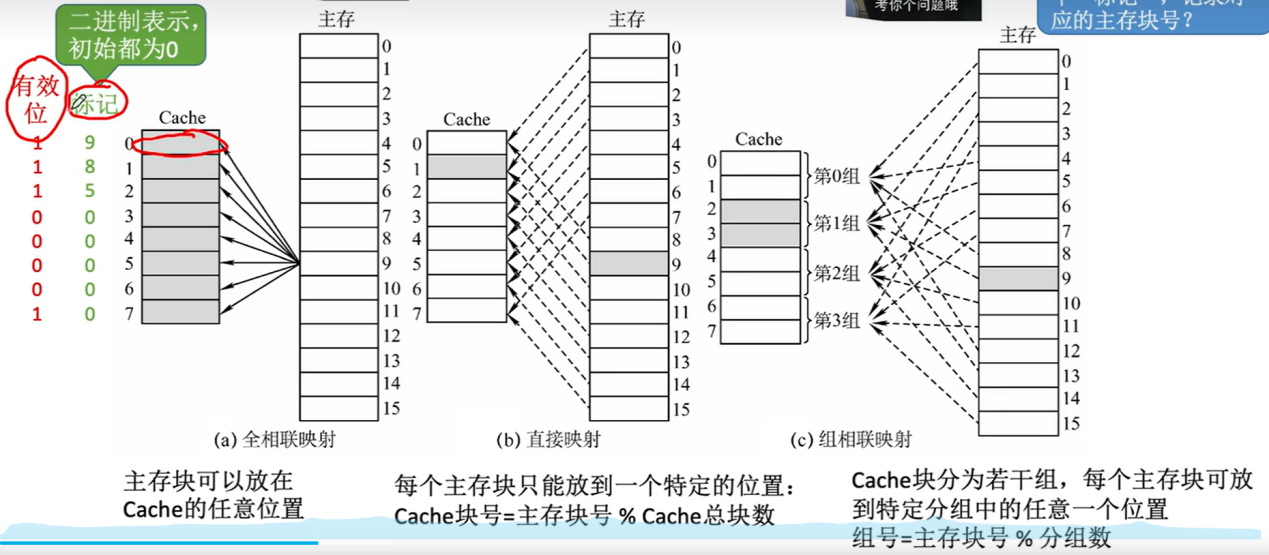
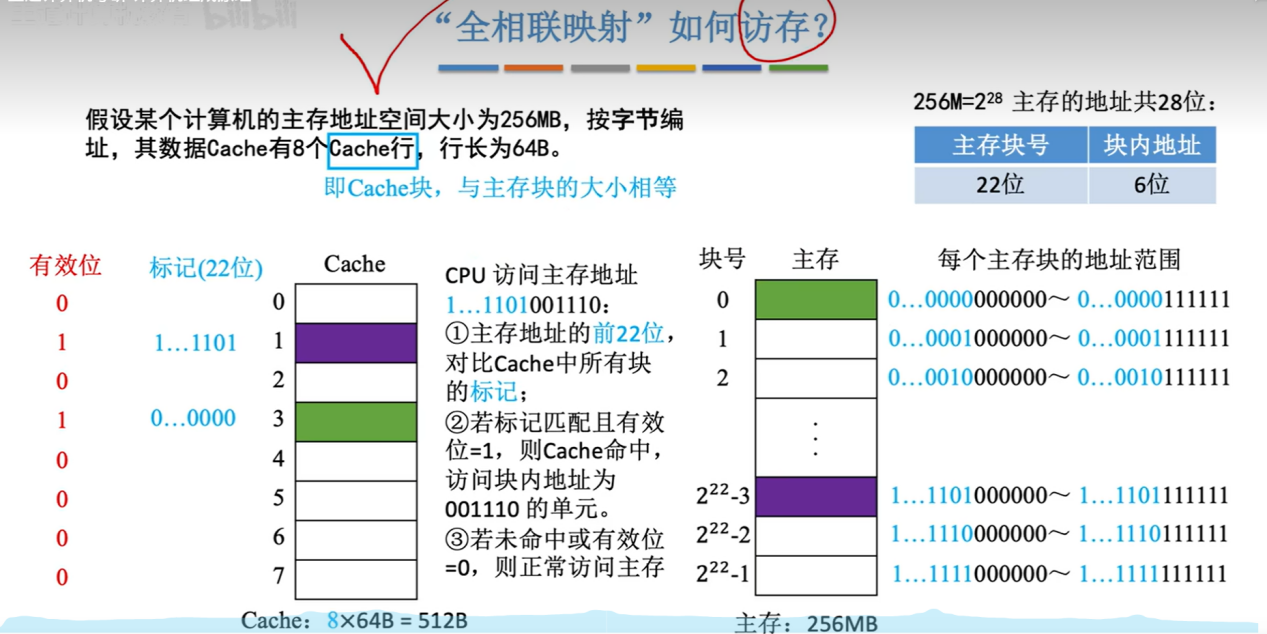
# Cache

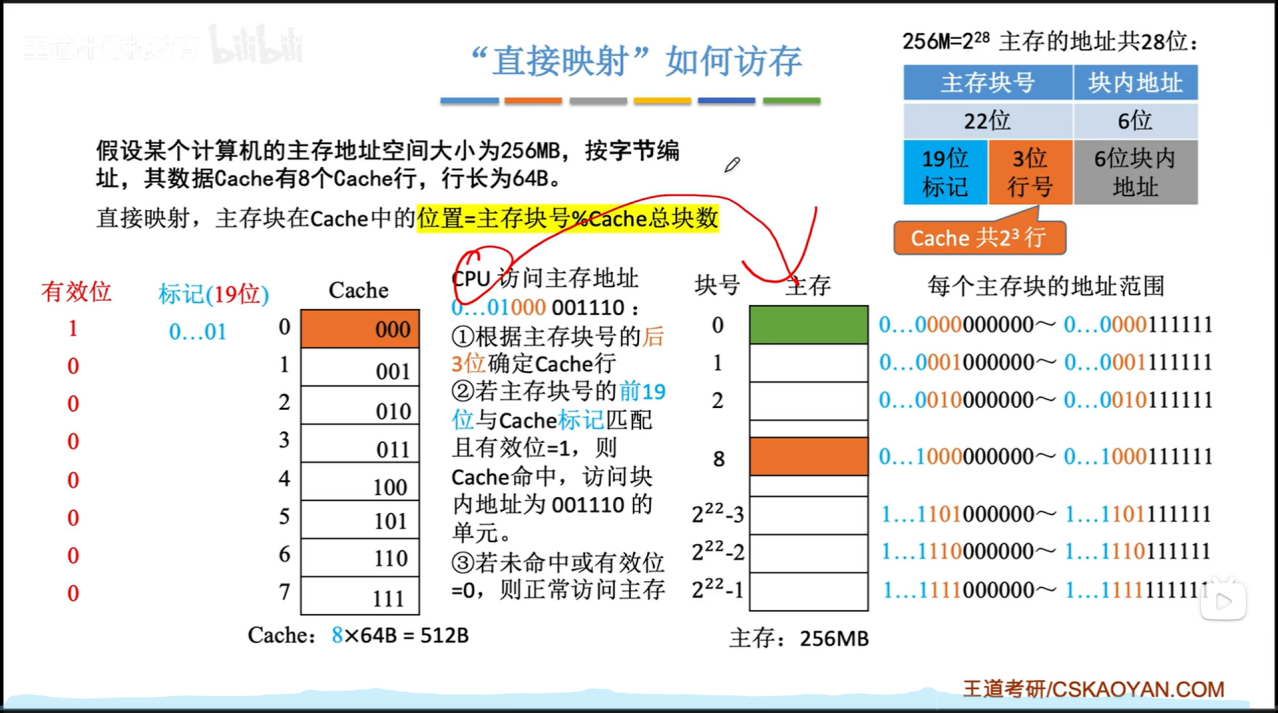
## 地址映射



### 全相联映射

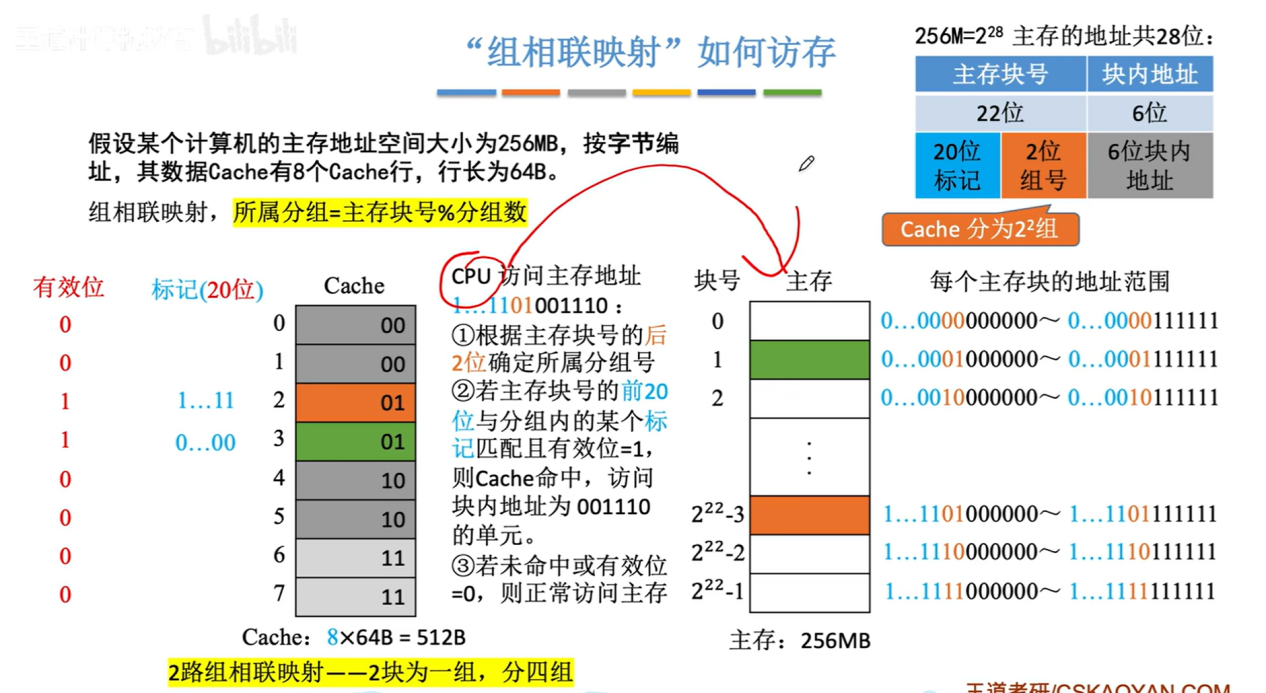


### 直接映射



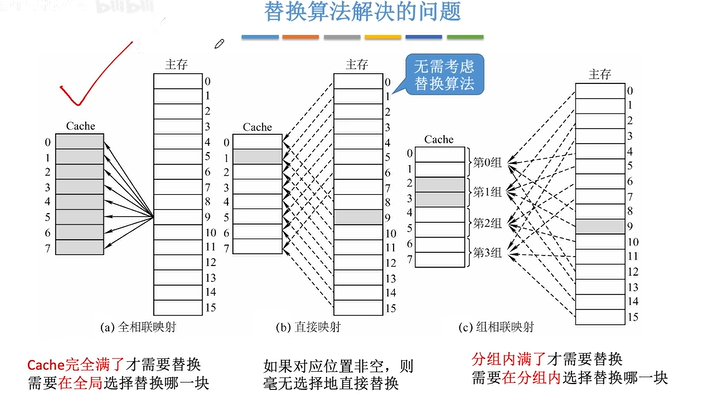
### 组相联映射

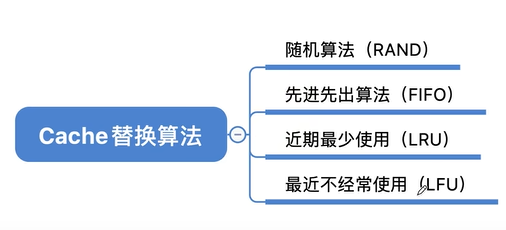
组间采用直接映射，组内采用全相联映射。



## 替换算法

注意每访问一个主存块，必须立即调入 Cache。





### 随机算法（RAND）

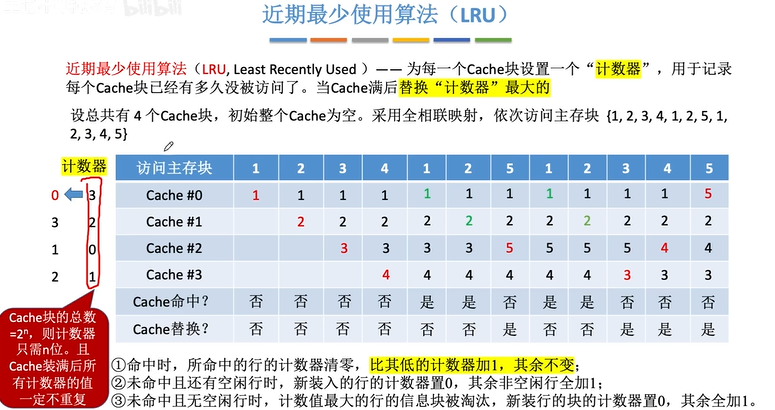
若 Cache 已满，随机选择一块替换。



### 先进先出算法（FIFO）



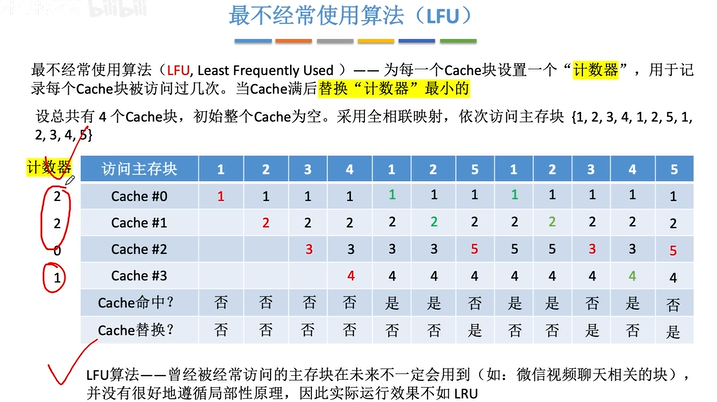
### 近期最少使用（LRU）



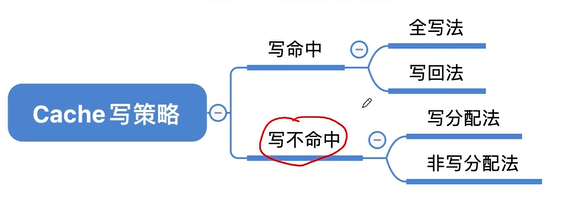
该算法考虑到了 “局部性原理”，近期被访问过的主存块，在不久的将来也有可能会再次被访问，因此淘汰最久没访问过的块是合理的。

若被频繁访问的主存块数量 > Cache 行的数量，则有可能发生 “抖动”。

### 最不经常使用（LFU）



## Cache 写策略



### 写命中的处理方法

**写命中**（Write Hit）是缓存（Cache）操作中的一个概念，指的是当需要向主存写入数据时，所要操作的地址 **已经在缓存(Cache)中存在** 的情况。

**写回法（write-back）**是指当 CPU 对 Cache 写命中时，只修改 Cache 的内容，而不立即写入主存，只有当此块被换出时才写回主存（可以为每个Cache行设置一个修改位）。

**全写法（write-through）**是指当 CPU 对 Cache 写命中时，必须把数据同时写入 Cache 和主存，一般使用写缓冲（write buffer）。

### 写不命中的处理方法

**写分配法（write-allocate）。**加载主存中的块到Cache中，然后更新这个Cache块。

**非写分配法（not-write-allocate）。**只写入主存，不调块。

### 联系

**写回法搭配写分配法 ，全写法搭配非写分配法。**

现代计算机多采用多级 Cache 结构，各级 Cache 之间常采用 “全写法 + 非写分配法”，Cache和主存之间通常采用 “写回法 + 写分配法”解决数据一致性问题。

写回法可以减少访存次数，但存在不一致的隐患 ；全写法（写直通法）安全性好。

## 题目类型一 Cache 容量计算

### 引入

[某存储系统中，主存容量是 Cache 容量的4096倍，Cache被分为64个块，当主存地址和Cache地址采用直接映像方式的时候，地址映射表的大小应为（）。（假设不考虑一致维护和替换算法位）。 - 知乎](https://zhuanlan.zhihu.com/p/14851079778)

本来不应该浪费时间写这些题目的，但这个知识点出现次数比较多，所以总结一下。

**根据王道书上的说法，Cache 的组成如下 ：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 有效位 | 脏位 | 替换控制位 | 标记(Tag) | 数据位 |

标记项

对于上面链接提到的题目，就是有效位 + tag，因为不考虑一直维护（脏位/修改位）跟替换算法位（替换控制位）。

有效位 1 位，一般脏位 1 位，一般替换控制位 1 位（LRU+组相联的话，貌似是 log2{路数} 位）

### 题目一

**主存地址 32 位，按字节编址，主存与Cache之间全相联映射，主存块大小为 1 个字，每字 32 位，采用 write-back 和随机替换策略，则能存放 32 K 字数据的 Cache 总容量至少应该有多少位 ？**

首先先求 Cache 行，主存块大小 1 个字，Cache 要放 32K 字的数据，所以一共就是 32K 行。

对于标记项，有效位就是 1bit，修改位 1bit，替换控制位不考虑了，因为采用随机替换策略。主存 32 位，块大小 32bit = 4B，块内地址 2 位，所以 Tag 有 30 位。

总容量至少就是 32K \* (1 + 1 + 30 + 32) = 32 \* 54K = 2048K。

**【答】：2048K**

### 题目二

**若计算机主存地址为 32 位，按字节编址，Cache 数据区大小为32 KB，主存块大小为 32B，采用直接映射方法和写回策略（Write Back），则 cache 行的位数至少是 ？**

先求 Tag 吧，Cache 一共 1K 行，直接映射，10 位组号，5 位块内地址，17 位 Tag。

32B = 256bit 是数据位，加上 17 位已经是 273 位了。

这题是一道选择题，最大才 275，采取写回，1 位修改位，还有一个有效位。

就算不考虑替换控制位了吧，正好 275 位。

**【答】 ：275 位**

### 题目三

**主存地址 32 位，按字节编址，某 Cache 数据库容量为 32KB，主存块大小为 64B，采用 8 路组相联映射，求 Cache 中的比较器个数和位数 ？**

题目愈来愈无聊，总是换着问法出。

*比较器是什么 ？*

比较器就是在组相联的Cache中，用来并行地比较分组中所有Cache行的 Tag 标记位与欲访问物理地址的 Tag 标记位，8 路组相联，比较器 8 个。

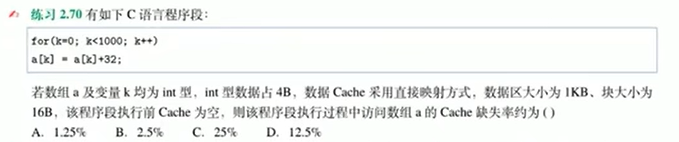
求标记位，32KB/64B = 2^9，2^9 / 2^3 = 6，6 位块内，剩 20 位 Tag。

因此，8比较器，选 20 位的。

**【答】 ：8, 20**

## 题目类型二 Cache 缺失率的计算

### 题目一

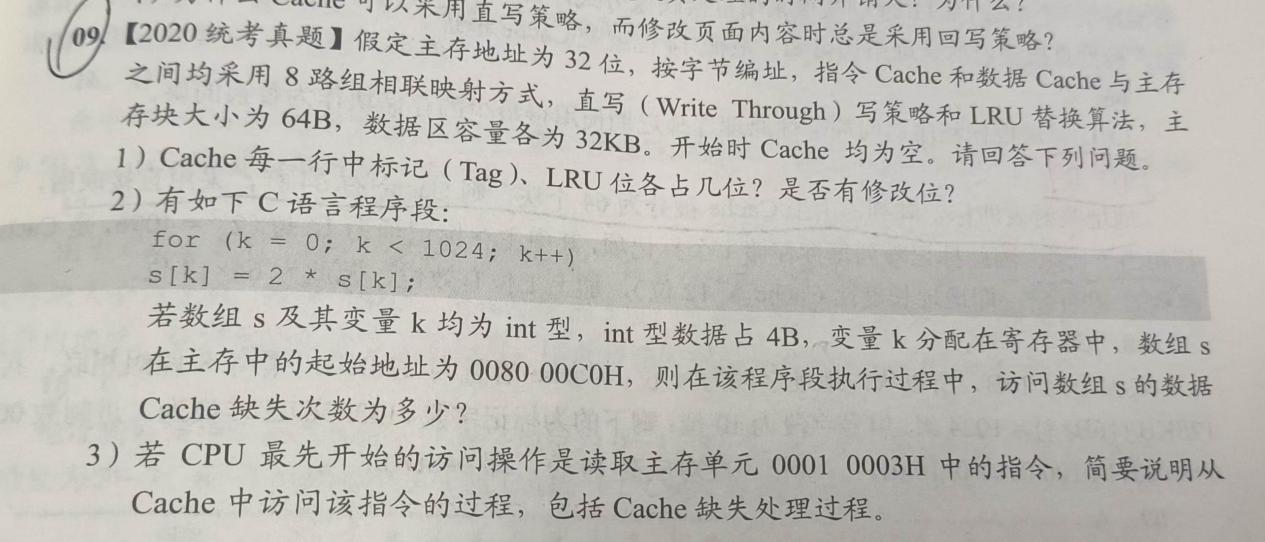


注意 Cache 一行是 16 B 大小，可以放 4 个int类型的数据。

k = 0 的时候，执行 a[0] = a[0] + 32，第一次访问 a[0]，Cache缺失，注意 cache 跟主存交换数据以块作为单位，将 { a[0] a[1] a[2] a[3] } 加载到 Cache 中，然后在访问 a[0]，不缺失了。后边两次访问 a[1] a[2] a[3]，均不缺失，也就是每 8 次访问只有第 1 次访问发生了缺失。就是 1 / 8 的缺失率。

**【答】 ：D**

### 题目二



1. 主存 32 位，6 位 offset ，Cache 好像是 32 KB，32KB/64B = 512 = 2^9 行，8路组相联，2^9 / 8 = 2^6，组号 6 位，20 位 Tag。

不懂 LRU 位问的什么，背的是 log\_2 {路数} = 3

write through，每次直接写回，不需要加修改位。

tips : 问题一跟 Cache 缺失率没啥关系

1. 块内偏移量 6 位，对于 0080 00C0H 的 C0 = 1100 0000 来说，后 6 位是 0。也就是起始地址是 0，装入某一行 Cache的话，从开头装入。（一般起始地址都给 0，不然很难做，我猜的）。一行 64B 能放 16 个数组元素，执行 s[k] = s[k] \* 2，访问 32 次，缺失 1 次。

也就是说，缺失率是 1 / 32。

一共访问了 2048 次，缺失次数 = 2048 / 32 = 2^11 / 2^5 = 2^6 = 64 次。

（3）0001 0003H = 0000 0000 0000 0001 0000 0000 0000 0011B

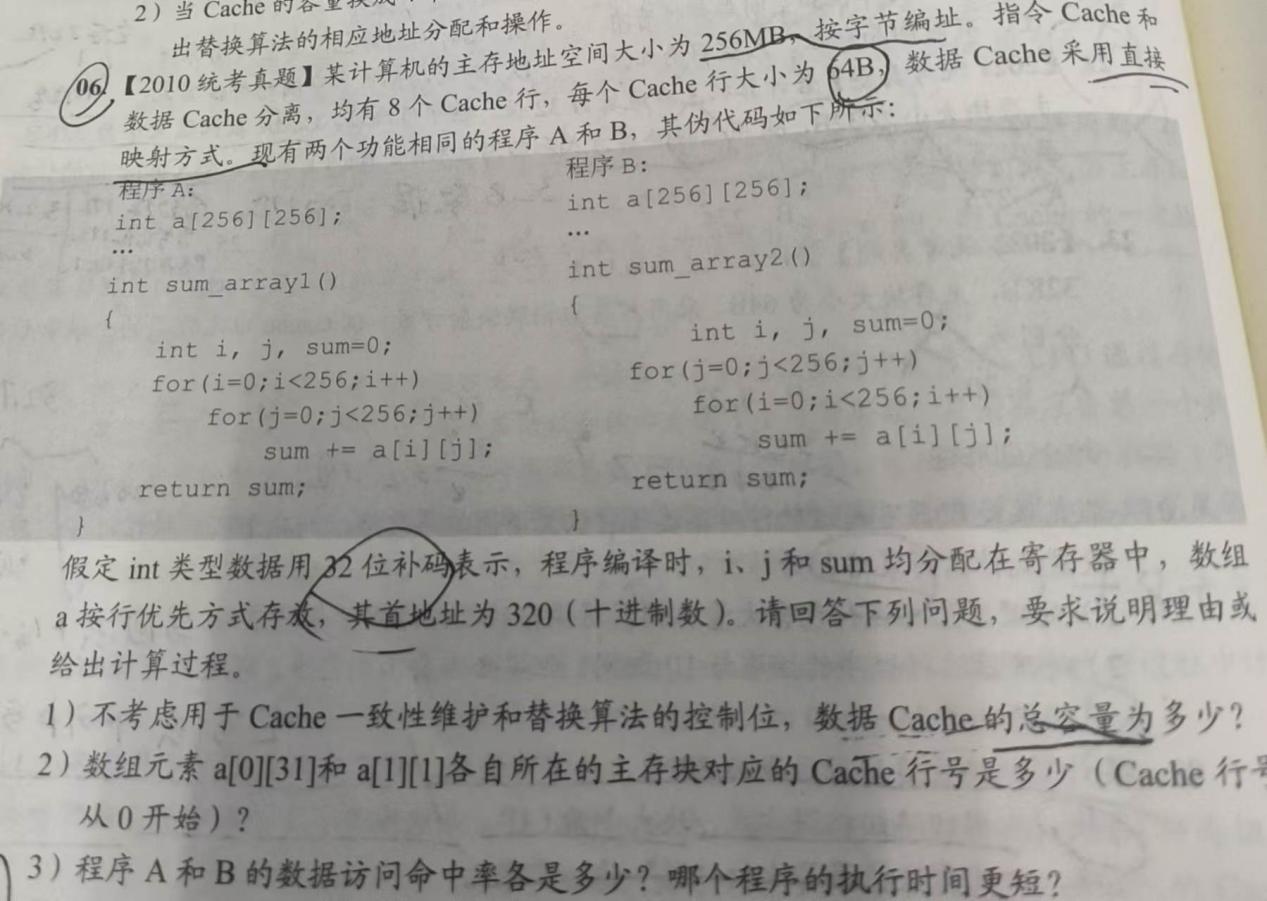
offfset 后 6 位，前 20 位 Tag。

组号是 0,（中间那6位），所以要去 Cache 的第 0 组。

题目说，开始时 Cache 是空的，所有 Cache 的有效位都是 0，所以 Cache 访问缺失。

此时，处理 Cache 缺失，用 LRU 从主存中换出一行（因为是初始状态，LRU没啥用，随便换一行）。然后加入标记字段，也就是 Tag（高20位），设置有效位为 1，修改 LRU 位，最后根据最后 6 位的块内地址读取相应内容（最终目的还是读取）。

### 题目三



1. ~~532 B。前面的理解肯定会这一小问。~~

~~32位主存地址 = Tag 19 + index 7 + offset 6。~~

~~标记位 1 位，数据 64B = 512 位。~~

~~一行532 位，8 行，乘一下，532B。~~

我都不知道自己之前怎么算对答案的，完全是晕的。

首先，地址是 28位而不是32 位，因为一共有 256M 个主存地址。

Cache 行 64 B，OFFSET = 6b ；8 个 Cache行 + 直接映射，INDEX 位 3 个 ；剩下 19 个作为 Tag。

考虑修改位 + 数据位一共 1 + 64 \* 8 位，修改位 1 + Tag 19 实际就是 2.5B.

8 \* (2.5 + 64) = 20 + 32 + 480 = 532B。

1. ~~一行放 16 个元素。~~

注意题目给了一个条件，a 数组按行优先方式存放，首地址是 320，也就是说首地址 a[0][0] 的起始地址是 320。

对于 a[0][31] , 相比 a[0][0]，偏移了 31 个 4 B，按照字节编址，1B一个地址。320 + 31 \* 4 = 444，得到了地址是多少，转化为二进制 444 = 256 + 128 + 32 + 16 + 8 + 4 = 0000 0001 1011 1100。（我这里做了二进制转换，用地址结构来求。也可以 address / 64 % 8 来求，感觉原理很奇怪，所以我用了这个方法）。注意 这个颜色 是主存块号，也就是 6，6 % 8 = 6 是对应的 Cache 行号。

对于 a[1][1] = 320 + (256 + 1) \* 4 = 1348 = 0000 0101 0100 0100，发现主存块号是 21，21 % 8 = 5，Cache行号是 5.

1. Cache 一行 64 B 可以放 16 个元素。

别忘了 a 有首地址，不过肯定 offset = 0 ，算一下 320 = 0001 0100 0000。

每 16 个元素，只有第一次访问才会缺失，命中率 15 / 16 = 93.75 %。

如果按照 j i 循环访问，我们有 8 个 Cache 行，先看访问 a[0][0] 到 a[7][0] :

Cache0 : a[0][0] ~ a[0][15]

Cache1 : a[1][0] ~ a[1][15]

....

Cache7 : a[7][0] ~ a[7][15]

接下来访问 a[8][0] 还是缺失，一直到访问完 a[255][0]。

这是再访问 a[0][1]，Cache里面没装 a[0] 这一行的内容，如此循环。

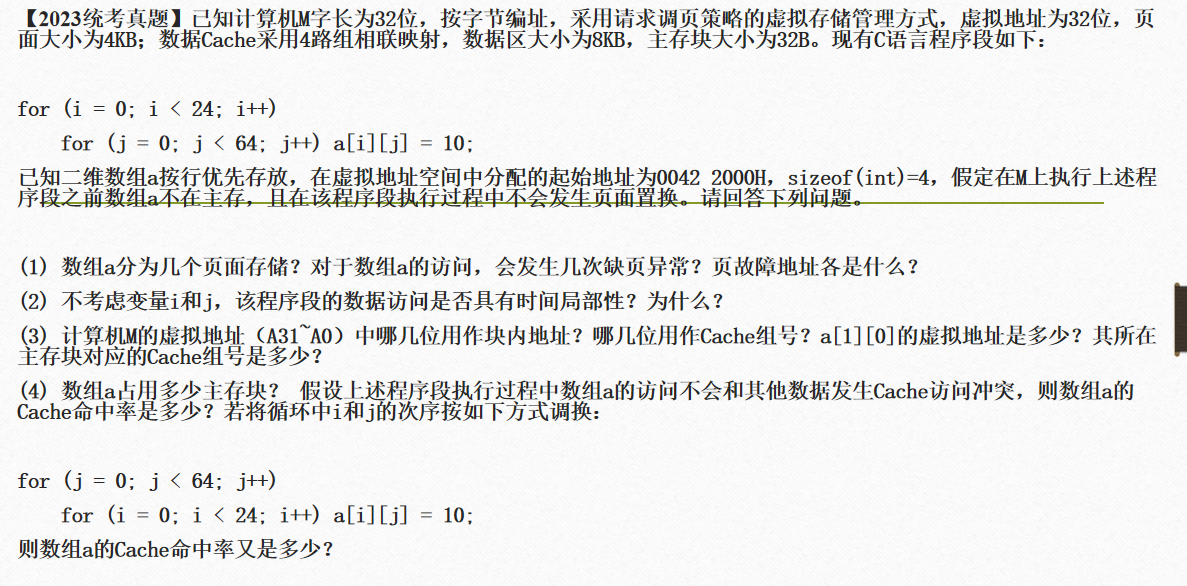
发现 Cache 一次都命中不了。

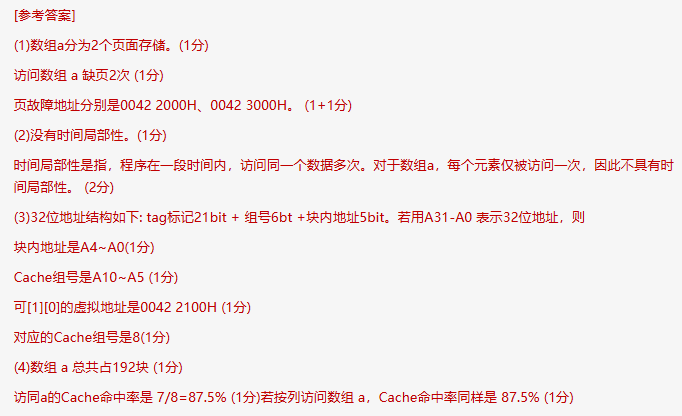
所以 B 的命中率是 0。

循环次数一样，一个命中率 93.75 %，一个是 0，所以 A 更快。

### 题目四

[2023年考研408计算机统考真题在线评测（附答案） - C语言网](https://www.dotcpp.com/exam/1060/)





1. 先分析虚拟地址结构，32位 = 20位虚拟页号 + 12位页内地址。

对于 a 来说，起始地址 0042 2000H，低 12 位 000H，说明 a 位于页首。

a 是 24 \* 64 \* 4B = 6KB，一页 4KB，两页就够了。

因为 a 不在主存，访问首地址 0042 2000H会缺页一次。访问完 4KB 之后还会缺页一次，就是过了 1000H 发生第二次缺页，地址位于 0042 3000H。

1. 没有时间局部性。并没有一段时间内被多次访问的数据。
2. 看答案吧
3. 按行 ：

32B放8个，只有第一次缺失，7/8

按列 ：

按照如下顺序 ：

a[0][0]

a[1][0]

...

a[23][0]

全部未命中，Cache64行，装入情况如下 ：

a[0][0] ~ a[0][7]

a[1][0] ~ a[1][7]

...

a[23][0] ~ a[23][7]

然后访问，a[0][1], a[1][1], ..., a[23][1], 全部命中 ...

同理，一直到 a[0][7], a[1][7], ..., a[23][7] 也全部命中 ....

到 a[0][8] 开始缺失，归纳所有情况，每 8 个元素只有一次缺失，结果还是 87.5%