**דוד מוסייב**

**יקיר מיימון**

**שאלה 2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

/\*structs\*/

typedef struct cacheTable

{

struct cacheSet\_S \*set;

}CacheTable\_S;

enum Bool

{

TRUE,FALSE

};

typedef enum Bool\_t Bool;

typedef struct cacheSet

{

long unsigned int setVal[2];//number of the set

int blockNum;//number of block

char validFlag;//flag if 0 unvalid if 1 valid

long unsigned int lruHistory[3];//least reasent

}CacheSet\_s;

double oneSet(FILE \*fp, int cacheSize, int blockSize);//function will count for 1set associativity number of cache misses as a precantage of number of memory reads

double twoSet(FILE \*fp, int cacheSize, int blockSize);//function will count for 2set associativity count number of cache misses as a precantage of number of memory reads

void enterLruHist(int address, CacheSet\_s \*setOnCache);//enter a value in history of the LRU for 2set associatvity

int getLruNum(int address, CacheSet\_s setOnCache);//get a value from history of the LRU for 2set associatvity

double FullSet(FILE \*fp, int cacheSize, int blockSize);//function will count for 8set associativity number of cache misses as a precantage of number of memory reads

CacheSet\_s \* findFreeSet(CacheSet\_s \* initialSet, int numOfBlocks);//find a free set for full associative

CacheSet\_s \* findOldest(CacheSet\_s \* initialSet, int numOfBlocks, unsigned long int \* LRU);//find a oldest set for full associative

Bool findMatch(CacheSet\_s \* initialSet, int numOfBlocks, unsigned long int tag);//find a match for full associative

void enterLRU(unsigned long int \* LRU, int numOfBlocks, unsigned long int tag);//enter a value for LRU history for full associative

void main(int argc, char\* argv[])

{

printf("\n%s", argv[1]);//cahceSize

printf("\n%s", argv[2]);//associative of cache

printf("\n%s", argv[3]);//block size

printf("\n%s", argv[4]);//benchmark file

int cacheSize = atoi(argv[1]);

int blockSize = atoi(argv[3]);

FILE \*traceBM = fopen(argv[4],"r");//open trace file

if (argv[2][0] == '1')//1 set associative

{

printf("\nthe miss ratio is: %f", oneSet(traceBM, cacheSize, blockSize));

}

if (argv[2][0] == '2')//2 set associative

{

printf("\nthe miss ratio is: %f", twoSet(traceBM, cacheSize, blockSize));

}

if (argv[2][0] == '8')//8 set associative - Full associative

{

printf("\nthe miss ratio is: %f", FullSet(traceBM, cacheSize, blockSize));

}

}

double oneSet(FILE \* fp, int cacheSize, int blockSize)

{

/\*simplicty\*/

FILE \*thisFile = fp;

int numOfBlock = cacheSize / blockSize;//number of block

unsigned int tag;//holding a tag value for each address

unsigned int memAdd = 0;//address of the word

unsigned int blockCacheAdd = 0;//will hold the right block that each address should get inside

int missCt = 0;//miss counter

unsigned numOflines = 0;//number of lines counter

/\*create table with one set \*number of blocks\*/

CacheSet\_s \*\*tableOfSets = (CacheSet\_s\*\*)calloc(1, sizeof(CacheSet\_s\*));

\*tableOfSets = (CacheSet\_s\*)calloc(numOfBlock, sizeof(CacheSet\_s));

/\*simplify holding a pointer\*/

CacheSet\_s \*thisSet = (\*tableOfSets);

while(!feof(thisFile))//until the end of the file keep doing..

{

fscanf(thisFile, "%u", &memAdd);//into block address

numOflines++;//count lines

blockCacheAdd = (memAdd % blockSize);//cache block after modulo its also the differnets value between the memory address in the file to its block address

if (blockCacheAdd != 0)

{

tag=(memAdd - blockCacheAdd);//get the valid block tag

blockCacheAdd = tag / blockSize;//index of the block

}

else

{

//its a beggining of the block

tag = memAdd;//valid tag

blockCacheAdd = tag / blockSize;//index of the block

}

while (blockCacheAdd >= numOfBlock)//use it when u block index higher then the posible values of the index blocks

{

blockCacheAdd %= numOfBlock;//its module of number of blocks as referance to of index posiblities

}

printf("\nThe Set number is:%u\tMemory block is:%u\n\n", blockCacheAdd, tag);

if ((thisSet[blockCacheAdd].validFlag == 1) && thisSet[blockCacheAdd].setVal[0] == tag)//condition for hit

{

printf("\nhit in line number %u\n",numOflines);//hit message

//already has the same value

}

else

{

thisSet[blockCacheAdd].setVal[0] = tag;//update the tag on this set

thisSet[blockCacheAdd].validFlag = 1;//change validate value

missCt++;//count as miss

}

}

double ratio = (double)missCt / (double)numOflines;//ratio between the number of misses and the number of lines

free(tableOfSets);

return ratio;

}

double twoSet(FILE \* fp, int cacheSize, int blockSize)

{

/\*simplicty\*/

FILE \*thisFile = fp;

unsigned int tag;//holding a tag value for each address

int numOfBlock = 2 \* (cacheSize / blockSize);//number of block in total

unsigned long int memAdd = 0;//memory address from file

unsigned long int blockCacheAdd = 0;//the right block/set

int missCt = 0;//miss counter

unsigned numOflines = 0;//lines counter

/\*create table with two sets \*number of blocks\*/

CacheSet\_s \*\*tableOfSets = (CacheSet\_s\*\*)calloc(1, sizeof(CacheSet\_s\*));

(tableOfSets)[0] = (CacheSet\_s\*)calloc(numOfBlock, sizeof(CacheSet\_s));

CacheSet\_s \*thisSet = (\*tableOfSets);

while (!feof(thisFile))

{

fscanf(thisFile, "%u", &memAdd);//into block address

numOflines++;//lines count

blockCacheAdd = (memAdd % blockSize);//cache block after modulo its also the differnets value between the memory address in the file to its block address

if (blockCacheAdd != 0)

{

tag = (memAdd - blockCacheAdd);

blockCacheAdd = tag / blockSize;

}

else

{

tag = memAdd;

blockCacheAdd = tag / blockSize;

}

while (blockCacheAdd >= numOfBlock)

{

blockCacheAdd %= numOfBlock;

}

/\*some user messages\*/

printf("\n\n\t\t\tLine number %u", numOflines);

printf("\n\t\t\tThe Set number is:%u\tMemory block is:%u", blockCacheAdd, tag);

printf("\n\t\t\tThe LRU is:%u ,%u ,%u ", thisSet[blockCacheAdd].lruHistory[0], thisSet[blockCacheAdd].lruHistory[1], thisSet[blockCacheAdd].lruHistory[2]);

if ((thisSet[blockCacheAdd].validFlag == 1) && ((thisSet[blockCacheAdd].setVal[0] == tag) || (thisSet[blockCacheAdd].setVal[1] == tag)))//hit conditions

{

printf("\t\t\tHit in line number %u\n", numOflines);

enterLruHist(tag, &(thisSet[blockCacheAdd]));//enter the value into the LRU history

//already has the same value

}

else

{

if (getLruNum(thisSet[blockCacheAdd].setVal[0],thisSet[blockCacheAdd]) > getLruNum(thisSet[blockCacheAdd].setVal[1], thisSet[blockCacheAdd]))//int the 1st partion is older so replace it with new value

{

//if in location 0 is older value

thisSet[blockCacheAdd].validFlag = 1;

thisSet[blockCacheAdd].setVal[0] = tag;

}

else

{

//if in location 1 is older value

thisSet[blockCacheAdd].setVal[1] = tag;

}

thisSet[blockCacheAdd].validFlag = 1;//change validate value

enterLruHist(tag, &thisSet[blockCacheAdd]);//enter the value into history

missCt++;

}

}

double ratio = (double)missCt / (double)numOflines;//ratio between the number of misses and the number of lines

free(tableOfSets);

return ratio;

}

void enterLruHist(int address, CacheSet\_s \*setOnCache)

{

//enter a value to small LRU history for 2 associated

setOnCache->lruHistory[2] = setOnCache->lruHistory[1];

setOnCache->lruHistory[1] = setOnCache->lruHistory[0];

(setOnCache->lruHistory)[0] = address;

}

int getLruNum(int address, CacheSet\_s setOnCache)

{

int k;

for (int i = 0; i < 4; i++)

{

if (address == setOnCache.lruHistory[i])

{

k = i;

return k;

}

}

return 100;

}

double FullSet(FILE \* fp, int cacheSize, int blockSize)

{

/\*simplicty\*/

FILE \*thisFile = fp;

unsigned long int tag=0;

int numOfBlock = 2 \* (cacheSize / blockSize);

unsigned long int memAdd = 0;//block add

unsigned long int blockCacheAdd = 0;

int missCt = 0;

unsigned numOflines = 0;

/\*create array with LRU history\*/

unsigned long int \*LRU = (unsigned long int\*)calloc(numOfBlock, sizeof(unsigned long int));

/\*create table with two sets \*number of blocks\*/

CacheSet\_s \*\*tableOfSets = (CacheSet\_s\*\*)calloc(1, sizeof(CacheSet\_s\*));

\*tableOfSets = (CacheSet\_s\*)calloc(numOfBlock, sizeof(CacheSet\_s));

CacheSet\_s \*thisSet = tableOfSets;

while (!feof(thisFile))

{

thisSet = \*tableOfSets;

/\*read the lines\*/

fscanf(thisFile, "%u", &memAdd);//into block address

numOflines++;

blockCacheAdd = (memAdd % blockSize);//cache block after modulo

if (blockCacheAdd != 0)

{

tag = (memAdd - blockCacheAdd);

}

else

{

tag = memAdd;

}

/\*logics\*/

if (findMatch(\*tableOfSets, numOfBlock, tag) == TRUE)//check if there is a match with all the posible sets

printf("\nHit in line:\t%d\n\n", numOflines);//HIT!

if (findMatch(\*tableOfSets, numOfBlock, tag) == FALSE)//if miss

{

/\*find a free set or oldiest set in cache\*/

if (findFreeSet(\*tableOfSets, numOfBlock) != NULL)//if there is a free set

{

thisSet = findFreeSet(\*tableOfSets, numOfBlock);//assign it to thisSet for values setting

}

else//if there is no free ,find the oldiest value

{

thisSet = findOldest(\*tableOfSets, numOfBlock, LRU);//go the LRU history and find the oldiest value in the cache after this assign that location to this set

}

/\*set the values\*///we get this stage with index of the cache that we going to store there a value of the tag of the memory

thisSet->setVal[0] = tag;

thisSet->validFlag = 1;//for next time it would be availble for checking

enterLRU(LRU, numOfBlock, tag);//enter the LRU history

missCt++;//count the miss

}

printf("\nInLine:%u", numOflines);

printf("\ntag is:%u", tag);

printf("\nThis Set Value is:%u\n", thisSet->setVal[0]);

}

double ratio = (double)missCt / (double)numOflines;

free(tableOfSets);

return ratio;

}

CacheSet\_s \*findFreeSet(const CacheSet\_s \*initialSet,int numOfBlocks)

{

for(int i=0;i<numOfBlocks;i++)//find a free set for storage the chache value

{

if (initialSet->validFlag == 0)//if its never used since the start

{

printf("\nFree set found,its value is:%d in position %d\n", \*initialSet->setVal,i);

return initialSet;

}

initialSet++;//check the next one

}

return NULL;

}

CacheSet\_s \*findOldest(const CacheSet\_s \*initialSet, int numOfBlocks, unsigned long int \*LRU)

{

for (int i = 0; i < numOfBlocks-1; i++)//find the oldiest value in the LRU and the object in the cache that matched to this value

{

if (LRU[numOfBlocks - 1] == initialSet->setVal)

{

printf("\nThe oldest LRU found,its value is:%d", initialSet->setVal);

return initialSet;

}

else

{

initialSet++;

}

}

return initialSet;

}

Bool findMatch(const CacheSet\_s \*initialSet, int numOfBlocks,unsigned long int tag)

{

for (int i = 0; i < numOfBlocks-1; i++)//search among the values in the blocks if there is a match to block tag if there is thats a hit =return TRUE else return FALSE

{

if (initialSet->validFlag != 0)

{

if (\*initialSet->setVal == tag)

{

return TRUE;

}

}

(initialSet)++;

}

return FALSE;

}

void enterLRU(unsigned long int \*LRU, int numOfBlocks,unsigned long int tag)

{

for (int i = 0; i < numOfBlocks; i++)/\*the new value going to the head of the queue and the last one going out all other values move toward the tail\*/

{

LRU[numOfBlocks - i] = LRU[numOfBlocks - i - 1];

}

LRU[0] = tag;

}

**1set**

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 1024 1 16 trace1.txt

the miss ratio is: 1.000000

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 1024 1 16 trace2.txt

the miss ratio is: 0.250000

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 1024 1 16 trace3.txt

the miss ratio is: 1.000000

**2set**

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 2048 2 32 trace1.txt

the miss ratio is: 0.500000

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 2048 2 32 trace2.txt

the miss ratio is: 0.125000

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 2048 2 32 trace3.txt

the miss ratio is: 1.000000

**8set-fullset**

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 2048 8 32 trace1.txt

the miss ratio is: 0.503906

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 2048 8 32 trace2.txt

the miss ratio is: 0.125000

C:\Users\David\source\repos\cacheBanchMark\Debug\cacheBanchMark.exe 2048 8 32 trace3.txt

the miss ratio is: 1.000000