时间内对某些数据的访问呈现出集中在某些区域的趋势，这些区域包括时间局部性和空间局部性。时间局部性指的是程序在短时间内对同一数据的访问次数很多，空间局部性指的是程序在访问一个数据时，很可能还会访问其相邻的数据。局部性原理影响处理器的访存性能，因为缓存的设计就是基于这个原理，利用缓存中已经存在的数据来减少处理器访问内存的次数和延迟，从而提高访存性能。  
1 中命中率特别高，说明其程序运行过程中的访存操作是符合局部性原理的，所以访存效率非常高。  
2 中的程序则持续访问一个 1GB 数组中的随机位置，访问的位置没有任何局部性，所以缓存命中率很低，无法提高访存性能。  
这就说明缓存仅在满足局部性原理，即其访存操作符合时间局部性或者空间局部性，才能够保证访存效率的提升。
- 4) 平均延时= $1 * x + 110 * (1 - x) = 110 - 109x < 103$   
 $x > 7 / 109 \approx 6.42\%$   
只要命中率高于 6.42% 就能获得性能收益了。

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	<u>32</u>	<u>5</u>	<u>21</u>	<u>6</u>
2	32	4	64	8	<u>8</u>	<u>3</u>	<u>23</u>	<u>6</u>
3	32	4	64	全相联	1	<u>0</u>	<u>26</u>	<u>6</u>
4	32	16	64	1	<u>256</u>	<u>8</u>	<u>18</u>	<u>6</u>
5	32	16	128	2	<u>64</u>	<u>6</u>	<u>19</u>	<u>7</u>
6	32	64	64	4	<u>256</u>	<u>8</u>	<u>18</u>	<u>6</u>
7	32	64	64	16	<u>64</u>	<u>6</u>	<u>20</u>	<u>6</u>
8	32	64	128	16	<u>32</u>	<u>5</u>	<u>20</u>	<u>7</u>

10.

- 1) A 的平均延时为  $0.22+100*(1-p1)$   
B 的平均延时为  $0.52+100*(1-p2)$   
A 比 B 优的条件为  
 $100.22-100*p1 < 100.52-100*p2$   
 $p2-p1 < 0.003=0.3\%$
- 2) A 的平均延时为  $0.22*p1+0.22*k*(1-p1)=0.22(p1+k-p1*k)$   
B 的平均延时为  $0.52*p1+0.52*k*(1-p1)=0.52(p2+k-p2*k)$   
A 比 B 优的条件为  
 $0.22(p1+k-p1*k) < 0.52(p2+k-p2*k)$   
 $0.22*p1*(1-k) < 0.3*k+0.52*p2*(1-k)$   
 $0.52*p2*(k-1)-0.22*p1*(k-1) < 0.3*k$   
 $0.52p2-0.22p1 < 0.3*k/(k-1)$

11.

- 1) 直接映射：  
组索引为 4 位，所以第一个数据先放进 1 号组，第二个数据 5 号组，然后后面所有的数据因为都和前面两个数据的组索引号相同，所以全部需要替换。  
共发生 5 次替换。
- 2) 2 路组相联  
组索引为 3 位，但是相联度为 2，所以第一个数据先放进 1 号组第一个，第二个数据 5 号组第一个，第三个数据 1 号组第二个，第四个数据 5 号组第二个，第 5、6、7 个数据都需要替换。  
共发生 3 次替换。
- 3) 4 路组相联  
组索引为 2 位，此时所有数据后两位已经都相同，所以都共用一个组。相联度为 4，所以总共发生  $7-4=3$  次替换
- 4) 8 路组相联  
组索引为 1 位，相联度为 8，一共才 7 个数据，所以不发生替换。  
共发生 0 次替换。

12.

调用的次序为 array 的 0->95,0->95,...,0->95 (共 100 次)。一共 16 个块，每个块容纳 4 个 int。  
对于 A:

2路组相联，共8个组。0-3存进第一组第一个，4-7存进第二组第二个.....32-35存进第一组第二个，.....60-63存进第八组第二个；64缺失，但是因为LRU，所以替换的是第一组第一个，把第二个留在了组里，导致在i++的时候，每组的都是块标签为1、2的而需要的是0的，故缺失，所以又重来一遍。综上，每4个就有一个缺失，缺失率25%。

对于B:

直接映射，共16个组。一次0-95的遍历，对于64个块，会把0-31号块覆盖成64-95号块，但是32-63都还留着，所以发生缺失的次数为 $1*(96/4)+99*(96/32/4)=1608$ ，所以缺失率为 $1608/9600=16.75\%$

13.

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

14. (本题没有说A是什么数据类型，我默认为每个占用1B)

1) 块内偏移5位，块数量128个，组索引为7位。

优化前:

i有64个，但是块内偏移一共才32个，所以128个块在遍历一遍j的同时每一个都被覆盖一遍，所以一直在错，缓存缺失次数 $64*128=8192$ 次。

优化后:

仅有在j mod 64改变的时候才会缓存缺失，而j不变i在变的时候一直都处于hit的状态，所以缓存缺失次数为 $1*128=128$ 次

2) 块内偏移5位，块数量1个，组索引为0位。

优化前:

在需要访问64的时候，64在前一次访问缓存刚好被踢出去，所以还是一样每次都在错，缓存缺失次数 $64*128=8192$ 次。

优化后:

仅有在j mod 64改变的时候才会缓存缺失，而j不变i在变的时候一直都处于hit的状态，跟刚才一样，缓存缺失次数为 $1*128=128$ 次

3) 都至少要8KB大小，以此保证块总数为256个。

15.

	input				output			
	列 0	列 1	列 2	列 3	列 0	列 1	列 2	列 3
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
行 4	miss	hit	hit	hit	miss	miss	miss	miss

块数量2，直接映射，块大小16字节，可以存储4个int。

16.

1) 32个块，两路组相联，16个组，块大小16字节，可以容纳4个int。

数组的行控制第八位，列控制 1-7 位。块索引 4 位，块内偏移 2 位。所以实际上行 0 与行 1 的同一列是放在同一个组里面的。

随着 i 从 0 到 63，恰好填满所有 32 个块，再往后再依次把这 32 个块重新替换一遍。当然，每 4 次有一次在替换，余下的都是 hit，所以命中率是 75%。

- 2) 不可以。即使增加缓存总大小，依然每次替换块都要缺失一次，所以还是 75%。
- 3) 可以。如果增加块大小至每个块可以容纳 x 个 int (即块大小为  $4 \times x$  Byte)，那么增加的块容量会成为 hit 状况下的块偏移，所以缺失率会降低至  $1/x$ ，命中率会提升至  $(x-1)/x$