



The American University in Cairo

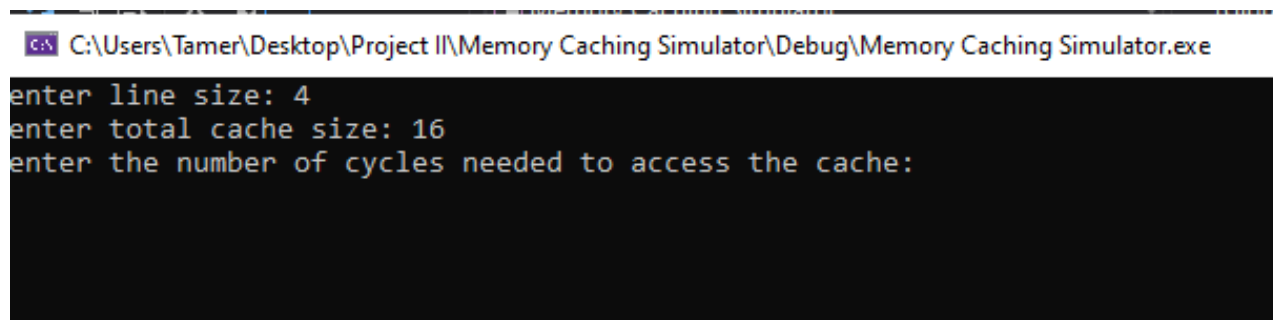
Computer Science and Engineering Department

Memory Hierarchy Simulator Project Report

| | |
|------------------|-------------|
| Mina Mikhael | - 900196040 |
| John William | - 900193002 |
| Tamer Abdelfatah | - 900192103 |

User Guide

After downloading the Zip file, the user can press on the .exe file to run the program or run the cpp file in any IDE of their choice. At the start of the program, the user is prompted to enter some characteristics of the memory cache they would like to simulate. Namely, the cache line size and the total cache size; both of which need to be powers of 2 equivalent to 2^x . In addition, they enter the number of clock cycles that is needed to access the cache which needs to be a number between 1 and 10.

A screenshot of a Windows command prompt window titled "C:\Users\Tamer\Desktop\Project II\Memory Caching Simulator\Debug\Memory Caching Simulator.exe". The window has a black background with green text. It displays three prompts: "enter line size: 4", "enter total cache size: 16", and "enter the number of cycles needed to access the cache:".

```
C:\Users\Tamer\Desktop\Project II\Memory Caching Simulator\Debug\Memory Caching Simulator.exe
enter line size: 4
enter total cache size: 16
enter the number of cycles needed to access the cache:
```

For the second part of the input, this program requires a text file containing a sequence of memory addresses to simulate a program accessing parts of the memory. Each memory address is expected to be in a 32 bit binary format and to be separated by a line end. Since this simulator supports separate caches for instructions and data, the input text file needs to indicate whether the address sequence is for the data or instructions cache. This is done by adding either i or d after the binary number with no spaces or commas as shown in the picture below.

[illegible]

Following successfully inputting the data required, the program will output the hit rate and miss rate after each memory access. It is also going to output the new valid bits of all cache, the new tag, the number of accesses and the new Average Memory Access Time (AMAT). Beside each output section, it is indicated whether this is for the data cache or the instructions cache. A picture is shown below for clarification.

```

                                this is the instructions cache
00: valid bit is: 1, tag is: 000000000000000000000000000000
01: valid bit is: 0
10: valid bit is: 0
11: valid bit is: 0
The number of accesses is 1
The hit rate is 0
The miss rate is 1
The AMAT is 103

                                this is the data cache
00: valid bit is: 1, tag is: 000000000000000000000000000000
01: valid bit is: 0
10: valid bit is: 0
11: valid bit is: 0
The number of accesses is 1
The hit rate is 0
The miss rate is 1
The AMAT is 103

                                this is the instructions cache
00: valid bit is: 1, tag is: 000000000000000000000000000000
01: valid bit is: 1, tag is: 000000000000000000000000000001
10: valid bit is: 0
11: valid bit is: 0
The number of accesses is 2
The hit rate is 0
The miss rate is 1
The AMAT is 103

                                this is the data cache
00: valid bit is: 1, tag is: 000000000000000000000000000000
01: valid bit is: 0

```

Implementation

The cache simulator is implemented in C++ and has multiple functions that executes the sequence of addresses given to be addressed.

Cache: The cache is implemented using a map data structure that maps the index(block number) to the tag in it.

Addresses: A string that contains all the information needed for the cpu to access the cache and the memory.

The program is implemented in 5 functions besides the main function:

decToBinary: A function that outputs a given decimal number as a binary string.

getParts: A function that takes an address that needs to be accessed and then splits it into tag, index and offset for further processing.

accessCache: A function where the cache accessing and mapping happens. It checks if the valid bit is one or zero and if the tag in the needed block is the same and acts accordingly by updating the cache and the values of the accessing sequence. It also takes care whether the instruction is d instruction or i instruction.

Print: prints the current state of the cache and the status of the sequences being processed.

Main: reads the input, calls functions for processing and outputs the results.

Bonus: Supporting separate caches for data and instructions

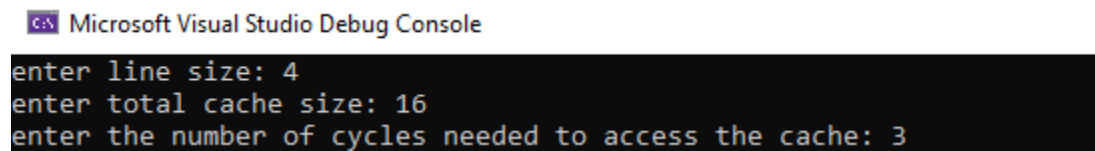
Assumptions

There are no new lines after the final address in the addresses text file.

The file only contains the addresses in binary followed by either i or d indicating whether it is for the data cache or instruction cache and addresses are separated by new lines.

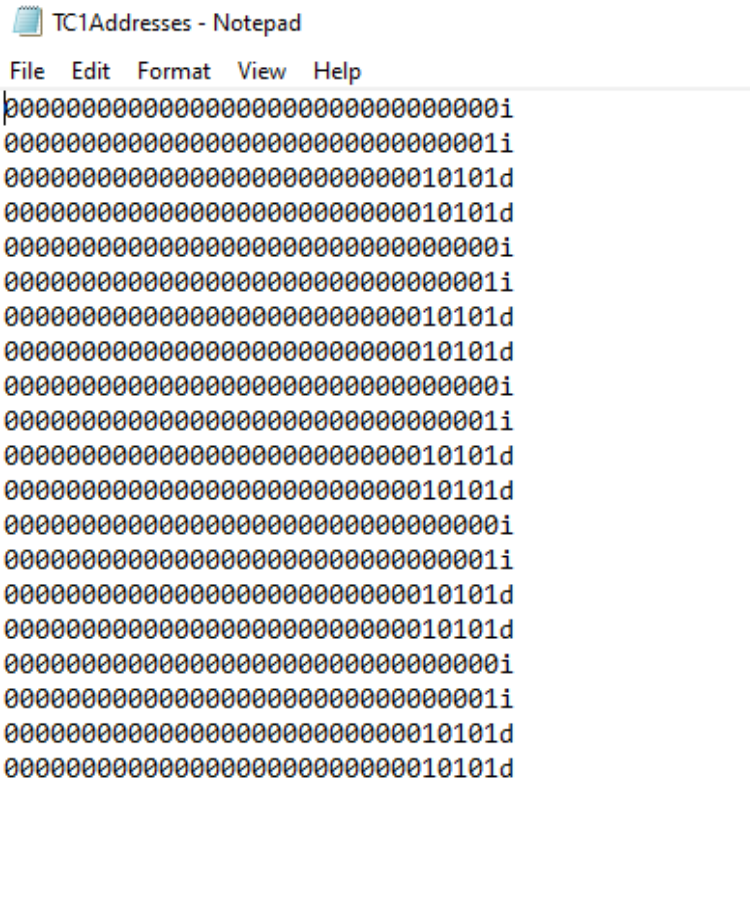
Test Case 1

The input for the line size is 4 bytes, the total cache size is 16 bytes and the number of cycles to access the cache is 3 cycles.

A screenshot of the Microsoft Visual Studio Debug Console. The title bar at the top reads "Microsoft Visual Studio Debug Console". The console area has a black background with white text. It displays three lines of input: "enter line size: 4", "enter total cache size: 16", and "enter the number of cycles needed to access the cache: 3".

```
Microsoft Visual Studio Debug Console  
enter line size: 4  
enter total cache size: 16  
enter the number of cycles needed to access the cache: 3
```

This test case contains 20 addresses as shown in the picture below. This text file is attached named TC1Addresses.txt



The program outputs the new data after each access. The final output for this input for data and instructions caches is

```
                                this is the instructions cache
00: valid bit is: 1, tag is: 000000000000000000000000000000
01: valid bit is: 0
10: valid bit is: 0
11: valid bit is: 0
The number of accesses is 10
The hit rate is 0.9
The miss rate is 0.1
The AMAT is 13

                                this is the data cache
00: valid bit is: 0
01: valid bit is: 1, tag is: 000000000000000000000000000001
10: valid bit is: 0
11: valid bit is: 0
The number of accesses is 10
The hit rate is 0.9
The miss rate is 0.1
The AMAT is 13
```

Test Case 2

The input for the line size is 4 bytes, the total cache size is 16 bytes and the number of cycles to access the cache is 5 cycles.

```
Microsoft Visual Studio Debug Console

enter line size: 4
enter total cache size: 16
enter the number of cycles needed to access the cache: 5
```

This test case contains 20 addresses as shown in the picture below. This text file is attached named TC2Adresses.txt


```
TC2Addresses - Notepad
File Edit Format View Help
00000000000000000000000000000000i
00000000000000000000000000000001d
000000000000000000000000000000010101i
000000000000000000000000000000010101d
00000000000000000000000000000000i
00000000000000000000000000000001d
000000000000000000000000000000010101i
000000000000000000000000000000010101d
00000000000000000000000000000000i
00000000000000000000000000000001d
000000000000000000000000000000010101i
000000000000000000000000000000010101d
00000000000000000000000000000000i
00000000000000000000000000000001d
000000000000000000000000000000010101i
000000000000000000000000000000010101d
00000000000000000000000000000000i
00000000000000000000000000000001d
000000000000000000000000000000010101i
000000000000000000000000000000010101d
```

The program outputs the new data after each access. The final output for this input for data and instructions caches is

```
                                this is the instructions cache
00: valid bit is: 1, tag is: 000000000000000000000000000000
01: valid bit is: 1, tag is: 000000000000000000000000000001
10: valid bit is: 0
11: valid bit is: 0
The number of accesses is 10
The hit rate is 0.8
The miss rate is 0.2
The AMAT is 25

                                this is the data cache
00: valid bit is: 1, tag is: 000000000000000000000000000000
01: valid bit is: 1, tag is: 000000000000000000000000000001
10: valid bit is: 0
11: valid bit is: 0
The number of accesses is 11
The hit rate is 0.818182
The miss rate is 0.181818
The AMAT is 23.1818
```

To try out a different test case, the user can change the input file in line 243 from TC1Addresses.txt to TC2Addresses.txt