## Fully Associative

/\*

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

Computer Architecture 3 Formal Element Cache Simulator

Fully 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"

struct cacheEntry {

unsigned int fAddr;

bool validFlag; // is entry valid

bool LRUFlag; // least recently used

char data[4];

};

cacheEntry way0[16384];

string lineSel;

ifstream myAddr("testAddresses.txt");

unsigned int CPUAddr; // full 32-bit addr

unsigned short CPUUppr, hits = 0, misses = 0, setNo = 0, counter = 0; // tag addr, hits, misses counter, set no., counter for index

unsigned char byteNo;

bool isHit = 0, isFull = 0; // hit occurs, cache is full

int main()

{

cout << "Fully Associative Cache Simulator\n\n";

for (int i = 0; i <= 16384; i++){ // init flags

way0[i].validFlag = 0;

way0[i].LRUFlag = 1;

}

if (myAddr.is\_open()){

while (getline(myAddr, lineSel)) {

stringstream tempS(lineSel); // selects line

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

CPUUppr = (CPUAddr & 0xffff0000) >> 16; // bit shift 16, and CPU addr0xffff0000

byteNo = CPUAddr & 3; // anded with 3 for byte no.

if (!isFull) { // is cache full

while (setNo <= counter) { // used when is not full to search

if ((way0[setNo].fAddr == CPUAddr) && (way0[setNo].validFlag == 1)){ // hit occurs when addr match and entry valid

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

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

cout << " Hit: 0x" << hex << CPUUppr << setNo << " " << endl; // formatted correctly

hits++; // hit counter incremeneted

isHit = true; // a hit is detected

}

setNo++; // must be incremented

}

if (!isHit){ // miss

if (counter == 0) { // first location of cache

way0[counter].fAddr = CPUAddr; // stores addr

way0[counter].validFlag = 1; // is valid

way0[counter].LRUFlag = 0; // most recently used when LRU=0

counter++; // inrementing counter for last loc

}

else { // when not at first addr

for (int i = counter;i > 0; i--) { // iterating through old addr

way0[i] = way0[i - 1]; // moving forward one element

way0[i].LRUFlag = 1;

}

way0[0].fAddr = CPUAddr; // storing addr in way

way0[0].validFlag = 1;

way0[0].LRUFlag = 0;

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

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

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

counter++;

misses++; // miss occurs

if (counter == 16384) // cache capacity=16384

isFull = true; // cache is full

}

}

}

else { // cache full

for (int i = 0; i < 16384; i++) { // not using setNo, all cache entries have usable addr

if ((way0[i].fAddr == CPUAddr) && (way0[i].validFlag == 1)) {

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

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

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

hits++; // hit occurs

isHit = true; // prog knows a hit happened

}

else { // miss

for (int i = 16384; i > 0; i--) {

way0[i] = way0[i - 1]; // moving all addrby one element

way0[i].LRUFlag = 1;

}

way0[0].fAddr = CPUAddr;

way0[0].validFlag = 1;

way0[0].LRUFlag = 0; // most recently used (FIFO)

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

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

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

misses++;

}

}

}

}

myAddr.close(); // closes file when done

}

else

cout << "\nCan't open file\n"; // error with file

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

cout << "Total Misses: " << dec << misses << endl; // must be decimal for values > 9

system("pause");

return 0;

}