

**实验六**

**cache模拟器实验**

**实验报告**

学科：计算机体系结构

学期：2024-2025 第一学期

编制日期：2024 年 12 月 27 日

编制人：江家玮

学号：22281188

班级：计科2204

**目录**

[一、实验目的 1](#_Toc20679)

[二、实验内容 1](#_Toc24245)

[实验内容一：熟悉模拟程序 1](#_Toc4814)

[实验内容二：利用该模拟程序仿真课后习题4.1，并得出结果。 1](#_Toc15117)

[三、实验内容一 2](#_Toc6687)

[3.1 实验内容 2](#_Toc17705)

[3.2 实验过程 2](#_Toc15420)

[3.3 实验结果 3](#_Toc15264)

[四、实验内容二 5](#_Toc26641)

[五、心得体会 11](#_Toc6548)

# 一、实验目的

本次实验的主要目的是熟悉cache的原理。加深对cache的映像规则、替换方法、cache命中与缺失的理解。通过实验对比分析映像规则对cache性能的影响。

# **二、实验内容**

## 实验内容一：熟悉模拟程序

阅读给出的cache模拟程序（cachesimulator.cpp），理解其中的主要参数与功能。修改代码，随机生成N个访存地址，运行程序观察并分析结果（例如，可分析其中命中次数，不命中次数，替换次数）。熟悉cache系统的执行过程（可举例详细分析一次命中/不命中/替换过程）。

## 实验内容二：利用该模拟程序仿真课后习题4.1，并得出结果。

习题4.1：The following C program is run (with no optimizations) on a machine with a cache that has four-word (16-byte) blocks and holds 256 bytes of data:

      inti, j, c, stride, array[256];

      for (i=0;i<10000;i++)

             for (j=0;j<256;j=j+stride)

                    c = array[j]+5;

If we consider only the cache activity generated by references to the array and we assume that integers are words.

1) What is the expected miss rate when the cache is direct-mapped and stride=132? How about if stride=131?

2) Would either of these changes if the cache were two-way set associative?

（理论分析该试题，得出上述两问的结果。利用cache模拟程序仿真该程序的cache执行过程，核实实验结果是否与理论分析计算结果一致。）

3) 对于每个miss，标注是哪种miss（compulsory，conflict，capacity）。

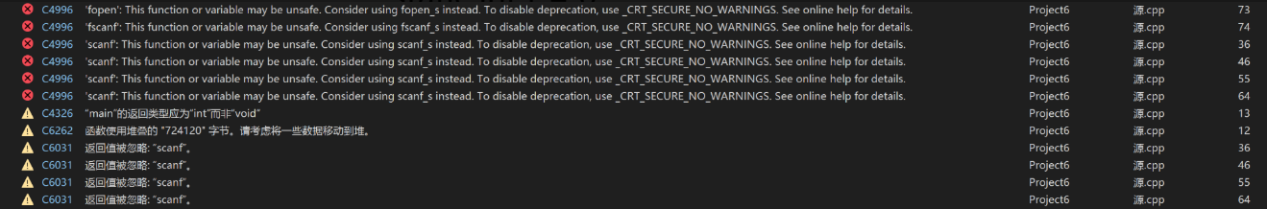
# **三、实验内容一**

## 3.1 实验内容

阅读给出的cache模拟程序（cachesimulator.cpp），理解其中的主要参数与功能。修改代码，随机生成N个访存地址，运行程序观察并分析结果（例如，可分析其中命中次数，不命中次数，替换次数）。熟悉cache系统的执行过程（可举例详细分析一次命中/不命中/替换过程）。

## 3.2 实验过程

下载模拟程序并在vs中打开、进行编译发现存在以下问题：



fopen, fscanf, scanf 的不安全警告 (C4996)：Visual Studio 对传统的 C 库函数（如 fopen, scanf 等）给出了安全性警告。带 \_s 后缀的函数（例如 fopen\_s 和 scanf\_s）。

函数返回值被忽略 (C6031)：某些函数（如 scanf 和 fscanf）的返回值未被处理，导致可能无法检测输入或文件读取错误。

main 的返回类型 (C4326)：标准 C 的 main 函数应返回 int 类型，而不是 void。

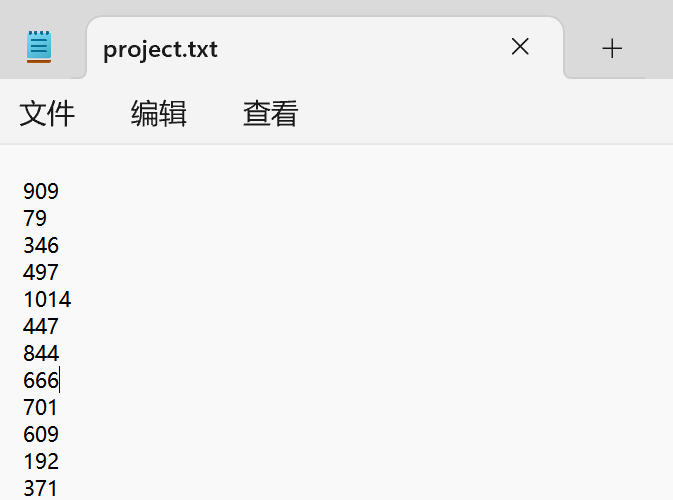
堆栈内存警告 (C6262)：代码中定义了较大的局部变量数组（如 bytearray[500] 和 newarray[300][300]），可能会导致栈内存不足。应考虑将它们动态分配到堆中。

这里需要我们先自己创建一个project.txt文件，文件的内容实际上表示访存请求的地址序列。具体来说，这个文件的内容应该包含一系列的整数，每一行代表一个字节地址，这些地址是程序运行时模拟访存操作的输入。

1. int\* bytearray = (int\*)malloc(5000 \* sizeof(int)); *// 使用动态分配以避免栈溢出*
2. int\* wordaddress = (int\*)malloc(5000 \* sizeof(int));
3. int\*\* newarray = (int\*\*)malloc(300 \* sizeof(int\*));
4. int\*\* lru = (int\*\*)malloc(300 \* sizeof(int\*));
5. for (int k = 0; k < 300; k++) {
6. newarray[k] = (int\*)calloc(300, sizeof(int));
7. lru[k] = (int\*)calloc(300, sizeof(int));
8. }
9. char ans = 'y';
10. int c1c = 0, c2c = 0, c3c = 0;
11. float missrate = 0, hitrate = 0;
12. *// 生成随机访存地址文件*
13. FILE\* gen\_fp;
14. if (fopen\_s(&gen\_fp, "C:/Users/86186/Desktop/project.txt", "w") != 0) {
15. printf("Error: Unable to create project.txt file.\n");
16. return -1;
17. }
18. srand((unsigned int)time(NULL));
19. for (int i = 0; i < N; i++) {
20. int address = rand() % 1024;
21. fprintf(gen\_fp, "%d\n", address);
22. }
23. fclose(gen\_fp);
24. printf("Random memory access addresses have been generated in project.txt\n");

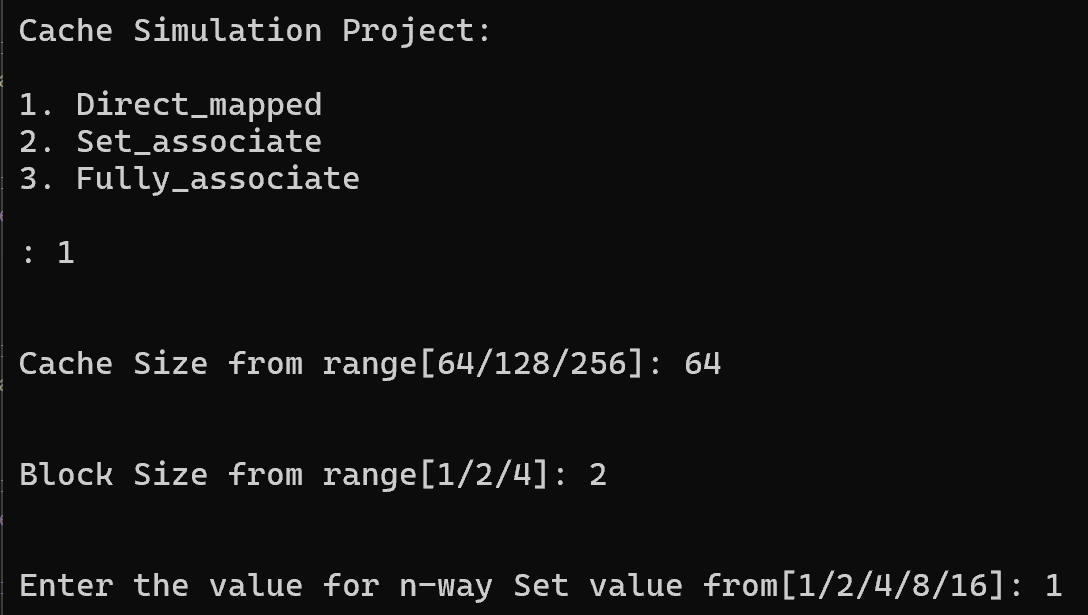
我用这段代码生成一份容量为500的随机访存地址文件，它们的数值范围在0-1023。（这里的容量和范围均可以改动）

Project.txt文件行数太多，这里只展示其中的一小部分：

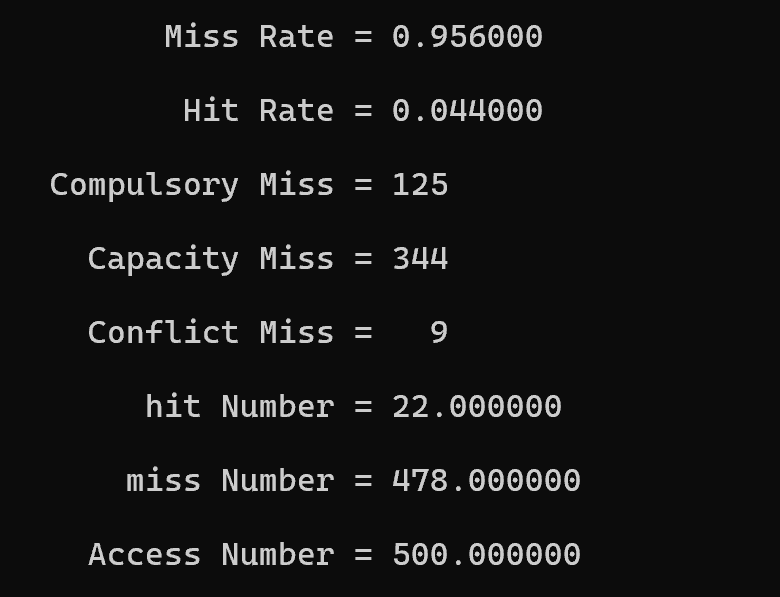


## 3.3 实验预期和结果

运行时我设置了直接映像的方式、cache大小为64，每块大小为2个字（直接映像的路数为1）：



运行程序之后的结果如下图所示：



分析运行结果：

Miss Rate（不命中率）= 0.956000：访问缓存时有95.6%的访存操作没有命中；Hit Rate（命中率）= 0.04400：表示4.4%的访存操作命中缓存。

Compulsory Miss（强制不命中）= 125：共有125次强制不命中，强制不命中是指访问某块数据时，这块数据第一次加载到缓存中。

Capacity Miss（容量不命中）= 344：表示有344次容量不命中。容量不命中发生在缓存已满，且新数据需要替换已有数据的情况下。多次容量不命中说明当前缓存容量不足以容纳所有访问的块。

Conflict Miss（冲突不命中）= 9：表示发生9次冲突不命中。冲突不命中一般与缓存的映射方式有关（如直接映射）。

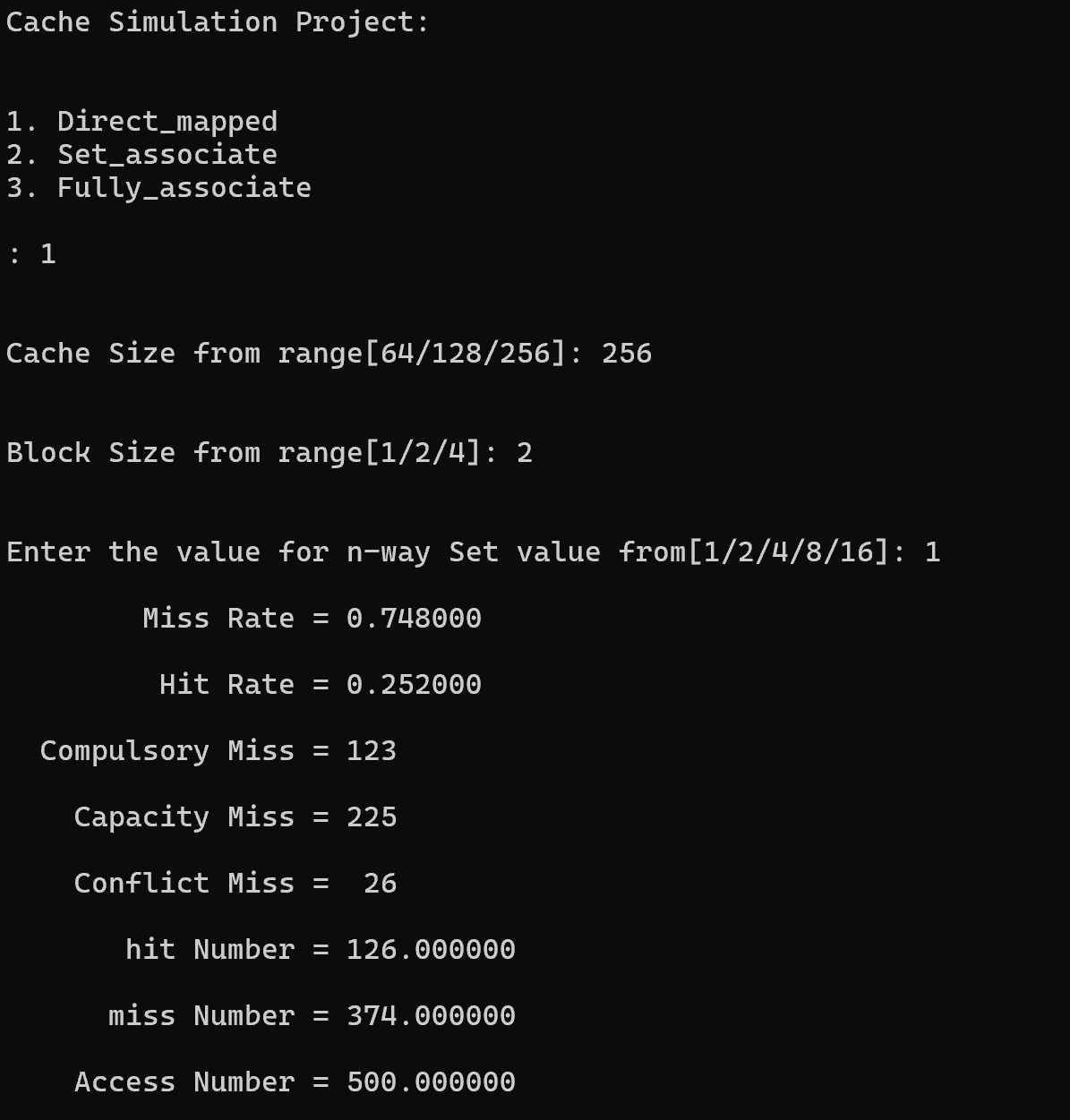
hit Number（命中次数）= 22：共有22次访存操作命中了缓存。

miss Number（不命中次数）= 478：共有478次访存操作未命中缓存。

Access Number（访存次数）= 500：总共进行了500次访存操作（hit Number + miss Number =22+478=500）。

分析高不命中率的原因：主要原因是强制不命中和容量不命中。这表明每个数据块第一次被访问时，缓存中没有相应的数据；并且容量不命中是最主要的原因可以推断当前缓存大小太小了；冲突不命中只有少数几次，可以判断访存地址分布基本合理。

后续调整cache大小为256，发现缺失率显著降低：主要是容量不命中降低。



# 四、实验内容二

习题4.1：The following C program is run (with no optimizations) on a machine with a cache that has **four-word (16-byte) blocks** and holds **256 bytes**of data:

      inti, j, c, stride, array[256];

      for (i=0;i<10000;i++)

             for (j=0;j<256;j=j+stride)

                    c = array[j]+5;

If we consider only the cache activity generated by references to the **array** and we assume that integers are words.

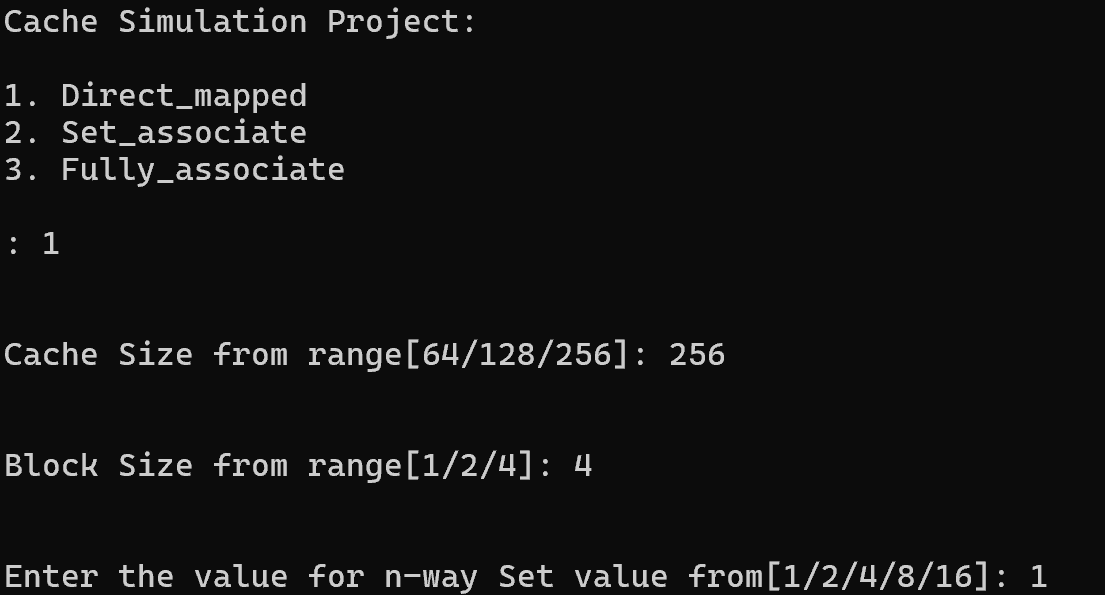
1) What is the expected miss rate when the cache is direct-mapped and stride=132? How about if stride=131?

由于stride为132，所以每次的内部循环我们都要访问的数据为arrar[0]和array[132]，又由于我们采用的是直接映射的方式并且每一个整数的大小为一个字长，所以array[0]所对应的内存块为0/4=0，而cache中的块的数量为16，0%16=0，所以array[0]对应的cache中的缓冲行的为0，同理可得array[132]对应的条目存放的缓冲行为1。因此可以预测，一共会产生两次未命中的情况，是在第一次循环时访问array[0]和array[132]时。

缺失率为2/(2\*10000)=0.01%。

在第一次执行j循环的时候，cache为空，发生了j=0和j=132两次强制失效，此时cache中会存有array数组中0,1,2,3；132,133,134,135这两组八个数据。后续的话都是命中的。因为这两个块之间没有竞争关系，所以后续循环里面是不会发生冲突的，都会命中，只有一开始的两个是强制失效。

使用软件的执行结果来验证预测：



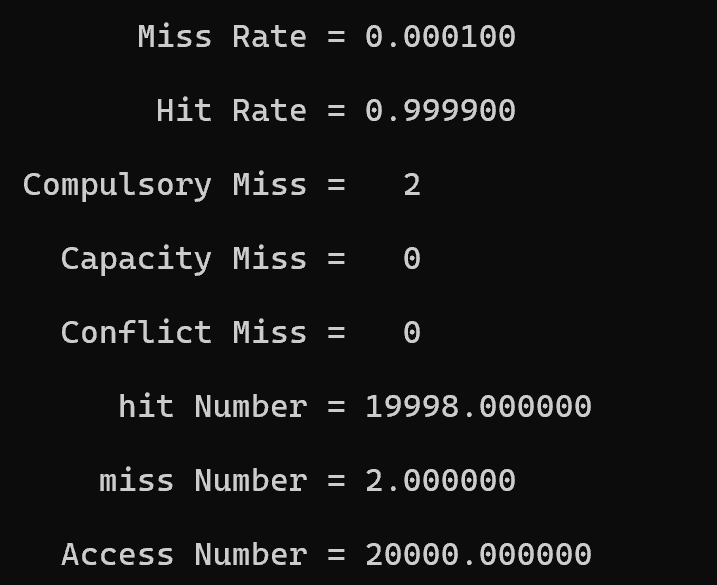
在执行程序的时候发现问题：存储访存块号，块地址等数据的数组容量为500，无法实现上述程序段中访问20000次的需求，所以对程序中数组的大小进行了扩充：

1. int bytearray[50000], wordaddress[50000];

接下来，将循环中的访存信息存进对应的数组（因为数组的单位是字，所以这里要\*4转换成对应的字节地址）

1. int stride = 132;
2. for (int k = 0; k < 10000; k++)
3. for (j = 0; j < 256; j = j + stride) {
4. bytearray[i] = j \* 4;
5. i++;
6. }

最后查看程序运行的结果发现和预期结果一致：

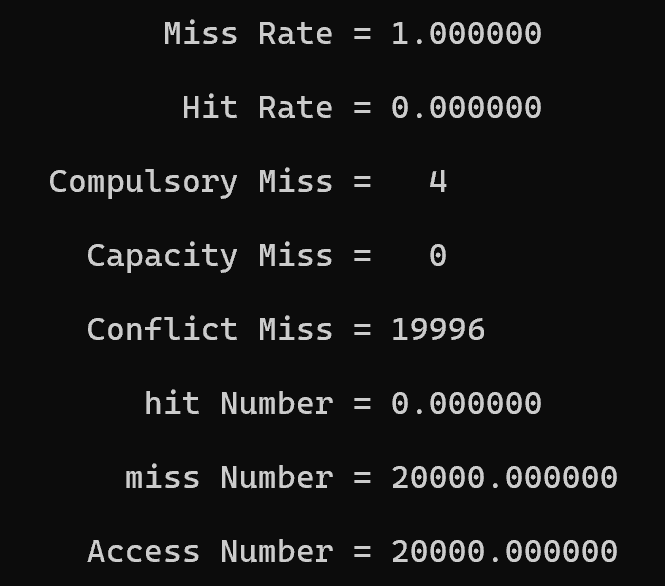


当stride=131时同理我们要访问的数组元素为array[0]和array[131],可以计算出这两个元素所对应的缓冲行都为第0个，所以每次访问的时候都会发生缺失现象，缺失率为100%。所以这个时候，进行循环，除开第一次属于强制失效之外，后续也会因为不断的在两个块的内容之间反复替换，所以在整个循环的过程中会一直处于miss的状态。缺失率为100%。

同样执行程序，查看预测结果的正确性：

修改stride=131。

1. int stride = 131;
2. for (int k = 0; k < 10000; k++)
3. for (j = 0; j < 256; j = j + stride) {
4. bytearray[i] = j \* 4;
5. i++;
6. }

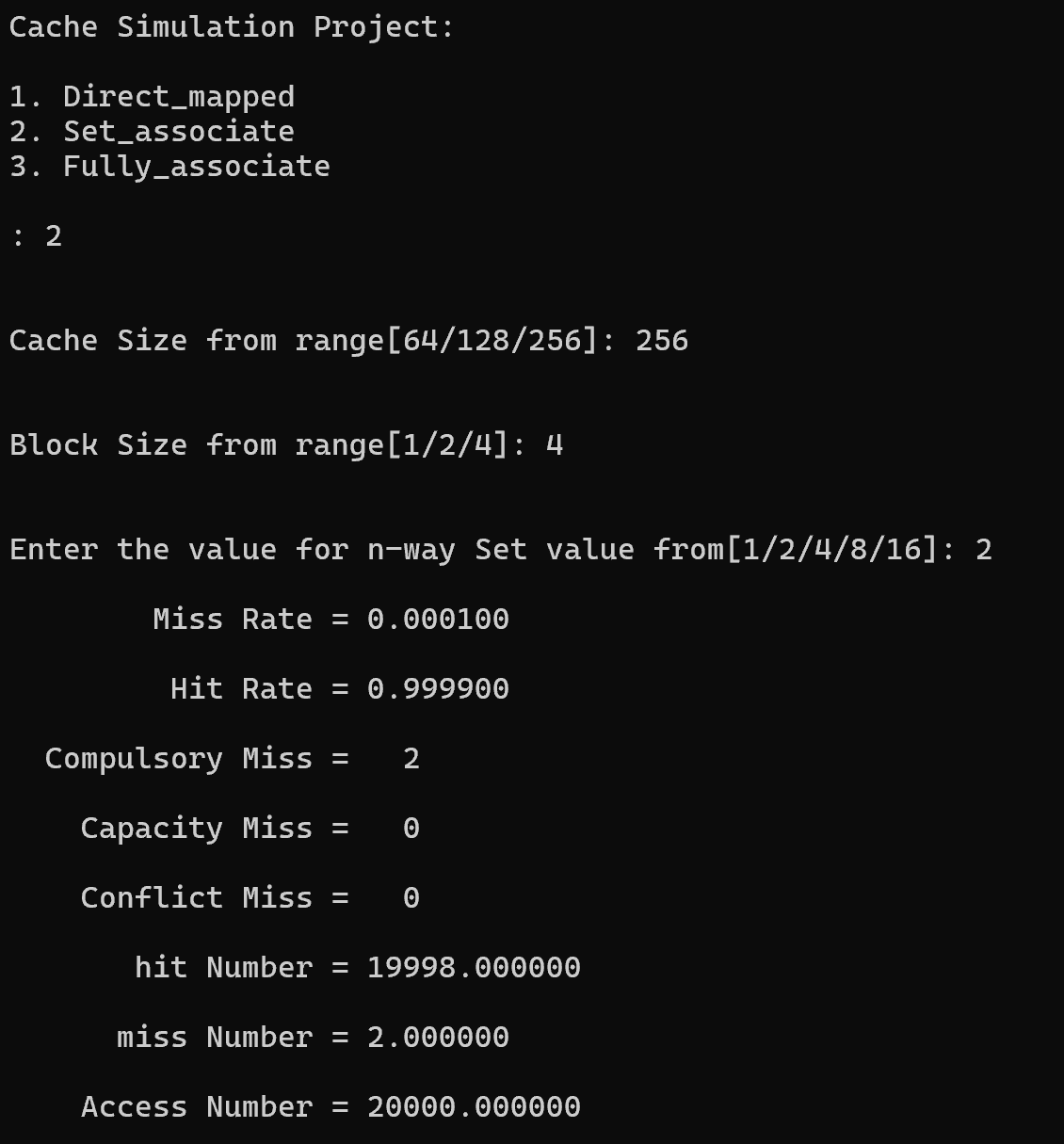


Miss率为100%，和我们的预期结果相一致。

2) Would either of these changes if the cache were two-way set associative?（理论分析该试题，得出上述两问的结果。利用cache模拟程序仿真该程序的cache执行过程，核实实验结果是否与理论分析计算结果一致。）

使用组相联映射不会改变stride=132的结果，也是两次强制失效，因为在j循环里面仅两次执行，虽然访问的0和132对应的块是一组的，但是因为一组有两块，彼此之间也不会有什么影响和冲突。不会影响最后的结果。

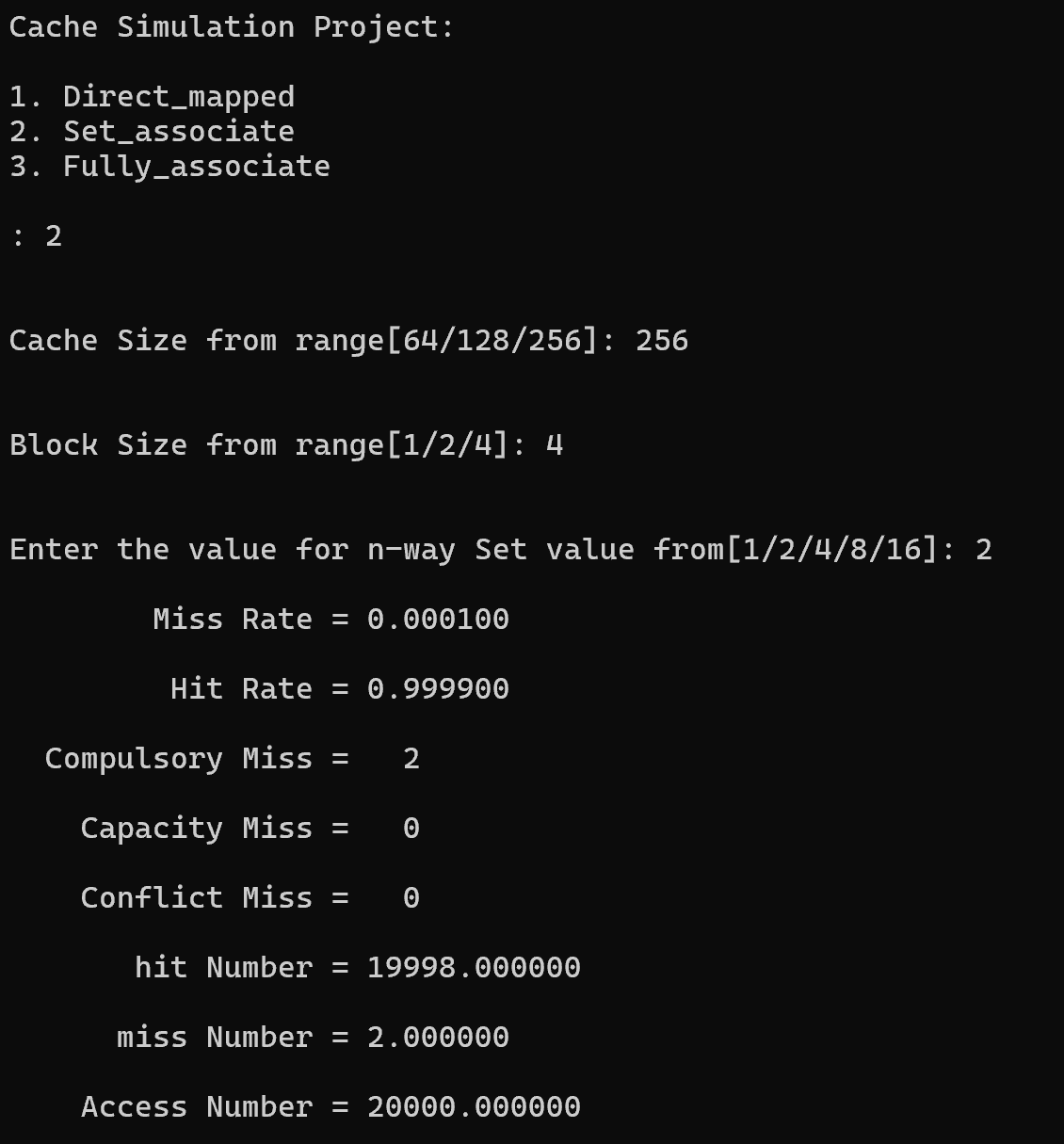
执行程序查看结果：



和预期结果一致。

但是使用组相联映射会改变stride=131的结果，因为在上一阶段中，发生100% 的miss率的主要原因还是在于两个块之间存在竞争关系，他们在cache中只有一个对应块，这就导致后面每次访问miss之后，都要将cache中的块替换掉。而使用了组相联映射之后，两个快对应一个组，而一组两块，就很好的解决了这个问题，除开两次强制失效之外，都会命中。所以对应的命中率的预测为：2/20000=0.0001

执行程序查看结果：



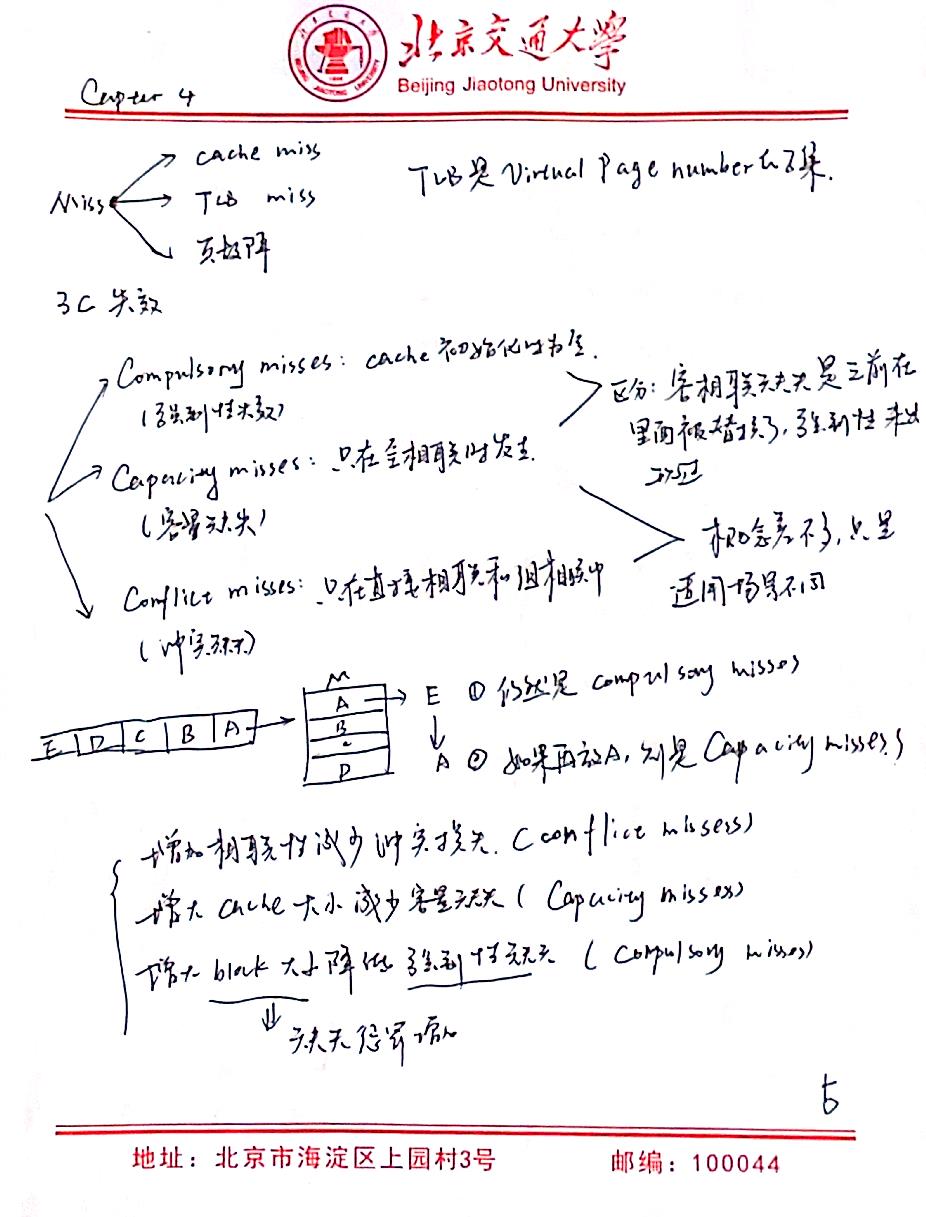
和预期计算结果相一致。

3) 对于每个miss，标注是哪种miss（compulsory，conflict，capacity）。

在第一问的直接映像情况下，当stride为132时，总共有两次miss，且都是强制缺失，因为这是对相应地址的第一次访问，缓存中必定找不到对应的内容。在stride为131时，总共有20000次miss，其中前两次和上面情况一样，都是强制缺失，而后面的19998次则是冲突失效。这是因为两个缓存块竞争同一个缓存位置，导致彼此替换，从而发生了冲突失效。我最开始有一个疑问，认为对于j=0和stride=131的情况，应该对应同一个缓存块，且缓存为空时应该只会发生一次强制缺失。但在查阅资料后，我明白了强制缺失指的是对某个地址的第一次访问，因此在这种情况下，应该发生两次强制缺失。

在第二问的组相联缓存中，不论stride是132还是131，情况都是相同的，只有两次miss，且这两次都属于强制缺失。这是因为组相联缓存允许多个地址映射到同一个缓存组，从而减少了由于缓存位置竞争而导致的冲突失效，因此只有第一次访问时才会发生强制缺失。

经过总结，将3C失效的异同整理为笔记：



**五、心得体会**

在完成这次实验之后，我对缓存（cache）系统有了更深入的理解，特别是缓存的映射规则、替换策略以及命中率和缺失率的计算方法。通过模拟不同的缓存配置（如直接映射和两路组相联缓存），我能够观察到不同步长（stride）对缓存性能的影响。

在实验中，我首先通过修改给定的程序，生成了N个随机访存地址，并通过模拟程序分析了命中次数、缺失次数以及替换次数。通过这些数据，我可以直观地了解在不同配置下缓存的工作原理，尤其是在缓存容量和缓存块大小的限制下，如何影响程序的执行效率。

对于实验内容二中的问题，我通过理论分析和模拟程序仿真得出了相关结论。例如，当stride为132时，由于访问模式和缓存映射方式的冲突，导致较高的缺失率；而stride为131时，由于访问模式和缓存映射方式的适配，命中率略有提高。此外，通过标注每次缺失的类型（如强制失效、冲突失效、容量失效），我对缓存的失效机制有了更清晰的认识。

总体来说，这次实验不仅帮助我加深了对缓存原理的理解，也提高了我对缓存性能优化的思考能力。在实际应用中，合理的缓存配置和访问模式优化可以显著提升程序的性能。这对我今后在计算机系统和硬件优化方面的学习和研究都有很大的帮助。