Memory Hierarchy Simulator

Ramy Shehata-900222862

Omar Ganna-900222646

Mohamed Khaled-900225303

Spring 2024

Department of Computer Science and Engineering

The American University in Cairo

Table of Contents

1- Introduction

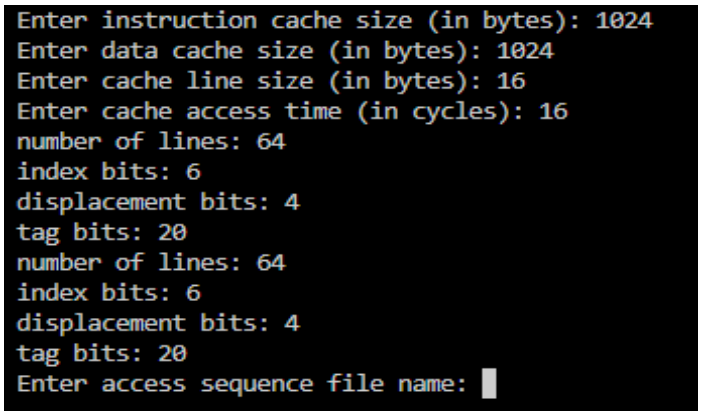
This report will discuss the implementation of a Memory Hierarchy Simulator on C++ and aims to emulate how a memory works in terms of direct mapping only. From the cache to the memory used, our simulator tries to emulate how memory simulator does hits, misses, and calculates the AMT. The report will also talk about the design choices we took while designing, such as the number of Bytes taken It will also include step by step guidelines on how to run the simulator, what is the needed code and its syntax for the simulator to run properly. Lastly, we will include around 2 sample sequence programs we used to showcase how our simulator works and what is the output.

2- Simulator Implementation and Design

We will first talk about the implementations we used for the simulator and then dive into the design choices we have for our simulator.

a- The Implementation and Code

Firstly, C++ was the main programming language we used as it was the most familiar language. Secondly, to classify everything, we used a class called Cache, which contains all the actual heart of our code. It uses a variety of data structures, mainly a struct for the valid bit and tag and vector to simulate a cache line. Thirdly, for our program flow, which is in Run.cpp, when the program is executed, it asks the user for the following things:



After that, it parses the access sequence and splits it depending on if the access sequence is a data or instruction. After the parsing sequence, the program will then loop across the instruction cache, then the data cache and output after each hit/miss.

Lastly, we implemented 1 bonus feature for our program which is the splitting of data and instructions in the cache. For more information about the code itself, the code is well documented and has comments on everything.

b-Design of Our Simulator

Throughout the process of doing the simulator, we employed many different designs that may differ from other simulators out there.

The first design choice is that all the cache information such as the cache size, line size, should be in powers of 2. The second design choice is the use of a specific range for the cycles for the cache memory access, where the user will put from 1 to 9 only. This is to ensure that the cache has a fast access unlike the memory which is assumed in this code to be 150 cycles and cannot be changed. The third design choice is the usage of the 30-bit address and lastly, the fourth design is that our simulator only doesn't care about the data in the memory, it only simulates the address of the memory, not the actual content itself.

3- Simulator Usage Guide

First, to run the simulator, you must ensure the access sequence is written like the following format:

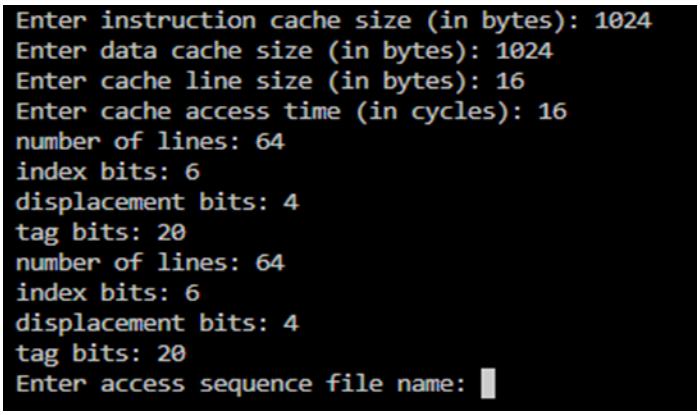
<binarycode>, (D or I)

Where D represents Data and I→ instruction

Below is a sample of how it is written:

```
00000000000001111000100100000,D
00000000001100000010101000011,D
00000000000001110110010111100,I
00000000000001111000100100000,D
00000000000011011000111010100 1,I
00000000000011011000111010100 1,I
00000000001100000010101000011,D
00000000000001111000100100001,D
00000000000001111000100100000,D
00000000000001111000100100010,I
00000000001100000010101000100,D
00000000001100000010101000101,D
00000000001100000010101000011,I
00000000001100000010101000110,D
00000000000001110110010111110,I
00000000000000001010110001010,I
00000000000000001010110001011,D
00000000000011110101011000 1010,I
00000000000001111000100100000,I
00000000000000000010101001 0101,I
00000000000000000010101011 0110,D
00000000000000000010101011 0110,D
00000000000000000010101011 0110,I
00000000000000000010101011 0110,I
```

After that, navigate to the source files folder where you will run the cpp file Run.cpp. After that it will ask you the following questions:



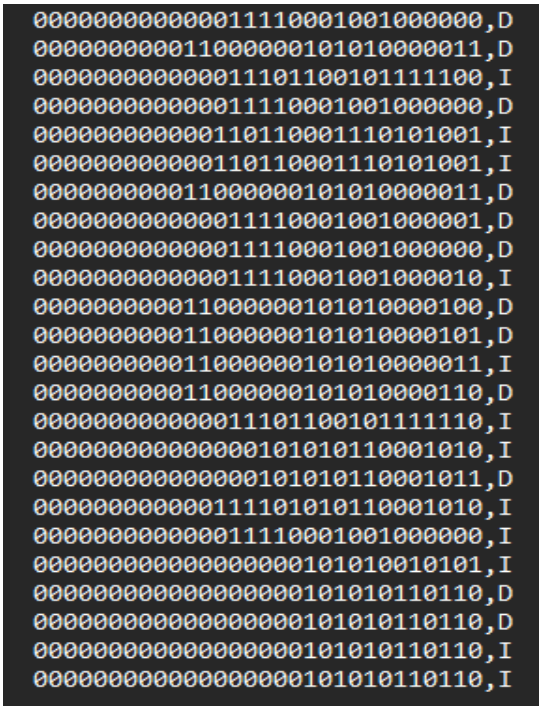
And it will showcase the no. of lines, the index bits and the displacementr bits for each cache.

After that, the user will put the path for the sequence and voila, the sequence will bne outputted where each sequence happen for each cache and then the final cache status will also be outputted

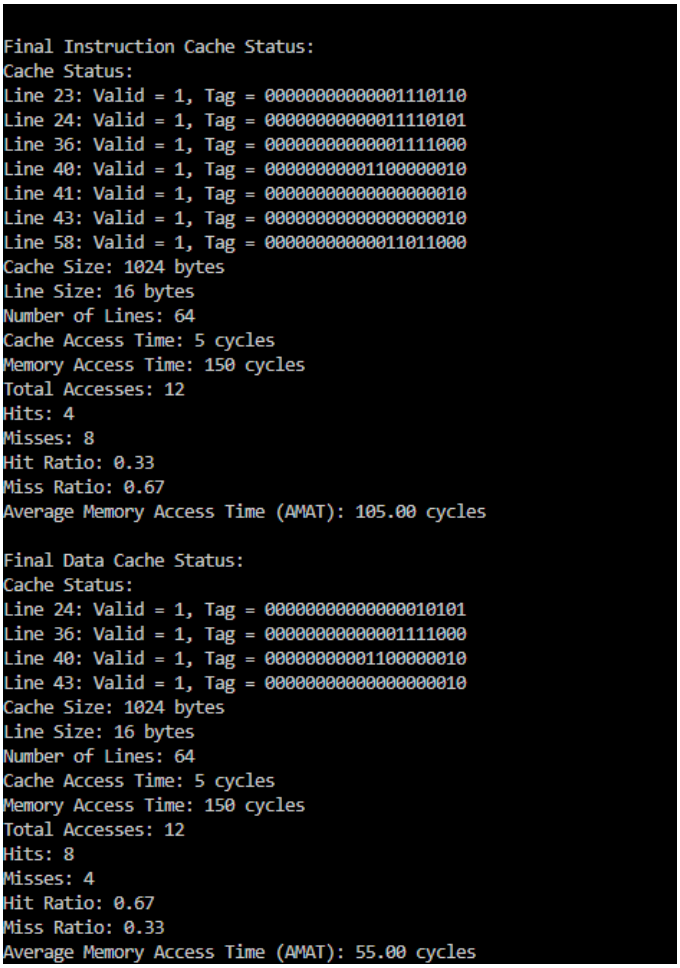
4- Program testing

During our testing phase, we tested 2 different sequences to ensure that our simulator can handle different codes.

This is the sample of the first sequence



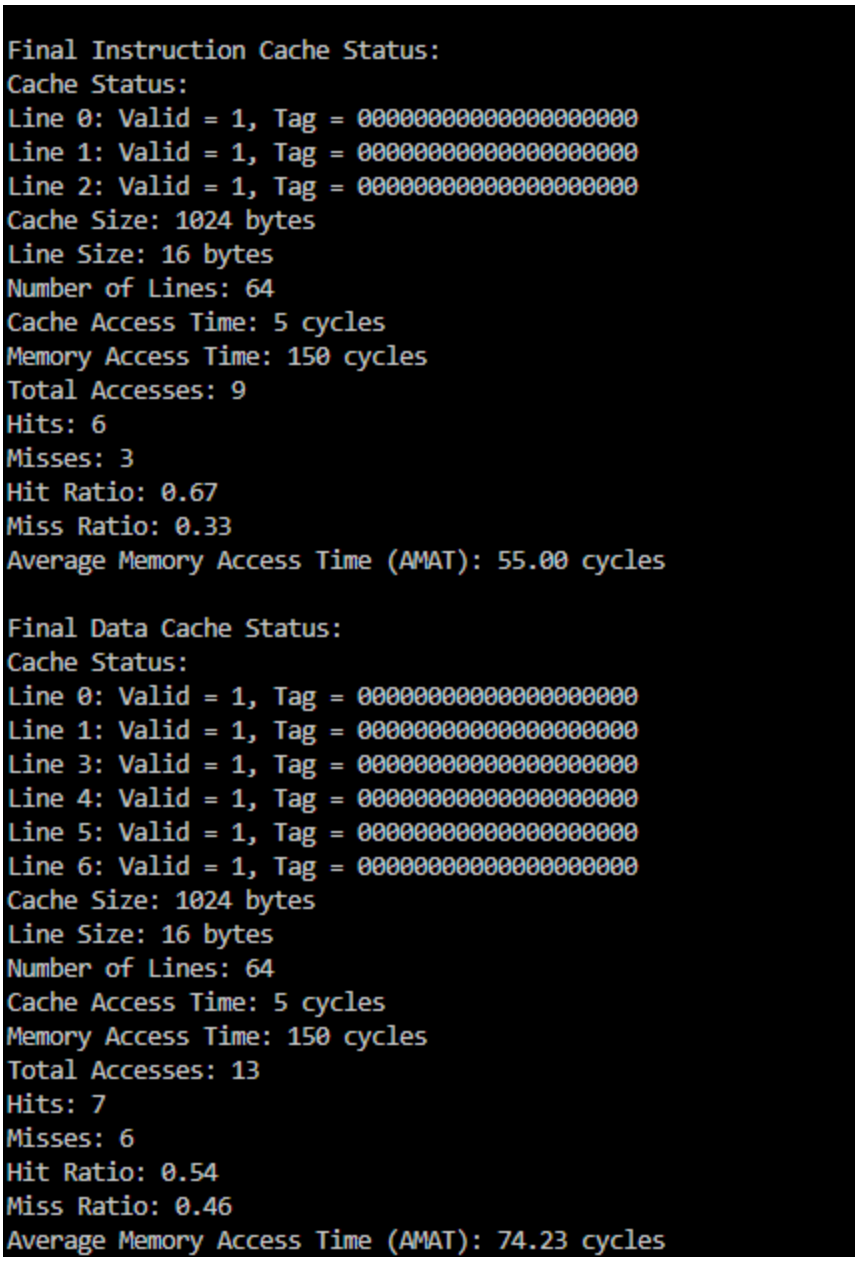
And the following is the final output assuming cache size for both =1024, and cache line =16 bytes and cycle of cache =5.



For the second input and output assuming same cache size and cycles.

```
0000000000001101, D
0000000000000111,I
0000000000011010,D
0000000001011100,D
0000000000001111,I
0000000000001101,D
0000000000000111, I
0000000000110010, D
0000000000010111, I
0000000000011010,D
0000000001100110,D
0000000000001111,I
0000000001001011,D
0000000000001101,D
0000000000000111,I
0000000000011010,D
0000000000101110,I
0000000000110010,D
0000000000010111,I
0000000001011100,D
0000000000001111,I
0000000001100110,D
```

Output:



5- Known Bugs and Issues:

With the current testing we have done, there seems to be no known issues or bugs that we have encountered thus far. However, there is always room for improvement. We could have added support for set and full associativity as well as write policies, however due to time constraints we couldn't implement them.

Thank you for reading the report. If you have any questions feel free to ask us