## 2-Way Associative

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Computer Architecture 3 Formal Element Cache Simulator

2-Way Set Associative Cache

32-bit CPU Addr, 8-bit data bus, 4 Byte data lines, 64kB data cache

Written in VS Studio 2017 by Jack Harding

08/11/2018

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "pch.h"

using namespace std;

struct cacheEntry {

unsigned short upprAddr; // creating variable for upper address

bool validFlag; // valdid flag indicates wether or not cache entry is valid

bool LRUFlag; // least recently used flag indicates if the cache entry was the last to be used

char data[4]; // array for storing data

};

cacheEntry way0[16384], way1[16384];

string addrS;

ifstream myAddr("testAddresses.txt"); // reading in file with addresses

unsigned int CPUAddr; // 32-bit CPU addr

unsigned char byteNo; // 8-bit No.

unsigned short CPUUppr, setNo, hits = 0, misses = 0;// tag addr, which elemnt in way, hit, miss counter

int main()

{

for (int i = 0; i <= 16384; i++) { // init flags before adress are sent

way0[i].validFlag = 0; // not valid init

way0[i].LRUFlag = 1; // store in way0 first: LRU

way1[i].validFlag = 0;

way1[i].LRUFlag = 0; //store in way0 first, way1 second: MRU

}

cout << "2-Way Set Associative Cache Simulator \n\n";

if (myAddr.is\_open()) {

while (getline(myAddr, addrS)) {

stringstream tempS(addrS); // reading first line of addr file

tempS >> hex >> CPUAddr; // string to hex

CPUUppr = (CPUAddr & 0xffff0000) >> 16;// bit shifting by 16, anded to get CPU upper addr

setNo = (CPUAddr & 0x0000fffc) >> 2; // and with fffc, shift by 2 bits

byteNo = CPUAddr & 3; // and CPU addr with 3 to show last 2 bits

if ((way0[setNo].upprAddr == CPUUppr) && (way0[setNo].validFlag == 1)) { // hit in way0

string\* pByte = (string\*)byteNo; // typecasts the byteNo to printed

cout << "Full Addr: 0x" << hex << CPUAddr;

cout << " Hit: 0x" << hex << CPUUppr << setNo << " " << pByte << endl; // formatting

hits++; // incrementing hit counter everytime a hit occurs

way0[setNo].LRUFlag = 0; // way0 to most recently used

way1[setNo].LRUFlag = 1; // way1 to least recently used

}

else if ((way1[setNo].upprAddr == CPUUppr) && (way1[setNo].validFlag == 1)) { // hit in way1

string\* pByte = (string\*)byteNo;

cout << "Full Addr: 0x" << hex << CPUAddr;

cout << " Hit: 0x" << hex << CPUUppr << " way1 " << setNo<< setNo << " " << pByte << endl;

hits ++;

way1[setNo].LRUFlag = 0; // way1 to most recently used (FIFO)

way0[setNo].LRUFlag = 1; // way0 to least recently used (LRU)

}

else { // miss

if (way0[setNo].LRUFlag == 1) { // if way0 is LRU

string\* pByte = (string\*)byteNo;

cout << "Full Addr: 0x" << hex << CPUAddr;// tidies output

cout << " Miss: 0x" << hex << CPUUppr << " way0 " << setNo << " " << pByte << endl;

way0[setNo].upprAddr = CPUUppr;

way0[setNo].validFlag = 1;

way0[setNo].LRUFlag = 0; // way0 to most recently used

way1[setNo].LRUFlag = 1; // way1 to least recently used

misses ++; // miss

}

else { // when way1 is LRU

string\* pByte = (string\*)byteNo;

cout << "Full Addr: 0x" << hex << CPUAddr;

cout << " Miss: 0x" << hex << CPUUppr << " way1 " << hex << setNo << " " << pByte << endl;

way1[setNo].upprAddr = CPUUppr; // sets the upper address in way1[setNo] to CPUUppr

way1[setNo].validFlag = 1; // entry is valid

way1[setNo].LRUFlag = 0; // way1 to most recently used

way0[setNo].LRUFlag = 1; // way0 to least recently used

misses ++;

}

}

}

myAddr.close(); // closes file

}

else cout << "Unable to openfile"; // prints error message in case file can't be opened

cout << "\nTotal Hits: " << dec << hits << "\nTotal Misses: " << dec << misses << endl;

system("pause");

return 0;

}