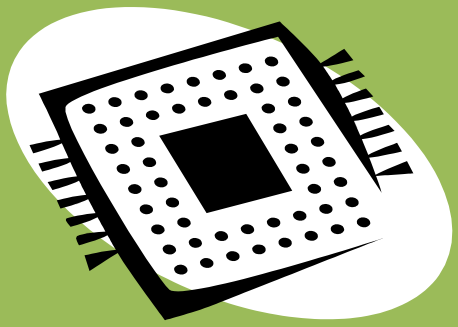
华中科技大学

2021

计算机系统结构 ·实验报告·

专　　业：	计算机科学与技术
班　　级：	计 1807
学　　号：	U201814682
姓　　名：	李世铭
电　　话：	13125077740
邮　　件：	1074822720@qq.com
完成日期：	2021.5.10



计算机科学与技术学院

目 录

1　缓存实验

1.1 实验概述

- 实验目的

  理解 cache 工作原理；

  加深 Cache 缓存组成结构对 C 程序性能的影响的理解

- 实验内容

  编写一个 200-300 行的 C 程序来模拟 Cache 缓存的行为。已给出模拟 cache 的基本框架，在此基础上完善代码，实现具体功能。

  - 任务：在 csim.c 提供的程序框架中，编写实现一个 Cache 模拟器：
    - 输入：内存访问轨迹
    - 操作：模拟缓存相对内存访问轨迹的命中/缺失行为
    - 输出：命中、缺失和（缓存行）淘汰/驱逐的总数
  - 具体要求：完成的 csim.c 文件应能接受与参考缓存模拟器 csim-ref 相同的命令行参数并产生一致的输出结果。
  - 命令行格式：csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>
    - -h: 显示帮助信息（可选）
    - -v: 显示轨迹信息（可选）
    - -s <s>: 组索引位数
    - -E <E>: 关联度（每组包含的缓存行数）
    - -b <b>: 内存块内地址位数

    -t <tracefile>: 内存访问轨迹文件名
  - csim.c 编程要求：
    - 模拟器必须在输入参数 s、E、b 设置为任意值时均能正确工作——即需要使用 malloc 函数（而不是代码中固定大小的值）来为模拟器中数据结构分配存储空间。
    - 由于实验仅关心数据 Cache 的性能，因此模拟器应忽略所有指令

cache 访问（即轨迹中“I”起始的行）

■ 假设内存访问的地址总是正确对齐的，即一次内存访问从不跨越块的边界——因此可忽略访问轨迹中给出的访问请求大小

■ main 函数最后必须调用 printSummary 函数输出结果，并如下传之以命中 hit、缺失 miss 和淘汰/驱逐 eviction 的总数作为参数：

printSummary(hit_count, miss_count, eviction_count);

1.2 总体设计

设计 csim.c 程序来实现对 cache 的模拟，程序可分为如下几个部分：

1.cache 结构体的声明

cache 结构体的定义如图 1.1 所示。

```
struct cache
{
  /* data */
  unsigned long s,E,b,S;
  unsigned long **cache;
}Cache;
```

图 1.1 cache 结构体的定义

结构体表示整个 cache 存储器，其中 s 表示组索引位数，也就是组号的数据位宽，E 表示关联度，也就是每组的 cache 块行数，b 表示内存块内地址位数，S 表示该 cache 存储器总共有几组。用一个 2 维数组 cache 来表示 cache 存储器的存储区，如图 1.2 所示。

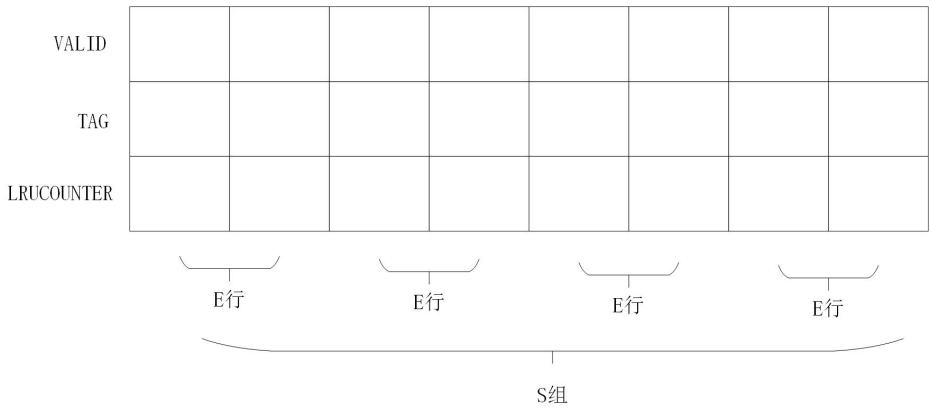


图 1.2 cache 数组结构

cache 数组的第一维大小为 S*E，每一个元素表示一个 cache 块，每一个元素都是一个大小为 3 的无符号整型数组，分别记录该 cache 块是否有效，该 cache 块存储的数据的 TAG 以及该 cache 块存储的数据的 LRU 计数。

2.常量及全局变量定义

定义程序中用到的常量及全局变量，方便程序处理，如图 1.3 所示。

```
#define VALID 0
#define TAG 1
#define LRUCOUNTER 2

#define HIT 0
#define MISS 1
#define EVICTION 2

FILE *file;
int Hits=0,Misses=0,Evictions=0,helpflag=0;
_Bool verbose=0;
```

图 1.3 常量及全局变量定义

VALID、TAG 和 LRUCOUNTER 是每个 cache 块记录该 cache 块是否有效，该 cache 块存储的数据的 TAG 以及该 cache 块存储的数据的 LRU 计数的数组下标，结构可以参考图 1.2。HIT、MISS 和 EVICTION 分别表示在模拟一次访问后返回的处理结果是命中、确实和淘汰。文件指针 file 用于指向输入的轨迹文件，Hits、Misses 以及 Evictions 表示命中、缺失和淘汰计数,helpflag 是是否打印帮助信息的标志，默认值为 0 不打印帮助信息，verbose 是是否打印轨迹信息的标志，默认值为 0 不显示轨迹信息。

3.输入处理

定义 inputhandle 函数用于处理输入的命令行字符串，如图 1.4 所示。

```
void inputhandle(int argc,char **argv)
{
  char opt;
  while ((opt=getopt(argc,argv,"s:E:b:t:vh"))!=-1)
  {
    switch (opt)
    {
    case 'h':
      print_help();
      helpflag=1;
      break;
    case 'v':
      verbose=1;
      break;
    case 's':
      Cache.s=atol(optarg);
      break;
    case 'E':
      Cache.E=atol(optarg);
      break;
    case 'b':
      Cache.b=atol(optarg);
      break;
    case 't':
      file=fopen(optarg,"r");
      break;

    default:
      break;
    }
  }
}
```

图 1.4 输入命令行处理函数

利用 linux 内置的 unistd 库中用来解析命令行选项参数的 getopt 函数处理输入的命令行。逐步读取选项参数，遇到 h 时，打印帮助信息并将 helpflag 置 1，遇到 v 时，将 verbose 置 1，遇到 s、E、b 时，将后面带的字符串转换为整型数值并保存为 cache 的 s、E、b，遇到 t 时，打开对应的文件并用 file 指向这个文件。

4.创建 cache 存储器

定义 creatCache 函数创建 Cache 存储器的数据存储区，如图 1.5 所示。

```
void creatCache()
{
  Cache.S=1<<Cache.s;
  Cache.cache=(unsigned long**)malloc(Cache.E*Cache.S*sizeof(unsigned long*));
  for(int i=0;i<Cache.E*Cache.S;i++)
  {
    Cache.cache[i]=(unsigned long*)malloc(3*sizeof(unsigned long));
    Cache.cache[i][VALID]=0;
    Cache.cache[i][TAG]=0;
    Cache.cache[i][LRUCOUNTER]=0;
  }
}
```

图 1.5 创建 Cache 存储器的存储区

首先根据组索引位数通过对 1 右移位运算得到组号个数，然后开辟 E*S 个无符号整型数指针的空间用来表示 E*S 个 cache 块，每个指针指向大小为 3 的无符号整型数组，VALID、TAG 和 LRUCOUNTER 都赋初值为 0。

5.释放 cache 占用的内存空间

定义 freeCache 函数清空 Cache 存储器的数据存储区所占内存，如图 1.6 所示。

```
void freeCache()
{
  for(int i=0;i<Cache.E*Cache.S;i++)
  {
    free(Cache.cache[i]);

  }
  free(Cache.cache);
}
```

图 1.6 释放 Cache 存储器占用内存

释放开辟的指针数组和无符号整型数组空间。

6.数组序号与组号、行号的转换

数组序号与组号、行号的转换函数的定义如图 1.7 所示。

```
unsigned long SEtoseq(unsigned long s,unsigned long E)
{
    return s*Cache.E+E;
}

unsigned long seqtoS(unsigned long seq)
{
    return seq / Cache.E;
}

unsigned long seqtoE(unsigned long seq)
{
    return seq % Cache.S;
}
```

图 1.7 数组序号与组号、行号的转换

由于 cache 块是由组号 E 和行号 S 两个值来确定，而 cache 的模拟存储器只使用了一维指针数组来表示，所以定义从组号和行号到数组序号转换的函数、从数组序号到组号转换的函数以及从数组序号到行号转换的函数，方便下一步操作。

7.cache 访问模拟

对 cache 的模拟访问函数的定义如图 1.8 所示。

```
int cacheaccess(unsigned long tag,unsigned long S)
{
  int res=-1;
  int emptyspace =  -1;
  int maxspace = -1;
  int maxlru=-1;
  for(int i=0;i<Cache.E;i++)
  {
    if(Cache.cache[SEtoseq(S,i)][VALID])
    {
      //块有效
      if(Cache.cache[SEtoseq(S,i)][TAG]==tag)
      {
        //命中
        Cache.cache[SEtoseq(S,i)][LRUCOUNTER]=0;
        res = HIT;
      }
      else
      {
        //该块未命中
        Cache.cache[SEtoseq(S,i)][LRUCOUNTER]++;
        if((signed)Cache.cache[SEtoseq(S,i)][LRUCOUNTER]>maxlru)
        {
          maxlru=Cache.cache[SEtoseq(S,i)][LRUCOUNTER];
          maxspace=i;
        }
      }
    }
    else
    {
      //块无效
      emptyspace = i;
    }
  }
  if(res==HIT) return res;
  else
  {
    if(emptyspace!=-1)
    {
      //有空闲块
      Cache.cache[SEtoseq(S,emptyspace)][VALID]=1;
      Cache.cache[SEtoseq(S,emptyspace)][TAG] =tag;
      Cache.cache[SEtoseq(S,emptyspace)][LRUCOUNTER]=0;
      res=MISS;
    }
    else
    {
      //无空闲块
      Cache.cache[SEtoseq(S,maxspace)][VALID]=1;
      Cache.cache[SEtoseq(S,maxspace)][TAG] =tag;
      Cache.cache[SEtoseq(S,maxspace)][LRUCOUNTER]=0;
      res=EVICTION;
    }
  }
  return res;
}
```

图 1.8 cache 访问模拟

输入的参数是要访问的组号和数据的 TAG，首先一次遍历该组所有 cache 块，如果块有效且命中，将 HIT 保存在 res 中，将 LRU 计数重置为 0 并继续遍历剩余的 cache 块；如果块有效但不命中，将该块的 LRU 计数加一并继续遍历剩余的 cache 块；如果块无效，保存该空块并继续遍历剩余的 cache 块。遍历完后，如果 res 为 HIT，返回 res，否则表示没有命中，如果有空块将该块置为有效，TAG 赋为数据的 TAG，LRU 计数初始化为 0，并且 res 赋为 MISS 并返回 res，如果没有空块，将 LRU 计数最大的块置为有效，TAG 赋为数据的 TAG，LRU 计数初始化为 0，并且 res 赋为 EVICTION 并返回 res。流程图如图 1.9 所示。

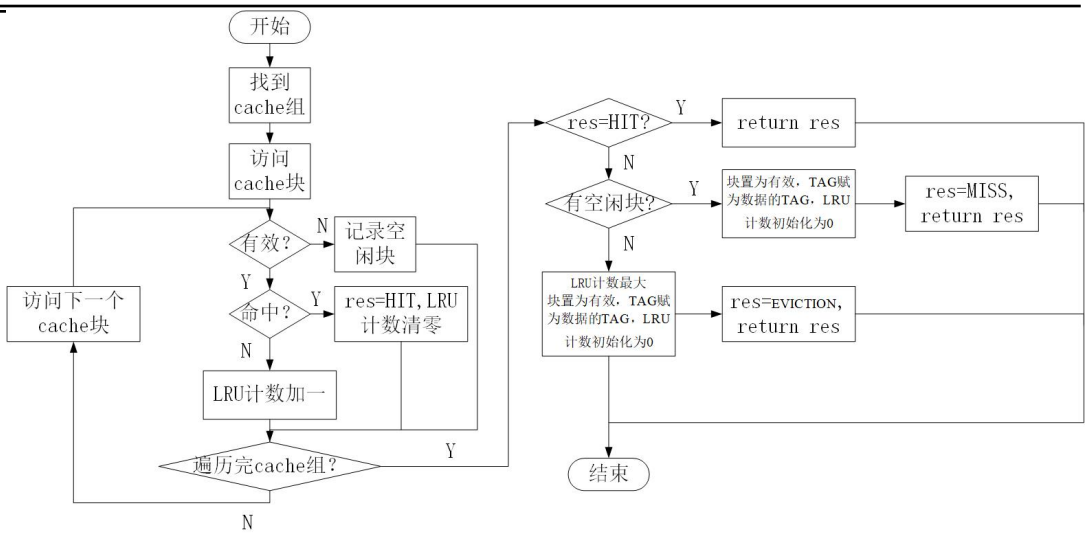


图 1.9 cache 模拟访问函数流程图

8.结果处理

对 cache 模拟访问函数的返回值处理函数定义如图 1.10 所示。

```
void reshandle(int res)
{
  switch (res)
  {
  case HIT:
    /* code */
    Hits++;
    if(verbose) printf(" hit");
    break;
  case MISS:
    Misses++;
    if(verbose) printf(" miss");
    break;
  case EVICTION:
    Misses++;
    Evictions++;
    if(verbose) printf(" miss eviction");
    break;
  default:
    break;
  }
}
```

图 1.10 结果处理函数

根据 res 的值进行选择，res 为 HIT 时，命中计数 Hits 加一，res 为 MISS 时，缺失计数 Misses 加一，res 为 EVICTION 时，缺失计数 Misses 和淘汰计数 Evictions 分别加一，另外如果 verbose 为 1，打印轨迹信息。

9.读取轨迹文件并处理访问语句

读取轨迹文件并处理访问语句的代码段如图 1.11 所示。

```
while (fscanf(file,"%s %lx,%lu",op,&addr,&size)!=EOF)
{
  /* code */
  tag=addr>>(Cache.s+Cache.b);
  S=(((1<<(Cache.s+Cache.b))-1)&addr)>>Cache.b;
  switch (op[0])
  {
  case 'M':
    /* code */
    if(verbose) printf("%c %lx,%lu",op[0],addr,size);
    res=cacheaccess(tag,S);
    reshandle(res);
    res=cacheaccess(tag,S);
    reshandle(res);
    if(verbose) printf("\n");
    break;
  case 'L':                          char op[3]
    if(verbose) printf("%c %lx,%lu",op[0],addr,size);
    res=cacheaccess(tag,S);
    reshandle(res);
    if(verbose) printf("\n");
    break;
  case 'S':
    if(verbose) printf("%c %lx,%lu",op[0],addr,size);
    res=cacheaccess(tag,S);
    reshandle(res);
    if(verbose) printf("\n");
    break;
  case 'I':
    if(verbose) printf("%c %lx,%lu",op[0],addr,size);
    if(verbose) printf("\n");
  }
}
```

图 1.11 读取轨迹文件并处理访问语句的代码段

格式化读取轨迹文件中的轨迹语句，利用位移运算通过访问的数据地址得到 TAG 和组号，根据访问类型进行不同的模拟访问操作，访问类型为 M 时，进行两次模拟访问，访问类型为 L 或 S 时，进行一次模拟访问，访问类型为 I 时，不进行模拟访问。

1.3 故障与调试

1.3.1 故障与调试

运行测试程序发现只有一个测试用例通过，如图 1.12 所示。

```
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     0 (1,1,1)      -1      -1      -1      -1      -1      -1  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     0 (2,1,4)      -1      -1      -1      -1      -1      -1  traces/dave.trace
     0 (2,1,3)      -1      -1      -1      -1      -1      -1  traces/trans.trace
     0 (2,2,3)      -1      -1      -1      -1      -1      -1  traces/trans.trace
     0 (2,4,3)      -1      -1      -1      -1      -1      -1  traces/trans.trace
     0 (5,1,5)      -1      -1      -1      -1      -1      -1  traces/trans.trace
     0 (5,1,5)      -1      -1      -1      -1      -1      -1  traces/long.trace
     3

TEST_CSIM_RESULTS=3
```

图 1.12 测试不通过

分析源程序，发现在 cache 模拟访问函数中循环体开头将序号 i 转换为要访问的 cache 组对应的数组序号范围，而函数中又调用 SEtoseq 将 i 作为组内行号转换得到数组序号导致错误，如图 1.13 所示。将 i 更改为从 0 开始的组内行号，如图 1.14 所示，测试通过。

```
int maxlru=-1;
for(int i = S * Cache.E; i < (S + 1) * Cache.E; i++)
{
  if(Cache.cache[SEtoseq(S,i)][VALID])
  {
    //块有效
    if(Cache.cache[SEtoseq(S,i)][TAG]==tag)
```

图 1.13 重复从行号到序号的转换

```
int maxlru=-1;
for(int i=0;i<Cache.E;i++)
{
  if(Cache.cache[SEtoseq(S,i)][VALID])
  {
    //块有效
    if(Cache.cache[SEtoseq(S,i)][TAG]==tag)
```

图 1.14 去掉一个转换

1.3.2 测试与分析

运行测试程序 test-csim，程序运行结果如图 1.15 所示，测试通过。

```
brave@brave-VirtualBox:~/exp/cachelab-handout$ ./test-csim
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     3 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     3 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     3 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     3 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     3 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     3 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
     6 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    27

TEST_CSIM_RESULTS=27
```

图 1.15 test-csim 程序运行结果

1.4 源代码

```
#include "cachelab.h"
#include <getopt.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

#define VALID 0
#define TAG 1
#define LRUCOUNTER 2

#define HIT 0
#define MISS 1
#define EVICTION 2

FILE *file;
int Hits=0,Misses=0,Evictions=0,helpflag=0;
_Bool verbose=0;
void print_help()
{
    printf("help\n");
}

struct cache
{
    /* data */
    unsigned long s,E,b,S;
    unsigned long **cache;
}Cache;
```

```
void creatCache()
{
    Cache.S=1<<Cache.s;
    Cache.cache=(unsigned long**)malloc(Cache.E*Cache.S*sizeof(unsigned long*));
    for(int i=0;i<Cache.E*Cache.S;i++)
    {
        Cache.cache[i]=(unsigned long*)malloc(3*sizeof(unsigned long));
        Cache.cache[i][VALID]=0;
        Cache.cache[i][TAG]=0;
        Cache.cache[i][LRUCOUNTER]=0;
    }
}

void freeCache()
{
    for(int i=0;i<Cache.E*Cache.S;i++)
    {
        free(Cache.cache[i]);

    }
    free(Cache.cache);
}

void inputhandle(int argc,char **argv)
{
```

```
char opt;
while ((opt=getopt(argc,argv,"s:E:b:t:vh"))!=-1)
{
    switch (opt)
    {
    case 'h':
        print_help();
        helpflag=1;
        break;
    case 'v':
        verbose=1;
        break;
    case 's':
        Cache.s=atol(optarg);
        break;
    case 'E':
        Cache.E=atol(optarg);
        break;
    case 'b':
        Cache.b=atol(optarg);
        break;
    case 't':
        file=fopen(optarg,"r");
        break;

    default:
        break;
    }
}
```

```
}

unsigned long SEtoseq(unsigned long s,unsigned long E)
{
    return s*Cache.E+E;
}

unsigned long seqtoS(unsigned long seq)
{
    return seq / Cache.E;
}

unsigned long seqtoE(unsigned long seq)
{
    return seq % Cache.S;
}

int cacheaccess(unsigned long tag,unsigned long S)
{
    int res=-1;
    int emptyspace =    -1;
    int maxspace = -1;
    int maxlru=-1;
    for(int i=0;i<Cache.E;i++)
    {
        if(Cache.cache[SEtoseq(S,i)][VALID])
        {
            //块有效
            if(Cache.cache[SEtoseq(S,i)][TAG]==tag)
```

```
                {
                    //命中
                    Cache.cache[SEtoseq(S,i)][LRUCOUNTER]=0;
                    res = HIT;
                }
                else
                {
                    //该块未命中
                    Cache.cache[SEtoseq(S,i)][LRUCOUNTER]++;

if((signed)Cache.cache[SEtoseq(S,i)][LRUCOUNTER]>maxlru)
                    {
                        maxlru=Cache.cache[SEtoseq(S,i)][LRUCOUNTER];
                        maxspace=i;
                    }
                }
            }
            else
            {
                //块无效
                emptyspace = i;
            }
        }
        if(res==HIT) return res;
        else
        {
            if(emptyspace!=-1)
            {
                //有空闲块
```

```
                Cache.cache[SEtoseq(S,emptyspace)][VALID]=1;
                Cache.cache[SEtoseq(S,emptyspace)][TAG] =tag;
                Cache.cache[SEtoseq(S,emptyspace)][LRUCOUNTER]=0;
                res=MISS;
            }
            else
            {
                //无空闲块
                Cache.cache[SEtoseq(S,maxspace)][VALID]=1;
                Cache.cache[SEtoseq(S,maxspace)][TAG] =tag;
                Cache.cache[SEtoseq(S,maxspace)][LRUCOUNTER]=0;
                res=EVICTION;
            }
        }
        return res;
}

void reshandle(int res)
{
        switch (res)
        {
        case HIT:
            /* code */
            Hits++;
            if(verbose) printf(" hit");
            break;
        case MISS:
            Misses++;
            if(verbose) printf(" miss");
```

```
            break;
        case EVICTION:
            Misses++;
            Evictions++;
            if(verbose) printf(" miss eviction");
            break;
        default:
            break;
        }
}


int main(int argc,char* argv[])
{
    // //printSummary(0, 0, 0);

    char op[3];
    unsigned long tag=0;
    unsigned long S=0;
    int res=0;
    inputhandle(argc,argv);
    if(helpflag) return 0;
    creatCache();
    unsigned long addr,size;
    while (fscanf(file,"%s %lx,%lu",op,&addr,&size)!=EOF)
    {
        /* code */
        tag=addr>>(Cache.s+Cache.b);
        S=(((1<<(Cache.s+Cache.b))-1)&addr)>>Cache.b;
```

```
        switch (op[0])
        {
        case 'M':
            /* code */
            if(verbose) printf("%c %lx,%lu",op[0],addr,size);
            res=cacheaccess(tag,S);
            reshandle(res);
            res=cacheaccess(tag,S);
            reshandle(res);
            if(verbose) printf("\n");
            break;
        case 'L':
            if(verbose) printf("%c %lx,%lu",op[0],addr,size);
            res=cacheaccess(tag,S);
            reshandle(res);
            if(verbose) printf("\n");
            break;
        case 'S':
            if(verbose) printf("%c %lx,%lu",op[0],addr,size);
            res=cacheaccess(tag,S);
            reshandle(res);
            if(verbose) printf("\n");
            break;
        case 'I':
            if(verbose) printf("%c %lx,%lu",op[0],addr,size);
            if(verbose) printf("\n");
        }
    }
```

```
    printSummary(Hits,Misses,Evictions);
    freeCache();
    return 0;
}
```

2　实验心得

通过这次实验，对课上所讲的 cache 存储器的知识有了更加深入的认识，这次实验难度适中，而且可以很形象的认识 cache 的访问原理和 LRU 替换策略，是对课上所讲知识很好的巩固和补充。而且这次实验还给我们提供了阿里云的服务器，之前没有怎么接触过使用远程主机，这次也给我们一个机会体验一下使用 SSH 远程登陆服务器主机并进行操作，也算是一次别开生面的体验。通过这次实验我收获了很多，感觉是一次很棒的经历。

参考文献

[1] 张晨曦，王志英.计算机系统结构教程(第 2 版).北京：清华大学出版社.