Homework 6

Poopy Cache

#include <stdio.h>

#include <math.h>

//test\_set\_1 is an array of ints that represents the sequence of

//addresses that the CPU will use to test the cache emulator

int test\_set\_1[] = {0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44,

48, 52, 56, 60, 64, 68, 72, 76, 80, 0, 4, 8,

12, 16, 71, 3, 41, 81, 39, 38, 71, 15, 39, 11,

51, 57, 41};

//Parameters s, l, and n

//s represents the size of the cache in bytes

int s;

//l represents the size of the lines in the cache, in bytes

int l;

//n represents the size of the sets in the cache - so a direct mapped

//cache (in which each line is a set) will have n = 1, while a fully

//associate cache (in which the entire cache is one set) will have

//n = s/l (the number of lines in the cache). Furthermore, a cache

//broken up into sets of 4 lines each will have n = s/4\*l

int n;

//a row is a struct containing an int valid, an int tag, and a pointer

//to a previous row (allowing for interactions similar to linked-lists)

struct row{

//int valid will be either a 0 or a 1, indicating whether or not

//this row of the cache has been accessed.

int valid;

//if the address specified is greater than the size of the cache,

//tag will increase by 1 for each time the index is set back to 0

//in order to reach the desired address.

//For example: if the address given is 20 and the number of rows in

//the cache is only 16, the index that the cache will attempt to

//read from will be 4, with a tag of 1. Similarly if the address

//given is 38, the index will be 6, with a tag of 2.

int tag;

//\*prev is a pointer to the previous row. This will aid in FIFO

//overwriting of data stored in the cache.

struct row \*prev;

};

//a cache is an Array of rows

struct row cache[100];

void initialize(){

//Ask user for inputs s, l, and n.

printf("input the size of this cache, in bytes: ");

scanf("%d", &s);

printf("input the size of one line in this cache, in bytes: ");

scanf("%d", &l);

printf("input the size of each set in this cache: ");

scanf("%d", &n);

//caculate the number of rows in the cache

int numRows = s/l;

//initialize the cache

int i;

for(i = 0; i < numRows; i++){

cache[i].valid = 0;

cache[i].tag = 0;

cache[i].prev = NULL;

}

}

int loadByte(int address){

int offset = address % l;

int index = floor(address % (s/l));

int tag = floor(address / (s/l));

//if the index given has been touched and has the

//same tag

if(cache[index].tag == tag && cache[index].valid == 1){

return 1;

//if the index given has be touched, but by an address

//with a different tag

} else if(!cache[index].tag == tag && cache[index].valid == 1){

cache[index].tag = tag;

return 0;

//if(cache[index].valid == 0)

//if the index given has not been touched in the past

} else {

cache[index].valid = 1;

cache[index].tag = tag;

return 0;

}

}

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

initialize();

int i;

int size = sizeof(test\_set\_1)/sizeof(int);

printf("%d\n", size);

for(i = 0; i < size; i++){

if(loadByte(test\_set\_1[i]) == 1){

printf("%d, Hit\n", test\_set\_1[i]);

}

//if(loadByte(test\_set\_1[i]) == 0)

else{

printf("%d, Miss\n", test\_set\_1[i]);

}

}

system("pause");

return 0;

}