CSCE 2303
Section 02
Dr. Cherif Salama

Habiba Seif (habibaseif@aucegypt.edu) - 900211308
Farida Ragheb (faridaragheb@aucegypt.edu) - 900225892
Haya Shalaby (hayashalaby@aucegypt.edu) - 900222031

Project Two Report

Implementation

Our cache simulator is a one-file C++ program making use of multiple data structures and procedures. The cache structure is a vector of cacheLine where cacheLine is a struct with the member variables tag and VB which are both initialized to -1 and false respectively since the cache is initially empty. We store the memory addresses (accesses) from the user-input text files into string vectors, specifically into two separate string vectors, one for data and another for instructions, to accommodate our implementation of allowing for separate instruction and data caches. The text files are read into the program using a simple helper readFile method. In order to also accommodate our multi-level cache design, and since we allow for the user to enter different cache sizes, cache line sizes, and cache access times for each level and each category (data or instructions), we created int vectors for each value, using the index to track which value belongs to which level (e.g. index = 0 is for level 1). These vectors are also split into ones for data and others for instructions to keep things organized and easy to trace/track. The same goes for values such as number of hits and misses, their ratios, etc. The simulator function, cacheSim, first initializes all variables/structures that have not yet been initialized. There are then two loops, one for instruction caches (which loops over instruction memory addresses) and a second one for data caches (which loops over data memory addresses). For each memory address, the tag and index are extracted and then compared to each cache level to find a hit. If there is a hit, the hits for that particular cache are incremented. Each time any cache has a miss, it increments its misses. If none of the caches get a hit, the data/instruction is fetched from memory into the first-level cache. At the end of all loops, the required calculations for each cache e.g. AMAT are made and output.

Design Choices

Our cache simulates a read-only, direct-mapped, multi-level, instruction and data cache system. The memory addresses are input by the user in the form of file paths for text documents. The user is also allowed to enter different cache values for each level and each category of cache, including choosing the number of cache levels desired. In the case of a complete hit, we chose for the program to fetch the data/instruction into the highest-level cache since that is the convention for most modern CPU architectures. However, something where we did not follow convention, is choosing to make each level split into instruction and data caches instead of only the first level. Since our project is relatively small-scale, we can avoid the complexity and cost issues that are the reasons for only making L1 into data and instruction while benefiting from the split. Separating data and instructions, for us, kept the code more organized in implementation and tracing and, conceptually, it also avoided conflicts and displacements between data and instructions. Our outputs show performance data for each level within each category of cache.

Assumptions

Sequence 1:

values entered in order:

32

100

2

16384

64

5

65536

64

7

32768

64

6

131072

64

8

C:/Users/Haya/Desktop/Cache-Simulator/instructions.txt *[Replace this with your full path for the file but in the exact format]*

C:/Users/Haya/Desktop/Cache-Simulator/test_data.txt *[Replace this with your full path for the file but in the exact format]*

Sequence 2:

values entered in order:

28

80

2

8192

32

2

32768

32
5

16384
64
3

65536
64
6

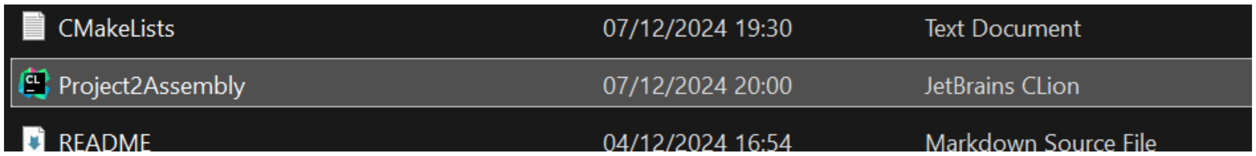
C:/Users/HP/OneDrive/Desktop/instructions2.txt *[Replace this with your full path for the file but in the exact format]*

C:/Users/HP/OneDrive/Desktop/data2.txt *[Replace this with your full path for the file but in the exact format]*

User Guide

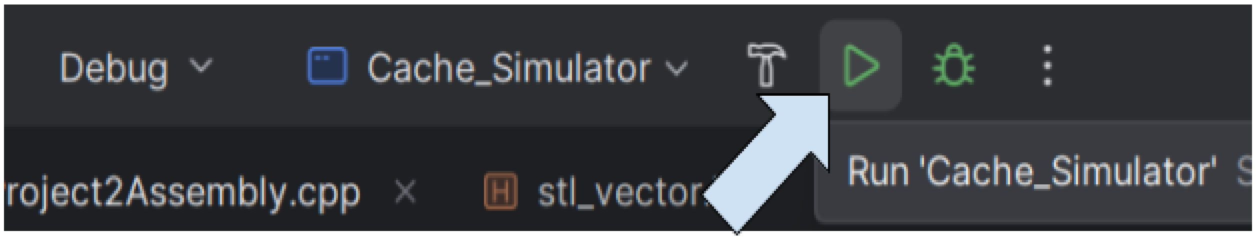
Our code is very simple and easy to use. Follow these steps and screenshots to run and compile it correctly.

1) Download our .zip file and extract all its contents.
2) Locate the ‘Project2Assembly.cpp’ file, and open it in any preferred IDE that supports C++.



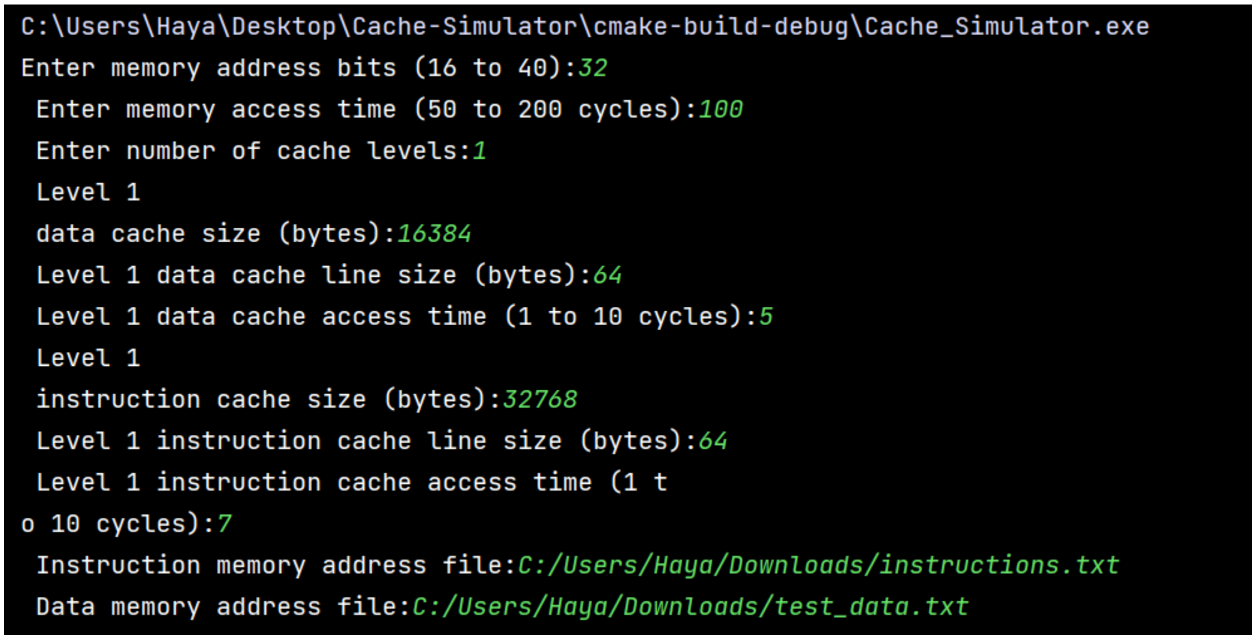
CMakeLists	07/12/2024 19:30	Text Document
Project2Assembly	07/12/2024 20:00	JetBrains CLion
README	04/12/2024 16:54	Markdown Source File

3) Click ‘Run’ and wait for the program to start executing in the IDE terminal.



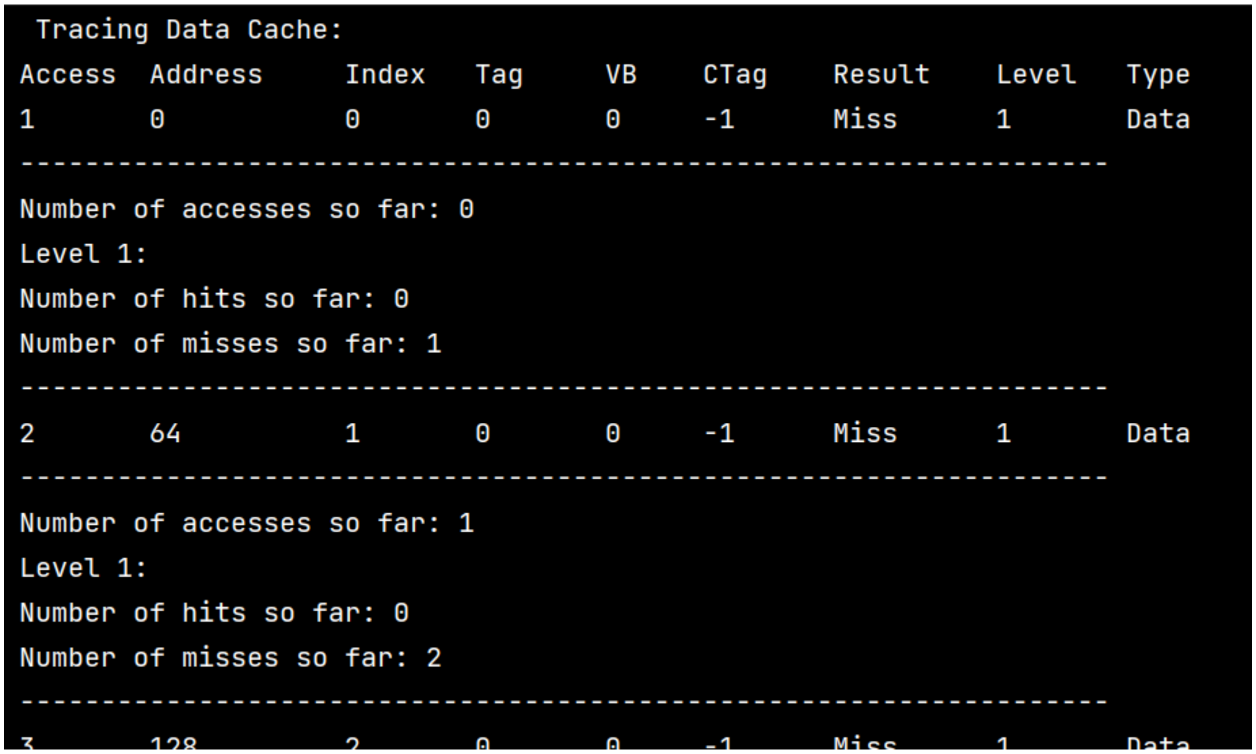
4) You will be prompted for multiple values in this order:
    a) Memory bits
    b) Memory Access Time
    c) Number of Cache Levels
    d) You will then be prompted to enter more values for data and instruction caches for each level:
        i) Cache Size
        ii) Cache Line Size
        iii) Cache Access Time

    Please enter valid values (within the specified ranges and in the specified data types) in order for the program to proceed; otherwise, it will keep prompting you for valid values.



```
C:\Users\Haya\Desktop\Cache-Simulator\cmake-build-debug\Cache_Simulator.exe
Enter memory address bits (16 to 40):32
 Enter memory access time (50 to 200 cycles):100
 Enter number of cache levels:1
 Level 1
 data cache size (bytes):16384
 Level 1 data cache line size (bytes):64
 Level 1 data cache access time (1 to 10 cycles):5
 Level 1
 instruction cache size (bytes):32768
 Level 1 instruction cache line size (bytes):64
 Level 1 instruction cache access time (1 t
o 10 cycles):7
 Instruction memory address file:C:/Users/Haya/Downloads/instructions.txt
 Data memory address file:C:/Users/Haya/Downloads/test_data.txt
```

5) That's it! The code will execute, and all required outputs will be displayed.



```
 Tracing Data Cache:
Access   Address     Index   Tag     VB    CTag    Result    Level   Type
1        0           0       0       0     -1      Miss      1       Data
-------------------------------------------------------------------
Number of accesses so far: 0
Level 1:
Number of hits so far: 0
Number of misses so far: 1
-------------------------------------------------------------------
2        64          1       0       0     -1      Miss      1       Data
-------------------------------------------------------------------
Number of accesses so far: 1
Level 1:
Number of hits so far: 0
Number of misses so far: 2
-------------------------------------------------------------------
3        128         2       0       0     -1      Miss      1       Data
```

Test Sequences

<u>Sequence 1</u>

Part a (data) - 10 accesses: 0,64,128,192,256,64,128,192,256,64 [text_data.txt in Tests folder]

Part b (instructions) - 14 accesses: 0,16,32,48,0,256,272,288,0,16,512,528,544,0 [instructions.txt in Tests folder]

Output: The output was too large to paste here, so please refer to file *Sequence1 Output* in the Tests folder.

<u>Sequence 2</u>

Part a (data) -15 accesses: 0,64,128,192,256,0,320,384,448,0,128,256,512,576,640,0

[data2.txt in Tests folder]

Part b (instructions) -18 accesses:
0,32,64,96,128,160,192,224,256,32,64,96,128,512,544,576,608,32 [instructions2.txt in Tests folder]

Output: The output was too large to paste here, so please refer to file *Sequence2 Output* in the Tests folder.