6. 为什么缓存一般用地址中间位段作为组号，高位作为标签而不是用高位做组号，中间位作标签？

因为地址的高位随地址改变时变化比较慢，作为标签可以减小查找时间，而中间位变化更多，作组索引可以更容易地减少缓存冲突，把不同地址有效分配至各组。

7. 假设页的大小为4KiB，则 Page offset 有12位，而按此设计，~~地址~~用于缓存的组号和字段和 Byte offset 也是12位（假设为直接映射，数据块大小为单字），因此 Cache 容量也为4KiB。

因此，这么做可以使页面内所有数据被缓存在缓存的不同组内，减小了页与缓存的冲突，提高了缓存效率。

8. (1) 平均访问延时设为 $T_1$

$T_1 = 97\% \times 1\,\text{cycle} + 3\% \times 110\,\text{cycles} \approx 4.27\,\text{cycles}$

(2) 假设平均访问延时为 $T_2$

因为程序是完全随机的，所以命中率 $= \frac{64\,\text{KB}}{1\,\text{GB}} = \frac{64 \times 2^{10}}{2^{30}} = \frac{1}{16384}$

$T_2 = \frac{1}{16384} \times 1\,\text{cycle} + (1 - \frac{1}{16384}) \times 110\,\text{cycles}$

$\approx 109.99\,\text{cycles}$

所以命中率很低，延时仍为缓存缺失时情况。

(3) 由(1)可知，对于大部分程序，都有时间和空间局部性，当一个数据项被访问时，不久可能再次被访问，其地址相邻的数据项也可能被很快访问，此时使用缓存技术可以有效降低访问延时，但对于(2)中的程序，则无法利用局部性原理，缓存甚至多层级化存储手段也无能为力，此时命中率无法提升。此外，这两种情况也提示我们在编写程序时要利用好局部性原理进行优化。

deli得力

Date. /

(4) 设命中率为 $x$，则有 $x + 110(1-x) \leq 105$

$x \geq 4.59\%$

所以平均缓存命中率的阈值大于4.59%才能使L1缓存有收益。

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 |

对地址位数为32 bit，缓存为4KB，块为64 Byte的情况，

$4KB = 4 \times 2^{10}$ Byte $= 2^{12}$ Byte，64 Byte $= 2^6$ Byte，数据块数量 $= \frac{2^{12}}{2^6} = 2^6$，共有64个数据块

2路：$\frac{64}{2} = 32$ 组　8路：$\frac{64}{8} = 8$ 组　全相联：1组

其余情况同理

10.(1)设A平均内存访问时间为$T_A$，B为$T_B$

$T_A=(1-p_1)\times 0.22+100p_1=99.78p_1+0.22$

$T_B=(1-p_2)\times 0.52+100p_2=99.48p_2+0.52$

当$T_A<T_B$时，有$99.78p_1-99.48p_2<0.3$

此时A平均访问内存时间优于B.

(2) 则$T_A=(1-p_1)\times 0.22+0.22k\cdot p_1$

$T_B=(1-p_2)\times 0.52+0.52k\cdot p_2$

$T_A<T_B \Rightarrow (k-1)p_1-\frac{26}{11}(k-1)p_2<\frac{15}{11}$

$p_1-\frac{26}{11}p_2<\frac{15}{11(k-1)}$

当满足上述条件时A平均访问时间优于B.

11. ①直接映射，$16=2^4$，需要4位组号31，共16组

则0x1001、0x1021会被映射至同一个组，发生一次替换

0x1005、0x1045、0x1035、0x2ee5、0xff05被映射至同一个组

发生4次替换

∴一共发生5次块替换.

②2路组相联

0x1001、0x1021，由于组内有2个数据块，不替换

0x1005、0x1045、0x1035、0x2ee5、0xff05：组内有2个数据块，替换3次

∴一共发生3次替换.

③8路组相联

0x1005、0x1045、0x1035、0x2ee5、0xff05，替换一次

一共替换1次.

④ 8路组相联：不发生替换。

12、缓存A：16个数据块，8个组，一个块为16字节
缓存B：16个数据块，16个组

```
#define N 100
int32_t array[96];
for(int32_t i=0; i<N; i++){
    for(int32_t j=0; j<96; j++){
        array[j] = i*j; }}
```

int32_t：32位整数，一个整数占用4 Byte。
96×4=384 Byte

1. 直接映射：假设字节地址从0开始，则对第一个元素，字节地址为0~3，对第二个元素，为4~7，以此类推...
因此前4个元素所包含的字节地址对应为块地址应为0，此后为1，2，3，4，... 故最后4个元素的块地址为23
∴块号=块地址 mod 块数量
∴前16个块地址分别被映射至16个块，最后8个被映射至和前8个块地址相同的块。
LRU替换：最近最少使用，∴直接映射第一个循环缓存miss率为100%，之后由替换规则缺失率为$\frac{2}{3}$... 个块

2. 2路组相连：
块地址相同，为0，1，2，...，23
∵有8个组，∴块地址0~7，8~15，16~23的映射规律相同
在这种情况和LRU替换规则下，缺失率仍为100%
缓存块

Date. /

因为每个数据块中有4个数组元素，所以实际缓存缺失率还应乘为块缺失率为4

∴缓存A（2路组相联）：cache miss rate = 1×25% = 25%

缓存B（直接映射）：cache miss rate = $(\frac{1}{100} + \frac{99}{100} \times \frac{3}{3}) \times 25\%$

$\approx 16.75\%$

∴在此情况下，直接映射缓存缺失率反而更低。

13、
```
for(int i=0; i<64; ++i){
    for(int j=0; j<128; ++j){
        A[j][i]=A[j][i]+1; }}
```
A[p][q]与A[p][q+1]相邻，∴优化后的代码应为

```
for(int j=0; j<128; j++){
    for(int i=0; i<64; i++){
        A[j][i]=A[j][i]+1; }}
```
这样访问时数组元素变为相邻。

14. 1) 4KB直接映射，块大小为32字节：有$2^7$个数据块，即有128组。

共访问64×128 = $2^{13}$个元素。

二维数组A：

A[0][0] A[0][1] --- A[0][63]

⋮ ⋱ ⋮

A[127][0] ------ A[127][63]

A[0][0] --- A[0][31] A[0][32] --- A[0][63]

⇒ 数组在内存中的存放方式

假设块地址和字节地址均从0开始

∴A[0][0]到A[0][31]对应的块地址为0，A[0][0]到A[15][63]对应的块地址为从0到127

优化前：缺失率为100%

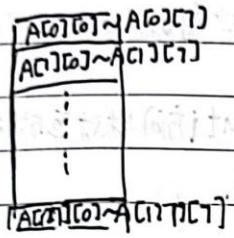
Date. /

优化后，变为连续访存，访存块地址规律为 0, 1, 2, …, … 1023

此时缓存缺失率为 100% × $\frac{1}{8}$ ≈ 12.5%

2) 全相连缓存，块大小为32字节.

未优化第一个内循环后，缺失率为100%，完成后的缓存情况：




则第二个循环为 A[0][1] → A[127][1]

此时没有发生缓存缺失

当 i = 8 时，上述情况重复，缓存中所有块再次更新

∴ 缓存缺失率为 $\frac{1}{8}$ × 100% = 12.5%

② 优化后：首先，A[0][0] 出现缺失，然后 A[0][1] 到 A[0][7] 均命中

以此类推，A[15][56] 至 A[15][63] 对应第128个数据块，出现1个缺失，其余7次命中，缓存装满


此后访问 A[16][0]，根据FIFO替换策略，

A[0][1]~A[0][7] 对应的数据块被替换

∴ 缓存缺失率 = 12.5%

③ 对未优化情况，块地址的规律为 0, 8, 16, … 1016, 0, 8, 16, …, 1016, … (重复8次), 1, 9, …, 1017, …

（A[0][8]）

若不出现其他任何缺失，则至少需要1024个缓存块

缓存容量 = 1024 × 32 = 32 KB

② 优化后：块地址访问规律为 0, 0, 0, …, 0, 1, 1, 1 …, 1, …

（8个0，8个1）

1023, 1023, …, 1023

（32个）

若不出现其他任何缺失，只需1个缓块块，即最小缓存容量为

32 Byte

15. int input[4][4], output[4][4];

```
for(int i=0; i<4; ++i){
    for(int j=0; j<4; ++j){
        output[j][i] = input[i][j];}}
```

input起始地址为0x0, input访问时对应的块地址为0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3

output起始地址为0x40, 即十进制为64, output访问时对应的块地址为4,5,6,7,4,5,6,7; 4,5,6,7,4,5,6,7.

缓存总大小为32字节, 有2块,

第一个循环中, input和output同时被映射至第一个缓存块中, 将导致替换, 而第2次内层循环则分别在两个组中, 以此类推

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

16. int input[2][128];

```
int sum=0;
for(int i=0; i<128; i++i){
    sum += input[0][i] * input[1][i];}}
```

1) 块大小16字节, 对应4个整型元素, 缓存共有16个组, 每组2个数据块. 每次循环迭代时, input[0][i]和input[1][i]将对应的地址将被映射到相同的缓组中, 连接下来4次中仅缺失第1次, 其余命中.

∴程序缓存命中率为75%

2) 增加缓存总大小不可以改变命中率, 因为缓在第一次循环时, 缓存的块缺失率一定为100%, 即使增加缓存总大小也不能改变初始缓存为空的情况

3) 增加缓存块大小可以改善命中率, 因为更多的数组元素可以被映射到同一块中, 使缓存块更新后之后的命中数增多. 举例如如果块大小变为32字节, 则对应8个元素, 命中率增加为7/8 = 87.5%.