Project 6: Cache Memory

**Due** Friday by 11:59pm | **Points** 50 | **Submitting** a file upload | **File Types** c | **Available** May 1 at 12am - May 6 at 11:59pm 6 days

**Project 5: Assembly Language Programming**

This project simulates a read-only L1 cache.

**Cache specifications**

* Read only - like an instruction cache. (It's bad to allow users to write to the instructions)
* This cache has 8 sets with 1 line each, and 32 bytes of data per line.
* Please find the struct defining this cache at the top of cache.c

**Getting Started**

Watch the lecture videos about cache memories & Read Section 6.1-6.4 of the CSAPP book.

**Project Specification**

* Complete the function in cache.c that reads a single character from the cache.  If the data in the cache is not valid then the cache goes to the next level down and reads the data into the cache (a function to put the data on the system bus is provided).
* Write additional tests in main. Only one very simple test is provided to demonstrate how to use the cache.
* You may find it useful to write a function to print the entire contents of the cache for debugging.
* A struct describing the cache is included for you at the top of the file.  Do not modify these structs.
* Global counters track cache hits and misses.  Increment these in your code to collect performance data.
* We will be testing your function with a separate main function and text file. We will be testing with the struct and the Read\_Data\_From\_Ram function provided.
* Caches are built into hardware and not changed after manufactured.  You may hardcode any data pertaining cache sizes.

**Hints**

* bit shift, bitwise 'and', integer division and modulus operators can be really useful for extracting the tag, set, and byte data.

**Files**

* [cache\_template.c](https://canvas.wisc.edu/courses/280031/files/26017993?wrap=1) - rename this to cache.c
* [alice\_in\_wonderland.txt](https://canvas.wisc.edu/courses/280031/files/26017995?wrap=1)

**Turn in**

Upload your cache.c file to Canvas

You may submit as many times as you wish

Make sure you do your own work - we use Moss to identify academic misconduct

#include <stdio.h>

// THE CACHE

struct LINE{

char valid;

int tag;

char data[32];

};

struct SET{

struct LINE line[1];

};

struct CACHE {

struct SET set[8];

} cache;

// GLOBALDATA

// increment these in your function

unsigned hit\_count = 0;

unsigned miss\_count = 0;

unsigned read\_data\_count = 0;

// SYSTEM BUS

// call Read\_Data\_From\_Ram to update this

char system\_bus[32];

// READ DATA FROM RAM

// this function copies 32 character from the text file to

// the system\_bus array aligned with the system bus

// (we will not test this with input larger than the text file)

void Read\_Data\_From\_Ram(unsigned address) {

address >>= 5; address <<= 5; // get align the data

read\_data\_count++;

FILE \*file;

file = fopen("alice\_in\_wonderland.txt", "r");

fseek(file, address, SEEK\_SET);

for (int i = 0; i<32; i++) system\_bus[i] = fgetc(file);

return;

}

// COMPLETE THE FOLLOWING FUNCTION

// this code should check to see if the data in the cache is valid and the tag matches

// if so it should return the char matching the address and increment the hit count

// if not it should read the data from the system bus copy it into the cache and

// increment the miss count and return the char matching the address

char Read\_Data\_From\_Cache(unsigned address){

// write your code here

// replace the hardcoded 'a' below

return 'a';}

int main() {

// INITIALIZE CACHE

for (int i = 0; i<8; i++) cache.set[i].line[0].valid = 0;

// READ SOME DATA

char c;

c = Read\_Data\_From\_Cache(0); printf("data = %c : hit count = %-3u : miss count = %-3u : read data count = %-3u\n", c, hit\_count, miss\_count,read\_data\_count );

c = Read\_Data\_From\_Cache(1); printf("data = %c : hit count = %-3u : miss count = %-3u : read data count = %-3u\n", c, hit\_count, miss\_count,read\_data\_count );

c = Read\_Data\_From\_Cache(2); printf("data = %c : hit count = %-3u : miss count = %-3u : read data count = %-3u\n", c, hit\_count, miss\_count,read\_data\_count );

c = Read\_Data\_From\_Cache(3); printf("data = %c : hit count = %-3u : miss count = %-3u : read data count = %-3u\n", c, hit\_count, miss\_count,read\_data\_count );

c = Read\_Data\_From\_Cache(4); printf("data = %c : hit count = %-3u : miss count = %-3u : read data count = %-3u\n", c, hit\_count, miss\_count,read\_data\_count );

// WRITE A LOT MORE TESTS

return 0;

}