CSCE2303 – Computer Organization and Assembly Language Programming - Fall 2023

Project 2: Memory Hierarchy Simulator

Names :-

Ismail Fawzy	900204704
Ahmed Allam	900214493
Hussein Elazhary	900211733

Brief about the implementation:-

We have divided the project into three main files which are the main.c file and the Cache manager file with its header file. Now let's break down the cash manager file. First we have the constructor “ CacheManager() ” that contains 5 parameters which are:-

- cacheSize: Total size of the cache in bytes.
- cacheLineSize: Size of each cache line in bytes.
- cacheAccessCycles: Number of cycles it takes to access the cache.
- cacheMode: The mode of the cache (0 for Direct Mapped, 1 for Fully Associative, 2 for m-way Set Associative).
- mWay: The 'm' value for m-way Set Associative cache.

- The constructor basically calculates the number of cache lines based on cacheSize and cacheLineSize. Initializes the instruction and data caches as vectors.

-The second function is the “accessCache(string accessLine)” that Determines the type of cache access (instruction or data) and initiates the cache access process.

It has only one parameter which is “accessLine” which is a string representing a line of access, beginning with either 'I' for instruction or 'D' for data, followed by the memory address.

This basically Extracts the type of access and the address from accessLine. Calls accessCacheInternal() with the appropriate cache (instruction or data) based on the type.

-The third one is “accessCacheInternal(int access, vector<string> &cache)” which basically handles the internal logic of accessing the cache.

This function has only two parameters which are:-

- access: The memory address being accessed.
- cache: Reference to the cache vector (either instruction or data) that is being accessed.

So this function converts the access address to a binary string. Depending on the cache mode, it calculates the tag and index, checks for a cache hit or miss, updates the cache content, and updates statistics like hits and misses.

- Fourthly we have “printStatistics()” which from its name we can indicate that it displays the cache's performance statistics.

It calculates and prints out statistics like the number of hits, misses, total accesses, hit rate, miss rate, and average access time. This function helps in evaluating the efficiency of the cache under the current configuration and access patterns.

- Our last function is the “printcache()”, this function prints the current state of both the instructions and the data caches.

It iterates over the cache lines in both the instruction and data caches, printing out their contents. This function is useful for visualizing how data is stored in the cache and observing the changes in the cache state over time.

So in conclusion, These functions collectively enable the CacheManager to simulate the behavior of a cache in a computer system, including how it stores data, handles different types of accesses, and tracks its own performance. This setup is crucial for understanding the dynamics of cache memory in computing systems, particularly for educational and experimental purposes in the field of computer architecture.

There are no bugs found or issues with the simulator

As for the bonus:-

We made bonus number two and four.

2- Supporting set and full associativity. In this case, in addition to the cache information listed above, the user will need to specify the associativity level.

4- Supporting separate caches for instructions and data. In this case, the user will provide 2 access sequences: One for instructions and one for data.

Design decisions and/or assumptions we made:

- Cache size and cache line size must be a power of 2
- Number of clock cycles to access cache must be between 1, 9
- Memory address is 24 bits in size
- Accessing memory takes 120 cycles

How to compile and run the simulator:

1- Open up any terminal emulator
2- Make sure you are in the project's src directory
3- write this command:

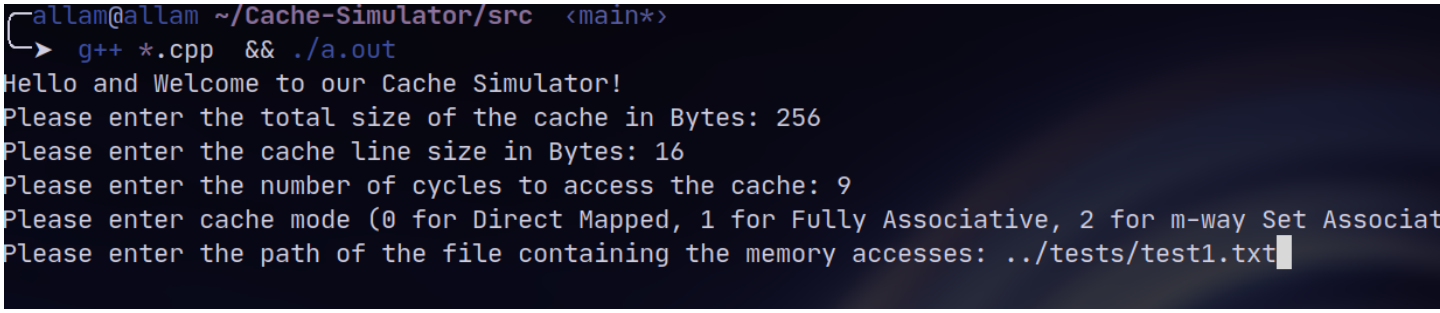
`g++ *.cpp  && ./a.out`

A list of sequences we simulated:

1. D 13 I 7 D 26 D 92 I 15 D 13
   I 7 D 50 I 23 D 26 D 102 I 15
   D 75 D 13 I 7 D 26 I 47 D 50
   I 23 D 92 I 15 D 102

2. I 34 D 58 D 16 I 89 D 34 I 45
   D 21 I 34 D 72 I 58 D 45 I 16
   D 89 I 21 D 34 D 72 I 45 D 16
   D 89 I 72 D 58 I 21

Screenshots of running the program:



```
Starting simulation...
Cache is empty.


========================================

Accessing data cache with address: 13...
Tag: 0000000000000000000
Index: 13
Hit: No


Printing instruction cache...

Cache line 0:
Cache line 1:
Cache line 2:
Cache line 3:
Cache line 4:
Cache line 5:
Cache line 6:
Cache line 7:
Cache line 8:
Cache line 9:
Cache line 10:
Cache line 11:
Cache line 12:
Cache line 13:
Cache line 14:
Cache line 15:
```

```
==============================

Printing data cache...

Cache line 0:
Cache line 1:
Cache line 2:
Cache line 3:
Cache line 4:
Cache line 5:
Cache line 6:
Cache line 7:
Cache line 8:
Cache line 9:
Cache line 10:
Cache line 11:
Cache line 12:
Cache line 13: 00000000000000000001101
Cache line 14:
Cache line 15:


==============================



========================================
```

After running all remaining accesses in the first sequence: