Qixiang Liu, Weiting Wei
2856114,2755254
Due in class March 13, 2018
EECS 645 – Homework #04

Report:

Tasks:

Weiting did create cache array, read address from file. Also, she spilted address into three parts that are tags, lines, and blocks.

Qixiang Liu wrote while loop to distinguish addresses if they were hited or not. Also, I wrote the report.

a) Filename: Test.bin
Cache Parameters: CacheSize_Exp: 0000000F ; CacheSize_Nbr: 00008000
Address size: AddressSize_Exp: 00000020
Cache Associativity: 00000008
Block Parameters: BlockSize_Exp: 00000006; BlockSize_Nbr: 00000040;
BlockSize_Mask: 0000003F
Line Parameters: Lines_Exp: 00000006; Lines_Nbr: 00000040; Lines_Mask: 0000003F
Tag Parameters: Tag_Exp: 00000014; Tag_Nbr: 00100000; Tag_Mask: 000FFFFF
Digit Mode
Cache Parameters: CacheSize_Exp: 15 ; CacheSize_Nbr: 32768
Address size: AddressSize_Exp: 32
Cache Associativity: 8
Block Parameters: BlockSize_Exp: 6; BlockSize_Nbr: 64; BlockSize_Mask: 63
Line Parameters: Lines_Exp: 6; Lines_Nbr: 64; Lines_Mask: 63

Tag Parameters: Tag_Exp: 20; Tag_Nbr: 1048576; Tag_Mask: 1048575

b) Data

1)First Index:
hits: 321694
misses: 66787170
Access Number: 67108864
Hit/Access_Max: 0.48%

2)Last Index:
hits: 55249464
misses: 11859400
Access Number: 67108864
Hit/Access_Max: 82.33%

3)Random Index:

hits: 8233
misses: 67100631
Access Number: 67108864
Hit/Access_Max: 0.01%

c)

Source Code:

```
//  Created by Gary J. Minden on 9/24/15.
//  Copyright 2015 Gary J. Minden. All rights reserved.
//
//  Updates:
//  Author: Qixiang Liu, Weiting Wei
//  Date: 03/13/2018
//      B40922 -- Added a loop with random indices
//
//
//    Description:
//              This program simulates a multi-way direct
mapped cache.
//


#include <stdio.h>
#include <time.h>
#include <strings.h>
#include <stdlib.h>
#include <unistd.h>


#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>


//
//    Definitions of the cache
//
//    The cache size exponent and capacity in bytes
//
#define     CacheSize Exp      15
```

```
#define     CacheSize_Nbr       ( 1 << CacheSize_Exp )


//

//    Address size exponent

//

#define     AddressSize_Exp          32


//

//    The cache associativity

//

#define      CacheAssociativity_Exp3

#define     CacheAssociativity   ( 1 <<
CacheAssociativity_Exp )


//

//    The cache block size exponent, capacity in bytes, and
mask

//

#define     BlockSize_Exp       6

#define     BlockSize_Nbr       ( 1 << BlockSize_Exp )

#define     BlockSize_Mask      ( BlockSize_Nbr - 1 )


//

//    The number of lines in the cache. A line can contain
multiple blocks

//

#define     Lines_Exp               ( (CacheSize_Exp) -
(CacheAssociativity_Exp + BlockSize_Exp) )

#define     Lines_Nbr               ( 1 << Lines_Exp )

#define     Lines_Mask              ( Lines_Nbr - 1 )


//
```

```
//    Tag size exponent and mask

//

#define     Tag_Exp                ( AddressSize_Exp -
BlockSize_Exp - Lines_Exp )

#define     Tag_Nbr                ( 1 << Tag_Exp )

#define     Tag_Mask               ( Tag_Nbr - 1 )


//=======================================================
====================

//

//    Function to report defined values.

//

void ReportParameters ( char* theFilename ) {


     printf( "Filename: %s\n", theFilename );


     printf( "Cache Parameters: CacheSize_Exp: %08X ;
CacheSize_Nbr: %08X \n",

                    CacheSize_Exp, CacheSize_Nbr );


     printf( "Address size: AddressSize_Exp: %08X\n",
AddressSize_Exp );


     printf( "Cache Associativity: %08X\n",
CacheAssociativity );


     printf( "Block Parameters: BlockSize_Exp: %08X;
BlockSize_Nbr: %08X; BlockSize_Mask: %08X\n",

                    BlockSize_Exp, BlockSize_Nbr,
BlockSize_Mask );


     printf( "Line Parameters: Lines_Exp: %08X; Lines_Nbr:
%08X; Lines_Mask: %08X\n",

                    Lines_Exp, Lines_Nbr, Lines_Mask );
```

```
     printf( "Tag Parameters: Tag_Exp: %08X; Tag_Nbr: %08X;
Tag_Mask: %08X\n",

     Tag_Exp, Tag_Nbr, Tag_Mask );

     printf( "Digit Mode\n" );



     printf( "Cache Parameters: CacheSize_Exp: %d ;
CacheSize_Nbr: %d \n",

                    CacheSize_Exp, CacheSize_Nbr );



     printf( "Address size: AddressSize_Exp: %d\n",
AddressSize_Exp );



     printf( "Cache Associativity: %d\n",
CacheAssociativity );



     printf( "Block Parameters: BlockSize_Exp: %d;
BlockSize_Nbr: %d; BlockSize_Mask: %d\n",

                    BlockSize_Exp, BlockSize_Nbr,
BlockSize_Mask );



     printf( "Line Parameters: Lines_Exp: %d; Lines_Nbr:
%d; Lines_Mask: %d\n",

                    Lines_Exp, Lines_Nbr, Lines_Mask );



     printf( "Tag Parameters: Tag_Exp: %d; Tag_Nbr: %d;
Tag_Mask: %d\n",

                    Tag_Exp, Tag_Nbr, Tag_Mask );



}




typedef struct Cache{

     bool valid;
```

```
     uint32_t tag;
}Cache;
Cache cache[Lines_Nbr][CacheAssociativity];
//
//    Allocate a CacheLineState array.
//
//
//    Allocate a BlockValid array.
//
void createCache(){
     for(int i=0;i<Lines_Nbr;i++){
          for(int j=0;j<CacheAssociativity;j++){
               cache[i][j].valid = false;
               cache[i][j].tag = 0;
          }
     }
}

//
//    Round Robin State
//
int roundRobinState(uint32_t cacheTag){
     int randomLine = rand()%Lines_Nbr;
     int randomBlock = rand()%CacheAssociativity;
     cache[randomLine][randomBlock].tag= cacheTag;
     return randomBlock;
}
//=========================================================
==================
//
//    Main function
```

```
//

int main (int argc, const char * argv[]) {


     //    Open address trace file, reset counters, and
process Accesses_Max addresses

     //

     double hits;

     double misses;

     FILE* iFile;

     uint32_t address;

     int choice;

     uint32_t tags = 0;

     uint32_t line = 0;;

     bool found = 0;

     char* filePath = "";

     srand(time(NULL));

     long Access_Nbr = 0;

     long Access_Max = 64*1024*1024;

     int blockIndex = 0;


ReportParameters( "Test.bin" );

//

//    Process all files.

//

while(choice !=4){

printf("\nPlease choose a binary file:\n");

printf("1) First Index\n");

printf("2) Last Index \n");

printf("3) Random Index\n");

printf("4) Exit\n");

printf("Choice: ");
```

```
scanf("%d", &choice);
//
//    Report cache parameters
//
if(choice == 1){
     filePath = "AddressTrace_FirstIndex.bin";
}else if(choice ==2){
     filePath = "AddressTrace_LastIndex.bin";
}else if(choice==3){
     filePath = "AddressTrace_RandomIndex.bin";
}else{
     break;
}
     //
     //    Open address trace file, reset counters, and
process Accesses_Max addresses
     //
     if((iFile = fopen(filePath, "rb")) == NULL)
     {
          fputs("Fail read",stderr);
          exit(1);
     }
     //    Flush cache
     createCache();
     //    Extract fields from address
     Access_Nbr = 0;
     hits = 0;
     misses = 0;
     blockIndex = 0;
     while(Access_Nbr<Access_Max){
               //4 byte = 32bits/8, each time read once!
```

```
            size_t readFlag =fread(&address,4,1,iFile);

               if(readFlag == 0){

                    printf("Some wrong OR read end of
file");

                    break; //check endof or problem

               }

               //shift right 6

               //    Print cache line state

               //    Print tags

               line = (address >> BlockSize_Exp) &
Lines_Mask;

               tags = ((address >> (BlockSize_Exp +
Lines_Exp)) & Tag_Mask);



               //Iterate through each line of the cache,
checking each block for hits

               for(blockIndex=0;blockIndex <
CacheAssociativity; blockIndex++)

               {

                    //         printf( "Cache Input:
Accesses_Nbr: %08X; Address: %08X; Tag: %08X; Line: %08X;
Block: %08X\n",

     //                   Access_Nbr, address, tags, line,
blockIndex );

                    //    Check if address is in the cache

                    if(cache[line][blockIndex].valid ==
true){

             if(cache[line][blockIndex].tag == tags){

                              //

                              //    Print cache line state

                              //printf( "Cache State:
BlockValid: %02X; CacheTag: %08X\n",

                              //              cache[line]
[blockIndex], tags );

               hits++;

               break;
```

```
              break;

            }

            else{


              if(blockIndex == CacheAssociativity-1){

                misses++;

                int RoundRobinNextReplacement=
roundRobinState(tags);

              }

                                //printf( "Cache Action:
CacheHit: %01d; InvalidBlock_Idx: %02d;
RoundRobinNextReplacement: %02d\n",

                                //         hits,
misses,RoundRobinNextReplacement);

            }

          }

          else {

            cache[line][blockIndex].valid = true;

            cache[line][blockIndex].tag = tags;

            misses++;

                          //printf( "Find free block:
InvalidBlock_Idx: %02d\n", indexBlock);

            break;

          }

              }

              Access_Nbr++;

    }


    //    Report cache performance

    //

    float hitRatio = ((hits/Access_Max)*100.0);

    printf("hits: %.00f\n", hits);

    printf("misses: %.00f\n", misses);
```

```
        printf("Access Number: %ld\n",Access_Max);
        printf("Hit/Access_Max: %.02f%%\n", hitRatio);
    }
        //Close the trace file
        fclose(iFile);

        return 1;

    }
```