Computer Organization and Assembly Project #2

Students' names and IDs

- Ragab Abbas 900211252
- Hadj Ahmed Chikh Dahmane 900214344
- Youssef Mansour 900212652

description of your implementation

This is a cache simulator that provides a way to model a computer's cache memory. The simulation accounts for different configurations and mapping strategies (Direct, Fully Associative, and Set Associative Mapping), giving a detailed report of hit and miss ratios, as well as the average memory access time (AMAT).

The program starts by declaring variables that will hold important parameters such as cache size, block size, number of lines in the cache, number of cycles, number of hits and misses, and access sequence. The access sequence is a sequence of memory addresses accessed by the CPU, stored in a vector, and obtained from a file specified by the user.

Next, the program defines several utility functions. isValid() checks if a number is a power of two, necessary as many cache parameters (e.g., size, block size, number of lines) are typically powers of two. print2DVector(), print(), and printCache() are used for displaying different aspects of the cache, while intToBinaryString() and binaryToDecimal() are used for conversions between integer and binary string representations. readIntegersFromFile() reads the access sequence from a specified file.

The cacheManager() function is the heart of the program. It interacts with the user, asking for input values for cache parameters. It validates these inputs and calculates secondary parameters like number of bits in a line and block size. It also reads the access sequence file.

The user is then asked to choose the cache mapping type they want: Direct ('d'), Full Associativity ('f'), or Set Associativity ('s'). Based on the user's choice, the appropriate simulation is executed.

For the cache simulation, each access from the sequence is analyzed in binary format. A line number is calculated from the binary address. The simulator checks if the memory access is a hit or a miss by comparing the stored addresses. If a miss occurs, the address is stored in the cache according to the selected mapping strategy.

After simulating all memory accesses, the program reports statistics: the number of hits, misses, and accesses, the hit ratio, the miss ratio, and the average memory access time (AMAT).

bonus features included

- 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.
- Supporting separate caches for instructions and data. In this case, each of the memory addresses in the provided sequence must be labelled as either an instruction or data access.

design decisions and/or assumptions.

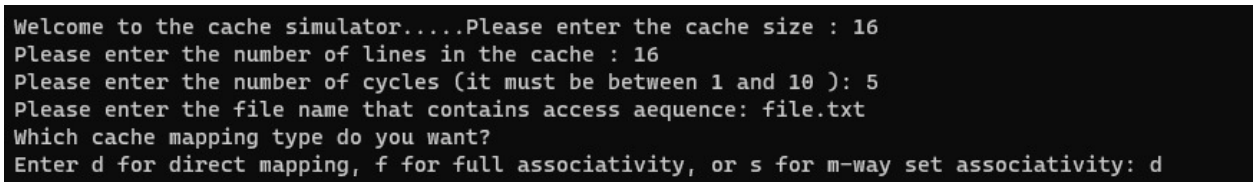
- One hit/miss rate for both caches (data / instructions)
- Memory Access takes 100 clock cycles.
- No space at the end of each line in file
- Any letter required from the user is small.

bugs or issues in your simulator.

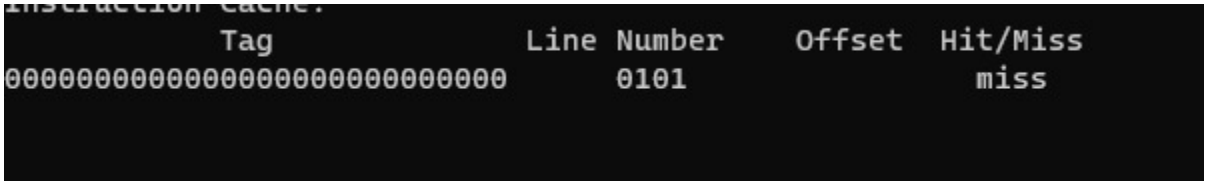
- No bugs

user guide for compiling & running (full simulation example step-by-step) with screenshots.

First we give the inputs at the start as following

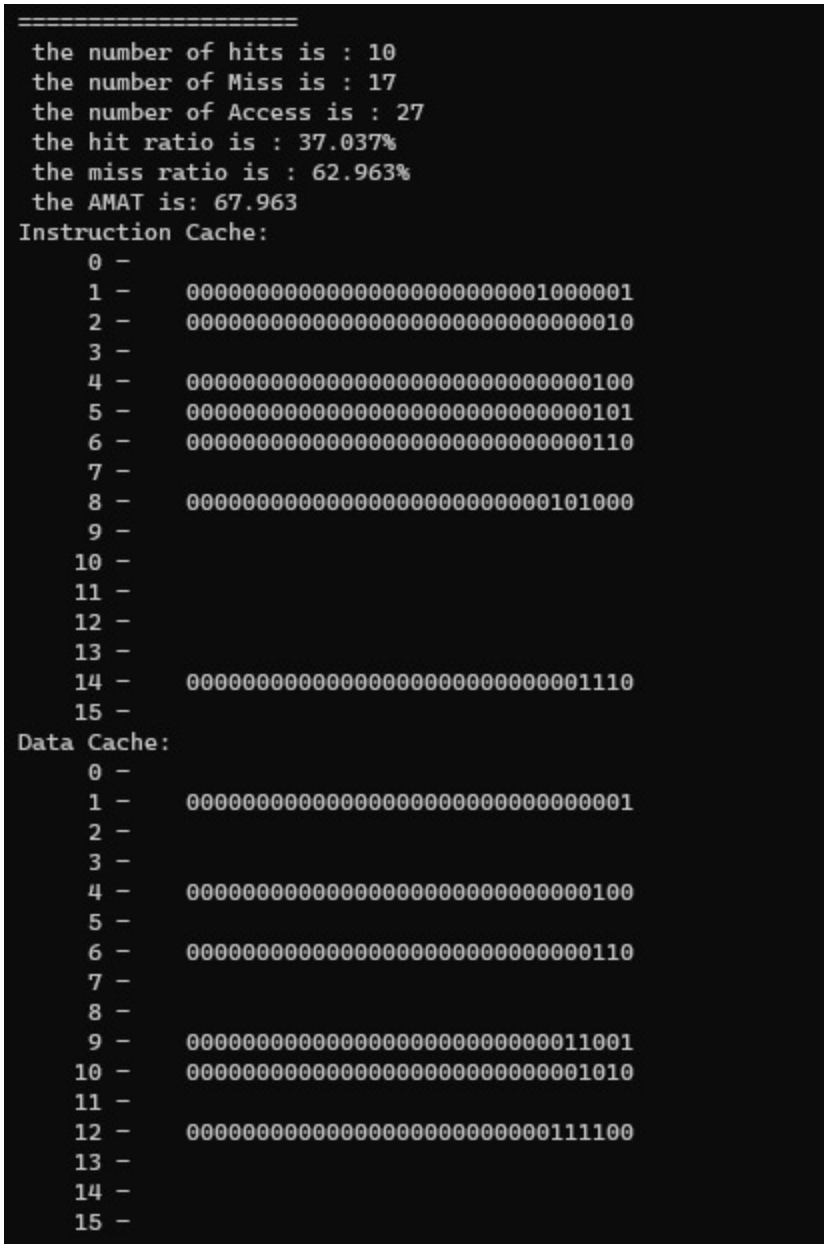


Then the input in th file is divided into sections as following according the the user input at the beginning and Hit/Miss is shown



Then Update (instruction or data) Cache if Miss

This happens for each line in the file.txt



A list of sequences simulated (at least provide two 20-access sequences**)**.

5i, 2i, 4d, 65i, 65d, 85i, 1d, 4i, 37, 6d, 650d, 5i, 10d, 40i, 307, 60d, 804, 40i, 25d, 84, 4i, 25d, 6i, 8g, 4i, 37, 18g, 14i, 137, 18g, 4i, 37, 8g, 4i, 37, 6d, 6d, 84, 4i, 25d

where i means instruction and d means data