**main.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jeffery Lim

Taylor Nguyen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <iostream>

#include <string.h>

#include <math.h>

#include "lru.h"

#include "cache.h"

**using** **namespace** std**;**

void readConfig**(**char**\*** file**,** cache**\*** L1\_I**,** cache**\*** L1\_D**,** cache**\*** L2**);**

void readTrace**(**cache**\*** L1\_I**,** cache**\*** L1\_D**);**

void printReport**(**char**\*** sim**,** char**\*** config**,** cache**\*** L1\_I**,** cache**\*** L1\_D**,** cache**\*** L2**);**

unsigned long long int instruction\_count**=** 0**;** //number of instructions

unsigned long long int read\_count**=** 0**;** //number of reads

unsigned long long int write\_count**=** 0**;** //number of writes

unsigned long long int read\_time**=** 0**;** //the amount of read time in cycles

unsigned long long int write\_time**=** 0**;** //the amount of write time in cycles

unsigned long long int inst\_time**=** 0**;** // the amount of instruction time in cycles

unsigned int chunksize **=** 8**;** //main memory chunk size.

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

//Summon the three caches

cache**\*** L1\_D **=** **new** cache**;**

cache**\*** L1\_I **=** **new** cache**;**

cache**\*** L2 **=** **new** cache**;**

cout **<<** "Begin Configurations" **<<** endl**;**

//argc == 3 means that there is a config file; otherwise, use the defaults

//Normally it should be argc == 2, but since we did not want to mess with writing shell scripts

//argc == 3 because we included the trace file name. This makes writing the report easier.

**if(**argc **==** 3**){**

cout **<<** "Reading config file" **<<** endl**;**

readConfig**(**argv**[**2**],** L1\_I**,** L1\_D**,** L2**);**

**}else{**

//no argument means we are using default case

L1\_I**->**set\_cache**(**32**,** 8192**,** 1**,** 1**,** 1**,** 10**,** 16**,** L2**);**

L1\_D**->**set\_cache**(**32**,** 8192**,** 1**,** 1**,** 1**,** 10**,** 16**,** L2**);**

L2**->**set\_cache**(**64**,** 32768**,** 1**,** 8**,** 10**,** 15**,** 8**,** **NULL);** //Null means Main Memory

**}**

cout **<<** "Reading Traces" **<<** endl**;**

readTrace**(**L1\_I**,** L1\_D**);**

/\*

cout << "L1\_I:" <<endl;

L1\_I->print();

cout << "L1\_D:" <<endl;

L1\_D->print();

cout << "L2:" <<endl;

L2->print();

\*/

cout **<<** "Printing Report" **<<** endl**;**

printReport**(**argv**[**1**],** argv**[**2**],** L1\_I**,** L1\_D**,** L2**);** //Print out the report to an external file

cout **<<**"Deconstructing Caches" **<<** endl**;**

L1\_D**->**cache\_deconstruct**();**

L1\_I**->**cache\_deconstruct**();**

L2**->**cache\_deconstruct**();**

**delete** L1\_D**;**

**delete** L1\_I**;**

**delete** L2**;**

cout **<<**"Finished"**;**

**return** 0**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void readConfig(char\* file, cache\* L1\_I, cache\* L1\_D, cache\* L2)

char\*file is the config file config

cache\* is the pointer to point to the three caches

This function reads the config files and sets the cache parameters with the proper values

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void readConfig**(**char**\*** file**,** cache**\*** L1\_I**,** cache**\*** L1\_D**,** cache**\*** L2**){**

unsigned int bs **=** 0**,** bs1 **=** 0**;** // block size

unsigned int cs **=** 0**,** cs1 **=** 0**;** // cache sinceze

unsigned int assoc **=** 0**,** assoc1 **=** 0**;** // ways

char input**[**20**];**

unsigned int value**;**

FILE **\***config **=** fopen**(**file**,** "r"**);**

**if(!**config**){**

L1\_I**->**set\_cache**(**32**,**8192**,** 1**,** 1**,** 1**,** 10**,** 16**,** L2**);**

L1\_D**->**set\_cache**(**32**,**8192**,** 1**,** 1**,** 1**,** 10**,** 16**,** L2**);**

L2**->**set\_cache**(**64**,** 32768**,** 1**,** 8**,** 10**,** 15**,** 8**,** **NULL);** //Null means Main Memory

**return;**

**}else{**

**while** **(**fscanf**(**config**,**"%s %d\n"**,** input**,** **&**value**)** **==** 2**)** **{**

**if(**strcmp**(**input**,** "L1\_block\_size"**)** **==** 0**){**

bs **=** value**;**

**}else** **if(**strcmp**(**input**,** "L1\_cache\_size"**)** **==** 0**){**

cs **=** value**;**

**}else** **if(**strcmp**(**input**,** "L1\_assoc"**)** **==** 0**){**

assoc **=** value**;**

**}**

**if(**strcmp**(**input**,** "L2\_block\_size"**)** **==** 0**){**

bs1 **=** value**;**

**}else** **if(**strcmp**(**input**,** "L2\_cache\_size"**)** **==** 0**){**

cs1 **=** value**;**

**}else** **if(**strcmp**(**input**,** "L2\_assoc"**)** **==** 0**){**

assoc1 **=** value**;**

**}**

**if(**strcmp**(**input**,** "mem\_chunksize"**)** **==** 0**){**

chunksize **=** value**;**

**}**

**}**

L1\_I**->**set\_cache**(**bs**,**cs**,** assoc**,** 1**,** 1**,** 10**,** 16**,** L2**);**

L1\_D**->**set\_cache**(**bs**,**cs**,** assoc**,** 1**,** 1**,** 10**,** 16**,** L2**);**

L2**->**set\_cache**(**bs1**,**cs1**,** assoc1**,** 8**,** 10**,** 15**,** chunksize**,** **NULL);**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void readTrace(cache\* L1\_I, cache\* L1\_D)

cache\* is the pointer to point to the two L1 caches

This function reads the trace files, figures out the number of requests needed to be made and then proceeds to call the

necessary find function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void readTrace**(**cache**\*** L1\_I**,** cache**\*** L1\_D**){**

char op**;**

unsigned long long int address**;**

unsigned int bytesize**;**

unsigned int references**;**

//scanf the lines by op code, address, and byte

**while(**scanf**(**"%c %Lx %d\n"**,** **&**op**,** **&**address**,** **&**bytesize**)** **==** 3**)**

**{**

references **=** **(**address**%**4 **+** bytesize**-**1**)/**4 **+** 1**;** //Address%4 gives us location of the start of block

//Adding it to bytesize-1 gives us the number of total bytes

//Dividing it by 4 gives us the number of references needed

//ex: 0003 with 6 bytesize means we want to start at 0003 and end 0009

//0003%4 = 3 + 6 - 1 = 8/2 = 2 + 1 = 3 references

**switch(**op**){**

**case** 'I'**:**

instruction\_count**++;**

inst\_time **+=** 1**;** //Execute cycle time

**for(**unsigned int i **=** 0**;** i **<** references**;** i**++){**

inst\_time **+=** L1\_I**->**find**(**address **+** i**\***4**,** op**);**

**}**

**break;**

**case** 'R'**:**

read\_count**++;**

**for(**unsigned int i **=** 0**;** i **<** references**;** i**++){**

read\_time **+=** L1\_D**->**find**(**address **+** i**\***4**,** op**);**

**}**

**break;**

**case** 'W'**:**

write\_count**++;**

**for(**unsigned int i **=** 0**;** i **<** references**;** i**++){**

write\_time **+=** L1\_D**->**find**(**address **+** i**\***4**,** op**);**

**}**

**break;**

**default:**

**break;**

**}**

**}**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void printReport(char\* sim, char\* config, cache\* L1\_I, cache\* L1\_D,cache\* L2)

char\* sim is the trace file config

char\* config is the name of the config file

cache\* is the pointer to point to the two L1 caches

This function reads the trace files, figures out the number of requests needed to be made and then proceeds to call the

necessary find function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printReport**(**char**\*** sim**,** char**\*** config**,** cache**\*** L1\_I**,** cache**\*** L1\_D**,**cache**\*** L2**){**

FILE **\*** report**;**

char result**[**50**];**

//default case will be empty

//copy the string into results and concatentate teh rest of the names into it

strcpy**(**result**,** sim**);**

strcat**(**result**,** "."**);**

**if(**config **==** **NULL){**

config **=** "default"**;**

**}**

strcat**(**result**,** config**);**

report **=** fopen**(**result**,** "w"**);**

unsigned long long int execution **=** inst\_time **+** read\_time **+** write\_time**;** //calculate the execution time

unsigned long long int total **=** read\_count**+**write\_count**+**instruction\_count**;**

unsigned long long int L1Cost **=** L1\_I**->**get\_cache\_size**()/**4096**\***100**+**L1\_I**->**get\_cache\_size**()/**4096**\*(**log2**(**L1\_I**->**get\_ways**()))\***100**;**

unsigned long long int L1Cost1 **=** L1\_D**->**get\_cache\_size**()/**4096**\***100**+**L1\_D**->**get\_cache\_size**()/**4096**\*(**log2**(**L1\_D**->**get\_ways**()))\***100**;**

unsigned long long int L2Cost **=** L2**->**get\_cache\_size**()/**16384**\***50**+** L2**->**get\_cache\_size**()/**16384**\*(**log2**(**L2**->**get\_ways**()))\***50**;**

unsigned long long int memCost **=** 50**+**25**+**100**\*(**log2**(**chunksize**)-**log2**(**8**));**

fprintf**(**report**,** "\n-------------------------------------------------------------------------\n"**);**

fprintf**(**report**,** " %s.%s Simulation Results\n"**,** sim**,** config**);**

fprintf**(**report**,** "-------------------------------------------------------------------------\n\n"**);**

fprintf**(**report**,** " Memory system: \n"**);**

fprintf**(**report**,** " Dcache size = %u : ways = %u : block size = %u \n"**,** L1\_D**->**get\_cache\_size**(),** L1\_D**->**get\_ways**(),** L1\_D**->**get\_block\_size**());**

fprintf**(**report**,** " Icache size = %u : ways = %u : block size = %u \n"**,** L1\_I**->**get\_cache\_size**(),** L1\_I**->**get\_ways**(),** L1\_I**->**get\_block\_size**());**

fprintf**(**report**,** " L2-cache size = %u : ways = %u : block size = %u \n"**,** L2**->**get\_cache\_size**(),** L2**->**get\_ways**(),** L2**->**get\_block\_size**());**

fprintf**(**report**,** " Memory ready time = %u : chunksize = %u : chunktime = %u \n\n"**,** 50**,** chunksize**,** 15**);**

fprintf**(**report**,** " Execute time = %llu; Total refs = %llu\n"**,** execution**,** total**);**

fprintf**(**report**,** " Inst refs = %llu; Data refs = %llu\n\n"**,** instruction\_count **,** write\_count**+**read\_count**);**

fprintf**(**report**,** " Number of reference types: [Percentage]\n"**);**

fprintf**(**report**,** " Reads =%13llu [%4.1f%%]\n"**,** read\_count**,** **(**double**)**read\_count**/(**double**)(**total**)\***100**);**

fprintf**(**report**,** " Writes =%13llu [%4.1f%%]\n"**,** write\_count**,** **(**double**)**write\_count**/(**double**)(**total**)\***100**);**

fprintf**(**report**,** " Inst. =%13llu [%4.1f%%]\n"**,** instruction\_count**,** **(**double**)**instruction\_count**/(**double**)(**total**)\***100**);**

fprintf**(**report**,** " Total =%13llu\n\n"**,** total**);**

fprintf**(**report**,** " Total cycles for activities: [Percentage]\n"**);**

fprintf**(**report**,** " Reads =%13llu [%4.1f%%]\n"**,** read\_time**,** **(**double**)**read\_time**/(**double**)(**execution**)\***100**);**

fprintf**(**report**,** " Writes =%13llu [%4.1f%%]\n"**,** write\_time**,** **(**double**)**write\_time**/(**double**)(**execution**)\***100**);**

fprintf**(**report**,** " Inst. =%13llu [%4.1f%%]\n"**,** inst\_time**,** **(**double**)**inst\_time**/(**double**)(**execution**)\***100**);**

fprintf**(**report**,** " Total =%13llu\n\n"**,** execution**);**

fprintf**(**report**,** " CPI = %.1f\n"**,** **(**double**)(**execution**)/**instruction\_count**);**

fprintf**(**report**,** " Ideal: Exec. Time = %llu; CPI = %.1f\n"**,** total**+**instruction\_count**,** **(**double**)(**total**+**instruction\_count**)/**instruction\_count**);**

fprintf**(**report**,** " Ideal mis-aligned: Exec. Time = %llu; CPI = %.1f\n\n"**,** L1\_I**->**request\_count **+** L1\_D**->**request\_count **+** instruction\_count **,(**double**)(**L1\_I**->**request\_count **+** L1\_D**->**request\_count **+** instruction\_count**)/**instruction\_count**);**

fprintf**(**report**,** " Memory Level: L1i\n"**);**

fprintf**(**report**,** " Hit Count = %llu Miss Count = %llu\n"**,** L1\_I**->**hit\_count**,** L1\_I**->**miss\_count**);**

fprintf**(**report**,** " Total Requests = %llu\n"**,** L1\_I**->**request\_count**);**

fprintf**(**report**,** " Hit Rate = %.1f%% Miss Rate = %.1f%%\n"**,** **(**double**)(**L1\_I**->**hit\_count**)/(**double**)(**L1\_I**->**hit\_count**+**L1\_I**->**miss\_count**)\***100**,(**double**)(**L1\_I**->**miss\_count**)/(**double**)(**L1\_I**->**request\_count**)\***100**);**

fprintf**(**report**,** " Kickouts = %llu; Dirty kickouts = %llu; Transfers = %llu\n"**,** L1\_I**->**kickout\_count**,** L1\_I**->**dirty\_kickout**,** L1\_I**->**transfer\_count**);**

fprintf**(**report**,** " VC Hit count = %llu\n\n"**,** L1\_I**->**vc\_hit**);**

fprintf**(**report**,** " Memory Level: L1d\n"**);**

fprintf**(**report**,** " Hit Count = %llu Miss Count = %llu\n"**,** L1\_D**->**hit\_count**,** L1\_D**->**miss\_count**);**

fprintf**(**report**,** " Total Requests = %llu\n"**,** L1\_D**->**request\_count**);**

fprintf**(**report**,** " Hit Rate = %.1f%% Miss Rate = %.1f%%\n"**,** **(**double**)(**L1\_D**->**hit\_count**)/(**double**)(**L1\_D**->**hit\_count**+**L1\_D**->**miss\_count**)\***100**,(**double**)(**L1\_D**->**miss\_count**)/(**double**)(**L1\_D**->**request\_count**)\***100**);**

fprintf**(**report**,** " Kickouts = %llu; Dirty kickouts = %llu; Transfers = %llu\n"**,** L1\_D**->**kickout\_count**,** L1\_D**->**dirty\_kickout**,** L1\_D**->**transfer\_count**);**

fprintf**(**report**,** " VC Hit count = %llu\n\n"**,** L1\_D**->**vc\_hit**);**

fprintf**(**report**,** " Memory Level: L2\n"**);**

fprintf**(**report**,** " Hit Count = %llu Miss Count = %llu\n"**,** L2**->**hit\_count**,** L2**->**miss\_count**);**

fprintf**(**report**,** " Total Requests = %llu\n"**,** L2**->**request\_count**);**

fprintf**(**report**,** " Hit Rate = %.1f%% Miss Rate = %.1f%%\n"**,** **(**double**)(**L2**->**hit\_count**)/(**double**)(**L2**->**hit\_count**+**L2**->**miss\_count**)\***100**,(**double**)(**L2**->**miss\_count**)/(**double**)(**L2**->**request\_count**)\***100**);**

fprintf**(**report**,** " Kickouts = %llu; Dirty kickouts = %llu; Transfers = %llu\n"**,** L2**->**kickout\_count**,** L2**->**dirty\_kickout**,** L2**->**transfer\_count**);**

fprintf**(**report**,** " VC Hit count = %llu\n\n"**,** L2**->**vc\_hit**);**

fprintf**(**report**,** " L1 cache cost (Icache $%llu) + (Dcache $%llu) = $%llu\n"**,** L1Cost**,** L1Cost1**,** L1Cost**+**L1Cost1**);**

fprintf**(**report**,** " L2 cache cost = $%llu; Memory cost = $%llu Total cost = $%llu"**,** L2Cost**,** memCost**,** L1Cost**+**L2Cost**+**memCost**+**L1Cost1**);**

fclose**(**report**);**

**}**

**cache.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jeffery Lim

Taylor Nguyen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef CACHE\_H

#define CACHE\_H

#include "lru.h"

//We had this like this because set had more stuff and we reduced it a lot

struct set

**{**

lru**\*** blocks**;**

**};**

class cache

**{**

private**:**

unsigned int block\_size**;**

unsigned int cache\_size**;**

unsigned int indexes**;**

unsigned int ways**;**

unsigned int hit\_time**;**

unsigned int miss\_time**;**

unsigned int transfer\_time**;**

unsigned int bus\_width**;**

set**\*** cache\_index**;**

set**\*** victim\_cache**;**

cache**\*** transfer\_level**;**

public**:**

unsigned int get\_block\_size**();**

unsigned int get\_cache\_size**();**

unsigned int get\_indexes**();**

unsigned int get\_ways**();**

unsigned int get\_hit\_time**();**

unsigned int get\_miss\_time**();**

unsigned long long int get\_address**(**unsigned long long int tag**,** unsigned long long int index**);**

unsigned long long int get\_tag**(**unsigned long long int address**);**

unsigned long long int get\_index**(**unsigned long long int address**);**

unsigned long long int hit\_count**;**

unsigned long long int vc\_hit**;**

unsigned long long int miss\_count**;**

unsigned long long int request\_count**;**

unsigned long long int kickout\_count**;**

unsigned long long int dirty\_kickout**;**

unsigned long long int transfer\_count**;**

cache**();**

void cache\_deconstruct**();**

void set\_cache**(**unsigned int bs**,** unsigned int cs**,** unsigned int assoc**,** unsigned int ht**,** unsigned int mt**,** unsigned int tt**,** unsigned int bw**,** cache**\*** transfer**);**

unsigned long long int find**(**unsigned long long int address**,** char op**);**

void print**();**

**};**

#endif

**cache.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jeffery Lim

Taylor Nguyen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <math.h>

#include <iostream>

#include "cache.h"

**using** **namespace** std**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

cache::cache()

Initailizes the counters of the cache

The actual cache initialization is in set cache

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

cache**::**cache**(){**

//initializes the counters

hit\_count **=** 0**;**

miss\_count **=** 0**;**

request\_count **=** 0**;**

kickout\_count **=** 0**;**

dirty\_kickout **=** 0**;**

transfer\_count **=** 0**;**

vc\_hit **=** 0**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void cache::set\_cache(unsigned int bs, unsigned int cs, unsigned int assoc, unsigned int ht, unsigned int mt, unsigned int tt, unsigned int bw,cache\* transfer)

Initializes all the cache parameters

bs -> blocksize

cs -> cache size

assoc -> ways

ht -> hit time

mt -> miss time

tt -> transfer time

bw -> bus width

transfer level -> next level of memory

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void cache**::**set\_cache**(**unsigned int bs**,** unsigned int cs**,** unsigned int assoc**,** unsigned int ht**,** unsigned int mt**,** unsigned int tt**,** unsigned int bw**,**cache**\*** transfer**){**

block\_size **=** bs**;**

cache\_size **=** cs**;**

indexes **=** **(**cs**/**bs**/**assoc**);**

ways **=** assoc**;**

hit\_time **=** ht**;**

miss\_time **=** mt**;**

transfer\_time **=** tt**;**

bus\_width **=** bw**;**

transfer\_level **=** transfer**;**

//Sets memory for the LRU and set

cache\_index **=** **new** set**[**indexes**];**

**for** **(**unsigned int i **=** 0**;** i **<** indexes**;** i**++)**

**{**

cache\_index**[**i**].**blocks **=** **new** lru**(**ways**);**

**}**

//Sets memory for the victim cache

victim\_cache **=** **new** set**;**

victim\_cache**->**blocks **=** **new** lru**(**8**);**

**return;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void cache::cache\_deconstruct()

Deconstructs the cache

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void cache**::**cache\_deconstruct**(){**

**for(**unsigned int i **=** 0**;** i **<** indexes**;** i**++){**

cache\_index**[**i**].**blocks**->**lru\_deconstruct**();**

**}**

**delete** cache\_index**;**

victim\_cache**->**blocks**->**lru\_deconstruct**();**

**delete** victim\_cache**;**

**return;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned long long int cache::find(unsigned long long int address, char op)

Returns cycles

Finds the address in the cache and will either write or read

In this simulator, the only difference between write and read is that the write operation marks a block dirty

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long long int cache**::**find**(**unsigned long long int address**,** char op**){**

unsigned long long int execution **=** 0**;**

unsigned long long int tag **=** 0**;**

unsigned long long int vc\_tag **=** 0**;**

unsigned long long int temp **=** 0**;**

unsigned long int index **=** 0**;**

//get the tag and index from address and cache parameters

tag **=** get\_tag**(**address**);**

index **=** get\_index**(**address**);**

node**\*** c\_ptr**;** //cache pointer

node**\*** v\_ptr**;** //victim pointer

request\_count**++;**

c\_ptr **=** cache\_index**[**index**].**blocks**->**get\_head**();** //get the head of cache

//Look for the specific node with the right tag in index

**while(**c\_ptr**->**next **!=** **NULL){**

c\_ptr **=** c\_ptr**->**next**;**

**if(**c\_ptr**->**valid **&&** c\_ptr**->**tag **==** tag**){** //Check the node's valid and tag

//Update, Hit, return

cache\_index**[**index**].**blocks**->**update**(**c\_ptr**);**

hit\_count**++;**

execution **+=** hit\_time**;**

**if(**op **==** 'W'**){**

c\_ptr**->**dirty **=** 1**;**

**}**

**return** execution**;**

**}**

**}**

//Miss

miss\_count**++;**

execution **+=** miss\_time**;**

vc\_tag **=** get\_address**(**tag**,** index**);** //Get the address from the tag

v\_ptr **=** victim\_cache**->**blocks**->**get\_head**();** //Get the head of the victim cache

**while(**v\_ptr**->**next **!=** **NULL){**

v\_ptr **=** v\_ptr**->**next**;**

**if(**v\_ptr**->**valid **&&** v\_ptr**->**tag **==** vc\_tag**){**

//VC Hit

c\_ptr **=** cache\_index**[**index**].**blocks**->**get\_tail**();**

vc\_hit**++;**

execution **+=** hit\_time**;**

**if(**op **==** 'W'**){**

v\_ptr**->**dirty **=** 1**;**

**}**

//Swap the values of the victim hit and the cache eviction

temp **=** v\_ptr**->**dirty**;**

v\_ptr**->**dirty **=** c\_ptr**->**dirty**;**

c\_ptr**->**dirty **=** temp**;**

temp **=** v\_ptr**->**valid**;**

v\_ptr**->**valid **=** c\_ptr**->**valid**;**

c\_ptr**->**valid **=** temp**;**

temp **=** v\_ptr**->**tag**;**

v\_ptr**->**tag **=** get\_address**(**c\_ptr**->**tag**,** index**);**

c\_ptr**->**tag **=** get\_tag**(**temp**);**

cache\_index**[**index**].**blocks**->**update**(**c\_ptr**);**

victim\_cache**->**blocks**->**update**(**v\_ptr**);**

**return** execution**;**

**}**

**}**

v\_ptr **=** victim\_cache**->**blocks**->**get\_tail**();** //Get the victim set and the cache set tail

c\_ptr **=** cache\_index**[**index**].**blocks**->**get\_tail**();**

**if(**c\_ptr**->**valid **==** 1**){** //Check if the victim cache is full i.e tail is valid

**if(**v\_ptr**->**valid **==** 1**){** //Check if the cache is full i.e tail is valid

kickout\_count**++;**

**if(**v\_ptr**->**dirty **==** 1**){** //If the victim eviction is dirty, we need to perform dirty kickout

dirty\_kickout**++;**

**if(**transfer\_level **!=** **NULL){** //Find the address from next level of memory

execution **+=** transfer\_level**->**find**(**v\_ptr**->**tag**,** 'W'**);** //v\_ptr->tag is full address

execution **+=** transfer\_time**\*(**block\_size**/**bus\_width**);**

**}else** **if(**transfer\_level **==** **NULL){** //Go to main memory

execution **+=** 10**+**50**+(**15**\***64**/**bus\_width**);**

**}**

**}**

**}**

//Insert the LRU Cache block

v\_ptr**->**tag **=** get\_address**(**c\_ptr**->**tag**,**index**);**

v\_ptr**->**valid **=** c\_ptr**->**valid**;**

v\_ptr**->**dirty **=** c\_ptr**->**dirty**;**

victim\_cache**->**blocks**->**update**(**v\_ptr**);**

**}**

transfer\_count**++;**

**if(**transfer\_level **!=** **NULL){** //Go to the next level cache to retrieve data

execution**+=** transfer\_level**->**find**(**address**,** 'R'**);**

execution**+=** transfer\_time**\*(**block\_size**/**bus\_width**);**

**}else** **if(**transfer\_level **==** **NULL){**

execution **+=** 10**+**50**+(**15**\***64**/**bus\_width**);**

**}**

//Insert the new data into the main cache

c\_ptr**->**tag **=** tag**;**

c\_ptr**->**valid **=** 1**;**

c\_ptr**->**dirty **=** 0**;**

**if(**op **==** 'W'**){**

c\_ptr**->**dirty **=** 1**;**

**}**

cache\_index**[**index**].**blocks**->**update**(**c\_ptr**);**

execution **+=** hit\_time**;**

**return** execution**;**

**}**

unsigned int cache**::**get\_block\_size**(){**

**return** block\_size**;**

**}**

unsigned int cache**::**get\_cache\_size**(){**

**return** cache\_size**;**

**}**

unsigned int cache**::**get\_indexes**(){**

**return** indexes**;**

**}**

unsigned int cache**::**get\_ways**(){**

**return** ways**;**

**}**

unsigned int cache**::**get\_hit\_time**(){**

**return** hit\_time**;**

**}**

unsigned int cache**::**get\_miss\_time**(){**

**return** miss\_time**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned long long int get\_index(unsigned long long int address)

Returns the index from the address

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long long int cache**::**get\_index**(**unsigned long long int address**){**

unsigned long long int temp**;**

unsigned long long int index**;**

temp **=** 64 **-** log2**(**get\_indexes**())-**log2**(**get\_block\_size**());** //Gives us the size of tag

index **=** address **<<** temp**;** //Zero out the top bits

index **=** index **>>** temp**;**

temp **=** log2**(**get\_block\_size**());** //Shifting it by log2(bytesize) gives us the index size

index **=** index **>>** temp**;**

**return** index**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned long long int get\_tag(unsigned long long int address)

Returns the tag from the address

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long long int cache**::**get\_tag**(**unsigned long long int address**){**

unsigned long long int temp**;**

unsigned long long int tag**;**

temp **=** log2**(**get\_block\_size**());** //Shift it right to get rid of the byteoffset

tag **=** address **>>** temp**;**

temp **=** log2**(**get\_indexes**());** //Shift it right to get rid of index

tag **=** tag **>>** temp**;**

**return** tag**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned long long int get\_address(unsigned long long int tag, unsigned long long int index){

Reconstructs the address from the tag and index

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned long long int cache**::**get\_address**(**unsigned long long int tag**,** unsigned long long int index**){**

unsigned long long int temp**;**

unsigned long long int address **=** 0**;**

temp **=** log2**(**get\_indexes**());**

address **=** tag **<<** temp**;** //Shift tag by index size

address **=** address **|** index**;** //Or in the index so its Tag Index

temp **=** log2**(**get\_block\_size**());** //Shift it left by byte offset size

address **=** address **<<** temp**;**

**return** address**;**

**}**

unsigned long long int log2**(**unsigned long long int number**){**

**return** **(**unsigned int**)**round**(**log**(**number**)/**log**(**2**));**

**}**

void cache**::**print**(){**

unsigned int i**;**

node **\*** temp**;**

**for(**i **=** 0**;** i **<** indexes**;** i**++){**

temp **=** cache\_index**[**i**].**blocks**->**get\_head**();**

**while(**temp**->**next **!=** **NULL){**

temp **=** temp**->**next**;**

cout **<<** hex **<<** i **<<** ": " **<<** "V: " **<<** temp**->**valid **<<** " D: " **<<** temp**->**dirty **<<** " tag: " **<<**hex **<<** temp**->**tag **<<** " | "**;**

**}**

cout **<<** endl**;**

**}**

cout **<<** endl**;**

cout **<<** "\nVictim Cache: \n"**<<** endl**;**

temp **=** victim\_cache**->**blocks**->**get\_head**();**

i **=** 0**;**

**while(**temp**->**next **!=** **NULL){**

temp **=** temp**->**next**;**

cout **<<** hex **<<** i **<<** ": " **<<** "V: " **<<** temp**->**valid **<<** " D: " **<<** temp**->**dirty **<<** " Addr: " **<<**hex **<<** temp**->**tag **<<** endl**;**

i**++;**

**}**

**}**

**lru.h**

#ifndef LRU\_H

#define LRU\_H

struct node **{**

node**\*** next**;**

node**\*** prev**;**

bool dirty**;**

bool valid**;**

unsigned long long int tag**;**

**};**

class lru **{**

private**:**

struct node**\*** head**;**

struct node**\*** tail**;**

public**:**

lru**();**

lru**(**unsigned int way**);**

void lru\_deconstruct**();**

node**\*** get\_tail**();**

node**\*** get\_head**();**

void update**(**node**\*** ptr**);**

void print**();**

**};**

#endif

**lru.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jeffery Lim

Taylor Nguyen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <stdio.h>

#include "lru.h"

**using** **namespace** std**;**

lru**::**lru**(){**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lru::lru(unsigned int way)

Initalizes the lru with way number of nodes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

lru**::**lru**(**unsigned int way**){**

node**\*** temp **=** **new** node**;**

head **=** temp**;**

node**\*** temp1 **=** temp**;**

**for(**unsigned int i **=** 0**;** i **<** way**;** i**++){**

node**\*** entry **=** **new** node**;**

temp**->**next **=** entry**;**

temp **=** temp**->**next**;**

temp**->**prev **=** temp1**;**

temp**->**tag **=** 0**;**

temp**->**dirty **=** 0**;**

temp**->**valid **=** 0**;**

temp1 **=** temp**;**

**}**

tail **=** temp**;**

temp**->**next **=** **NULL;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void lru::lru\_deconstruct()

Deconstructs the LRU

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void lru**::**lru\_deconstruct**(){**

node**\***temp **=** head**;**

**while(**head **!=** **NULL){**

head **=** head**->**next**;**

**delete** temp**;**

temp **=** head**;**

**}**

**delete** temp**;**

**return;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

node\* lru::get\_tail()

Returns the tail

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

node**\*** lru**::**get\_tail**(){**

**return** tail**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

node\* lru::get\_head()

Returns the head

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

node**\*** lru**::**get\_head**(){**

**return** head**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void lru::update(node\* ptr)

Updates the linked list by bringing the ptr to the head of the list

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void lru**::**update**(**node**\*** ptr**){**

**if(**ptr**->**valid **==** 0 **||** head**->**next **==** ptr**){**

**return;**

**}**

node**\*** temp **=** head**;**

temp **=** head**->**next**;**

**(**ptr**->**prev**)->**next **=** ptr**->**next**;**

**if(**tail **==** ptr**){**

tail **=** **(**ptr**->**prev**);**

**}else{**

**(**ptr**->**next**)->**prev **=** ptr**->**prev**;**

**}**

temp**->**prev **=** ptr**;**

ptr**->**next **=** temp**;**

ptr**->**prev **=** **NULL;**

head**->**next **=** ptr**;**

**return;**

**}**

void lru**::**print**(){**

node**\*** temp **=** head**;**

int i **=** 0**;**

**while(**temp**->**next **!=** **NULL** **){**

temp **=** temp**->**next**;**

cout **<<** hex **<<** i **<<** ": " **<<** "Valid: " **<<** temp**->**valid **<<** " Dirty: " **<<** temp**->**dirty **<<** " Tag: " **<<**hex **<<** temp**->**tag **<<** endl**;**

i**++;**

**}**

**if(**i **==** 0**){**

cout **<<** hex **<<** i **<<** ": " **<<** "Valid: " **<<** temp**->**valid **<<** " Dirty: " **<<** temp**->**dirty **<<** " Tag: " **<<**hex **<<** temp**->**tag **<<** endl**;**

**}**

cout **<<** endl**;**

**}**